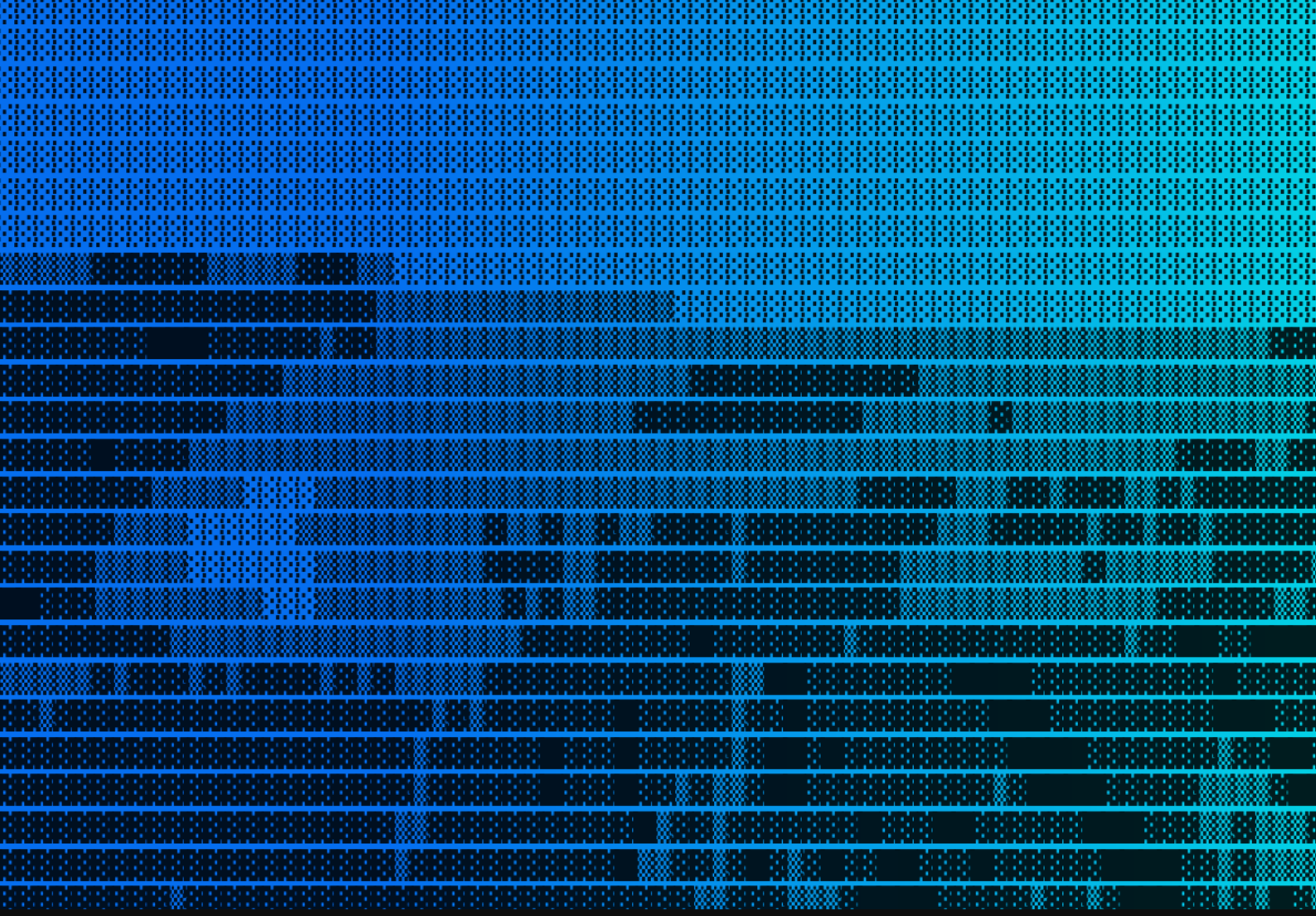


SPECIAL REPORT

Key Considerations for New Java Projects in 2025

www.vaadin.com

vaadin

Executive summary

There are now a dozen or more reports that list technology decisions across the java industry.

Rather than simply tracking which technologies and practices are being used, this report explores the **considerations, challenges, and thought processes behind the decisions that affect project outcomes,** including:

→ Java's role in modern business applications,
→ Key challenges when starting new projects and modernizing Java applications,
→ The reasoning behind evolving team structures, budget decisions, and technology choices.

This report provides insights into the thought processes and considerations affecting the way Java teams are adapting to the changing development landscape. It is designed for everyone planning new projects or modernization projects in 2025.

Java continues to be the backbone of business-critical applications, with organizations actively investing in both new application development and modernization efforts. The shift toward cloud-native architectures is accelerating as businesses look to improve scalability, efficiency, and security, though many still rely on private infrastructure due to security concerns.

At the same time, modernization remains a key priority, with most organizations opting for incremental update strategy to minimize risk while ensuring long-term maintainability. Additionally, full-stack teams are on the rise, potentially transforming development workflows by improving agility and reducing bottlenecks. Specialized roles in QA, DevOps, and UX remain essential for maintaining quality and scalability.

The intent of this report is to support better decision-making for anyone starting a new Java project or modernization project in 2025.

Report highlights

1. **Java in 2025: Keeps driving business software.** 70% of teams plan to start a new Java project in the next 12 months, while over half are modernizing existing systems—showing Java's role is only growing in modern enterprise development.

2. **Modernization is a priority,** with 56% of teams focusing on upgrading legacy Java applications, and most (55%) taking an incremental approach over full rewrites.

3. **Full-stack teams are preferred over decoupled teams** (61%), showing that teams prefer colleagues and technologies that support developing both the front end and back end of applications. Specialized roles in QA, DevOps, and UX remain critical for scalability and maintainability.

4. **56.9% of respondents** reported **50% or greater drop in communication overhead** after moving to a full-stack structure. **Even modest improvements pay off:** A 50% cut in coordination overhead saves **$6,250 per developer annually** (based on a $100K salary).

5. **Cloud-native Java adoption is growing,** but 43% of organizations still use private cloud or on-premises due to security, cost, and compliance concerns.

6. **Upgrading tech stacks (44%) and testing and QA (38%) top the list of Java challenges** —highlighting the importance of choosing tools and frameworks that prioritize maintainability and make upgrades easier.

7. **Spring Boot (86%) dominates Java frameworks,** while React (39.7%) and Angular (26.3%) are the leading choices for the front end.

8. **Collaborative tech decisions lead to better outcomes.** 81% of teams involve developers in proposing technologies, with architects or leads making the final call.

9. **Developer happiness matters.** Choosing tools that are intuitive and enjoyable to use boosts productivity and morale.

10. **Start small, win big.** Successful teams recommend **incremental modernization** over full rewrites. Break it down, and modernize one function or module at a time.

Table of contents

Methodology

This report is based on a global survey of 1,000 technology leaders and practitioners in organizations of all sizes across a wide set of industries. The survey was conducted in February 2025 and remains open. As we reach new respondent milestones, we'll release updated reports and highlight trends over time.

Respondents were Java users sourced from public social media channels as well as Vaadin's database. The respondents are not limited to users of Vaadin's open-source or commercial Java frameworks. Not all respondents answered every question, so the number of respondents for each question is provided in the results.

Java in 2025 Continues to power business software

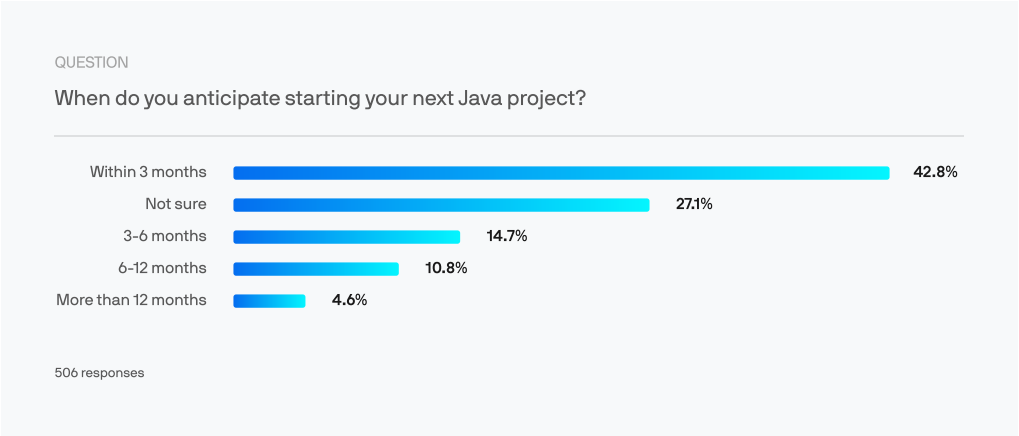
Planning a new Java project this year? You're not alone—and you're in good company. Java might've started in 1996, but in 2025, it's still one of the most trusted choices for building secure, scalable, and business-critical applications.

Whether you're modernizing a legacy system or building something brand new, Java remains a strong foundation. It consistently ranks among the top programming languages globally and continues to evolve to meet the demands of modern software development.

What other teams are doing right now?

In our survey, **70% of respondents expect to start a new Java project within the next 12 months, and over half (55.4%) are focused on modernization.** This tells us that Java isn't just holding its ground—its growing with the needs of modern business.

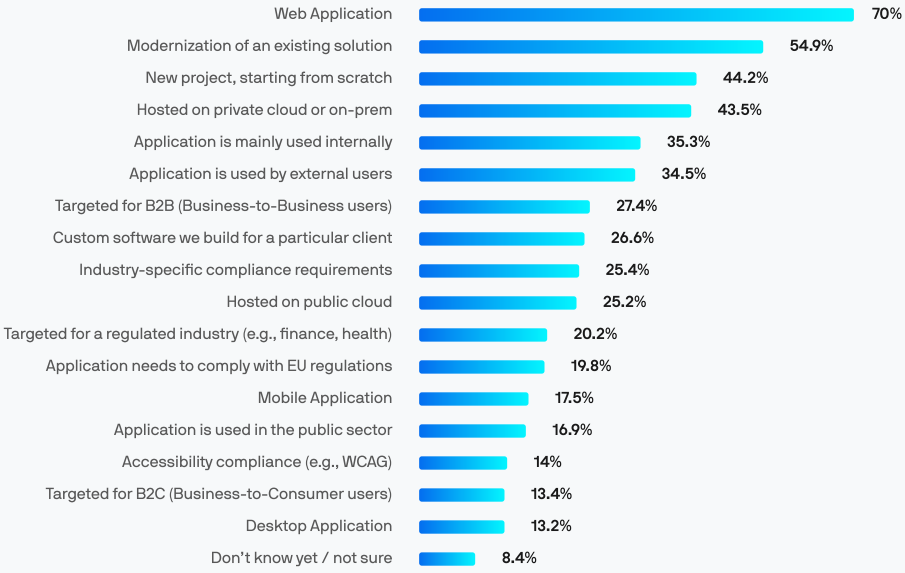
→ 44.1% of respondents plan to start a Java project within the next three months
→ 14.8% are targeting a 3–6 month timeline
→ 10.9% expect to kick off within 6–12 months
→ 44.6% of respondents are building new Java projects from scratch, while
→ 55.8% are focused on modernizing an existing solution.



What other teams are doing right now?

QUESTION

How would you describe your upcoming project?



478 responses

How–and where–your Java app will run matters

As you plan your architecture, think about how your deployment strategy will shape development and delivery. The shift toward the cloud is well underway, but it's not one-size-fits-all.

→ 43.5% of respondents plan to deploy new Java apps on private cloud or on-prem infrastructure
→ 24.8% are targeting public cloud platforms

This trend reflects a strategic mindset: Java teams are choosing cloud models based on security, compliance, cost, and existing infrastructure—not hype.

Questions to answer for your own project

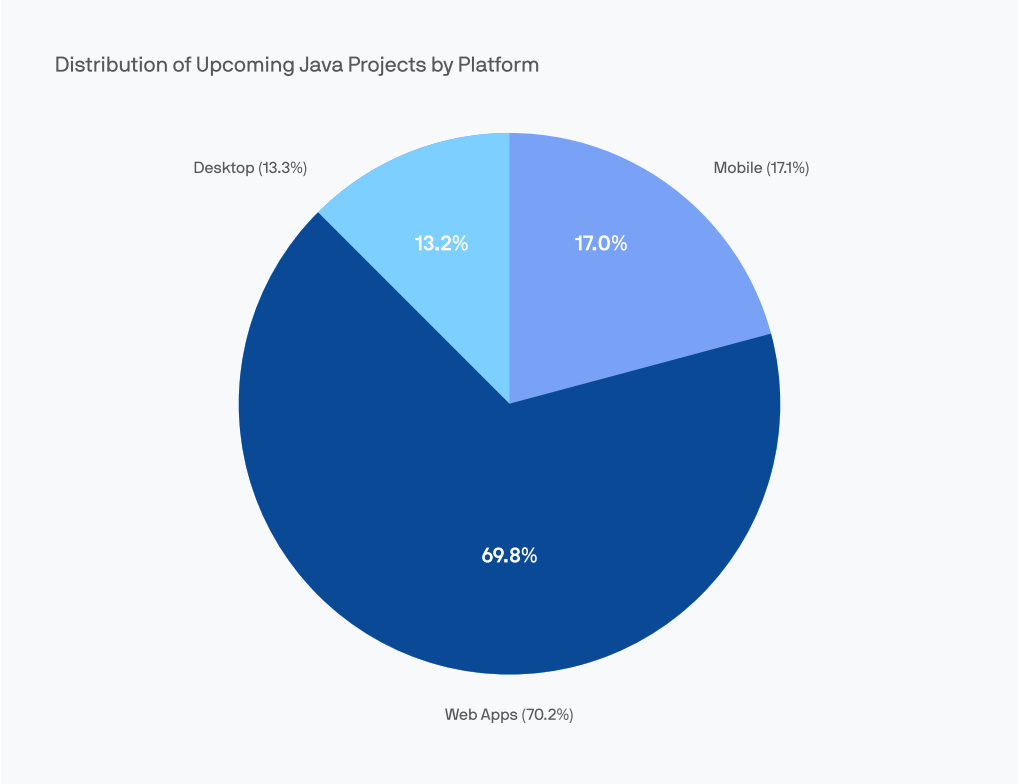
→ Will you deploy to public, private, or hybrid cloud—or keep it on-prem?
→ Are there regulatory or security constraints that shape your hosting options?
→ Does your team already have cloud infrastructure in place, or are you starting from scratch?

The best deployment choice is the one that fits your long-term goals, not just your launch date.

Web applications lead the way

If you're unsure which type of app to build, the data is clear:
Web applications dominate Java's future. According to our survey:

→ 70.2% of upcoming Java projects are being built as web apps
→ Compared to just 17.1% mobile and 13.3% desktop



Why? Web apps offer easier deployment, faster updates, and cross-platform support—making them an ideal default for internal tools, customer portals, and B2B services.

Questions to answer for your own project

→ What platforms do your users expect to access your application from?
→ Could a modern web UI or PWA meet your needs without requiring native
mobile or desktop development?
→ Are you building for users with consistent internet access—or do you need offline capabilities?
→ Are you making UI choices based on user experience—or development familiarity?

All Vaadin UI components are built and tested to meet WCAG 2.1 AA standards, with support for screen readers, keyboard navigation, and semantic HTML out of the box. Whether you're preparing for upcoming regulations like the European Accessibility Act (2025) or just building inclusive apps from the start, Vaadin helps you stay compliant—without extra work.

What are you building –and who are you building it for?

Java is powering everything from internal business tools to customer-facing apps and public-sector systems. Here's how new projects break down:

- → **35.3%** are building internal enterprise tools
- → **34.5%** are focused on customer applications
- → **27.1%** are creating B2B solutions
- → **17.1%** are for the public sector
- → **13.5%** are building B2C products

This range shows just how versatile Java still is—no matter who your users are.

Considerations for starting a new Java project

As you start scoping out your project, here are some helpful questions to guide your planning:

→ Are you launching a greenfield project or modernizing something existing?
→ What specific business outcome is driving this initiative?
→ What technical debt or outdated systems do you need to address first?
→ What does success look like for you in the first 3,6, and 12 months?
→ What team structure will help you achieve he best results?
→ How will you budget for this project within your organization?

And just as importantly—who are you building for?

→ Who are you building for—internal teams, customers, B2B partners, or the public?
→ What specific needs or constraints does your primary audience introduce?
→ For internal tools: How will your app streamline workflows or reduce inefficiencies?
→ For customer-facing apps: What kind of user experience are you aiming to deliver?

→ For B2B: How will you manage integration complexity, SLAs, or high-availability needs?
→ For B2C or public-sector apps: Are accessibility, localization, or compliance planned from the start?
→ How does your audience influence your decisions around UI, scalability, and deployment?

Thinking through your priorities now will help your team align strategy, timelines, and technology from day one - and we'll share the thought processes of our respondents across many of these areas.

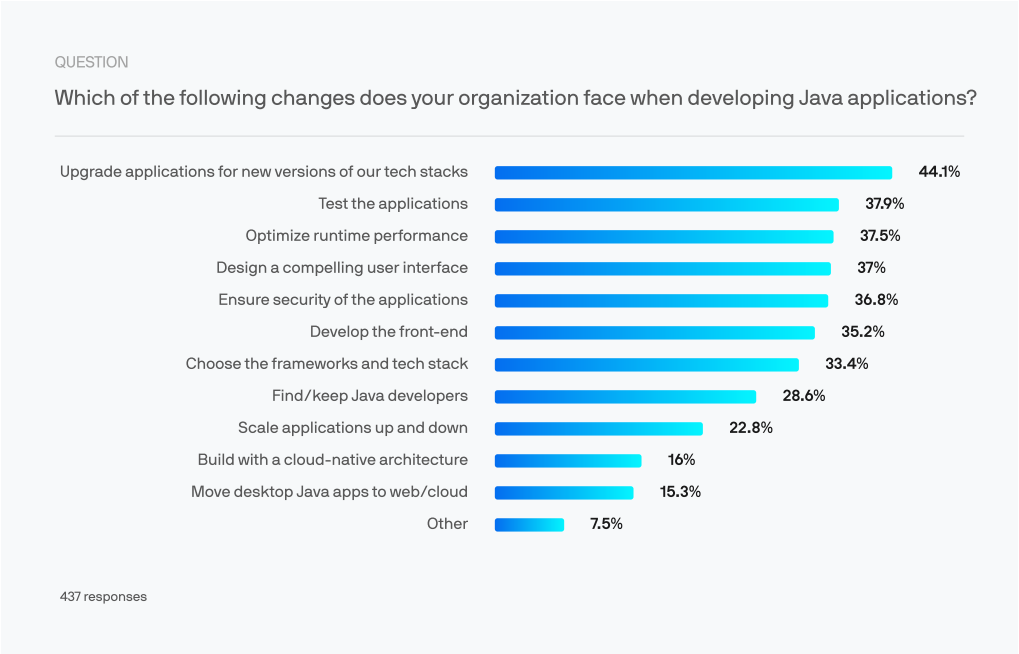
Key challenges in Java development

As we've seen, Java remains a reliable foundation for business-critical applications—but that doesn't mean development is without its challenges. Teams today face a wide range of hurdles that span both backend and frontend concerns.

According to our survey, the most commonly reported challenges include:

- → Upgrading applications for new tech stack versions – 44.1%
- → Testing and quality assurance – 37.9%
- → Performance optimization – 37.5%
- → UI/UX design – 37%
- → Security and vulnerability management – 36.8%
- → Frontend development – 35.2%
- → Framework selection and architectural decisions – 33.4%

> "Where you start testing depends on the risk—some apps benefit from 'outside-in' testing to de-risk service integrations, others from 'inside-out' focusing on complex business logic."
>
> 
> Josh Long,
> Spring Developer Advocate



These challenges span both backend and frontend domains, highlighting the complexity of delivering modern applications. While some are technical (like performance or upgrades), others relate to user experience and decision-making, such as UI/UX and framework selection.

Considerations for tackling Java development challenges

If you're starting a new Java project or modernizing an existing one, these challenges are a valuable reference point for what to prepare for—and what to actively plan around.

Consider the following:

→ What challenges is your team most likely to face—and are you prepared to meet them?
→ How aligned are your developers and engineering leaders when it comes to technical decisions and business priorities?
→ How are you planning for tech stack upgrades, including frameworks and third-party dependencies?
→ Do your developers have what they need to deliver on leadership's expectations—or will you need to bring in new skills, tools, or support?
→ What's your team's approach to frontend development, and do you have the skills to deliver modern, responsive UIs?

The takeaway?

To succeed in 2025, Java teams must balance modern tooling, great user experience, and long-term maintainability. The most effective teams align around shared priorities and empower developers to move fast, without compromising on quality or security.

Considerations on team structure & full-stack development

As development cycles accelerate and flexibility becomes more valuable, choosing the right team structure is one of the most important decisions for your Java project. While the right setup depends on your goals, your people, and the complexity of your application, full-stack teams are emerging as a favored approach.

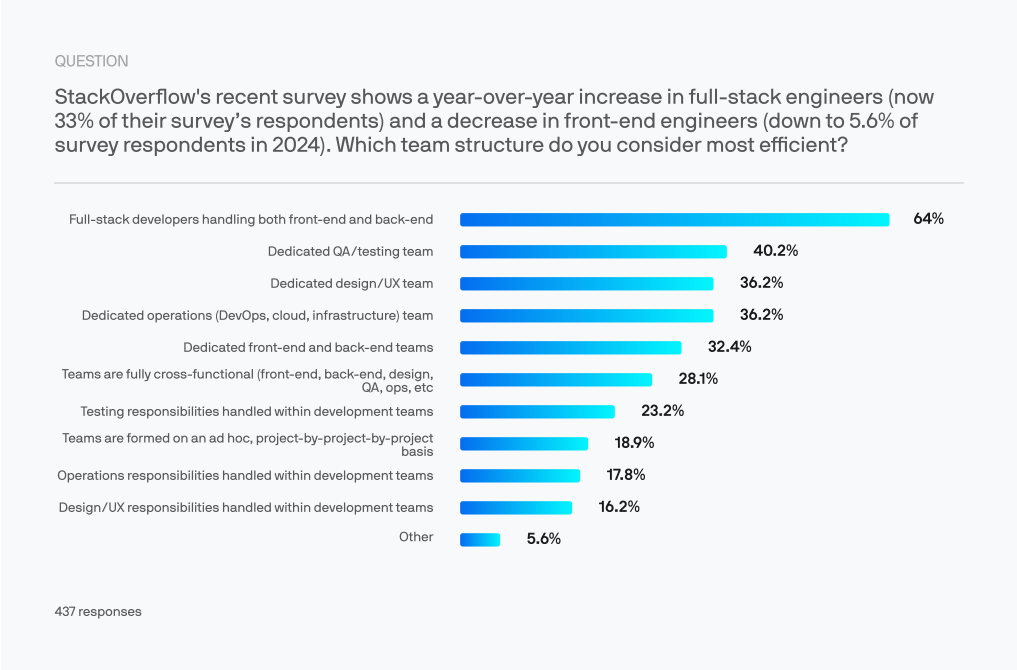
In this section, we'll explore the rise of full-stack roles, their benefits and challenges, and how developers and leaders view team structure differently.

> "Frontend complexity has spiraled. It doesn't surprise me at all that teams are moving toward unifying the stack and simplifying delivery."
>
> 
> Josh Long,
> Spring Developer Advocate

The rise of full-stack teams: Developer perspective

According to our data, **64% of Java developers see full-stack teams as the most efficient structure for handling both frontend and backend work.** This trend aligns with broader industry patterns, including Stack Overflow's findings on the growing demand for full-stack roles. It shows that teams are looking for members who can handle both front and back end, as well as technologies that support both.

Frameworks, like Vaadin Flow and Hilla, enable developers to build responsive, modern web UIs entirely in Java or in combination with Java and React. This approach can be especially useful for business applications that require strong UX and benefit from a structured component model, while also reducing frontend complexity for full-stack teams.



While full-stack ensures that front and back end deliverables are built by the same people, other roles and teams support critical functions:

→ 40.2% for QA/testing
→ 36.2% for DevOps
→ 36.2% for UX/design

These roles are seen as essential for maintaining quality, scalability, and user experience.

Team structures also vary:
→ 32.4% prefer dedicated frontend and backend teams
→ 28.1% support fully cross-functional teams
→ 23.2% believe testing should be integrated into development
→ 18.9% think operations should be embedded in dev teams

If you're building a new team, the takeaway is clear: full-stack is gaining traction. Other roles are incredibly important to the overall success of a project, but if you're starting a new team and you have the option – it's worth exploring whether building a fullstack team is right for you.

"Don't compromise on QA and DevOps. Similarly, programmers are valuable resources—avoid distracting them with non-core tasks that take away from their primary focus."

- VP, C-level, (Name and company withheld at the request of survey respondent) Technology/IT Services/Consulting

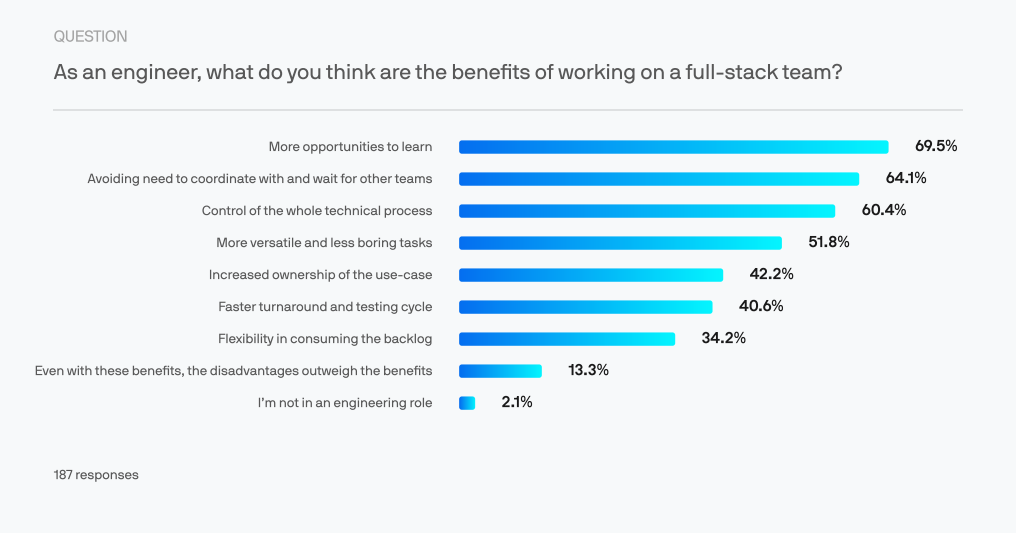
"The choice of team structure depends on project needs: a full-stack team is more agile, while a front-end/back-end split allows for deeper expertise. Clear communication is essential to avoid silos and ensure consistency. Testing different approaches and adapting to the company's constraints is often the best strategy."

- Damlabin Nkamneu Arsene Vivien, Full-stack developer @ Université Laval

Why developers choose full-stack: Learning, speed, and ownership

When asked about the benefits of full-stack roles, developers pointed to key advantages that influence both team dynamics and project outcomes:

→ **Broader skill development:** 70% appreciate the opportunity to work across the stack and grow professionally
→ **Faster development cycles:** 64% believe full-stack teams reduce bottlenecks by limiting team-to-team dependencies
→ **Greater control & ownership:** 60% value making decisions across the full stack without frontend/backend friction
→ **More engaging work:** 52% find full-stack work more varied and less repetitive
→ **Increased efficiency:** 41% report faster turnaround times, while 34% benefit from more flexible backlog handling



“A full stack developer team plays a vital role in the success of any project, particularly when utilizing Vaadin. Vaadin is a powerful framework that simplifies the development of user interfaces by handling many frontend components. This allows developers to focus on creating robust backend systems while ensuring a seamless and responsive user experience. By leveraging the strengths of both the frontend and backend, a full stack developer team can effectively deliver high-quality applications that meet the needs of users and stakeholders alike.”

- Barış Şahin,
  Full-stack developer @ License4J

"I think full-stack is the way to go. But let’s be honest—no one can do everything. Most people should have a foot in each world, but also work with teammates who complement their skills.”

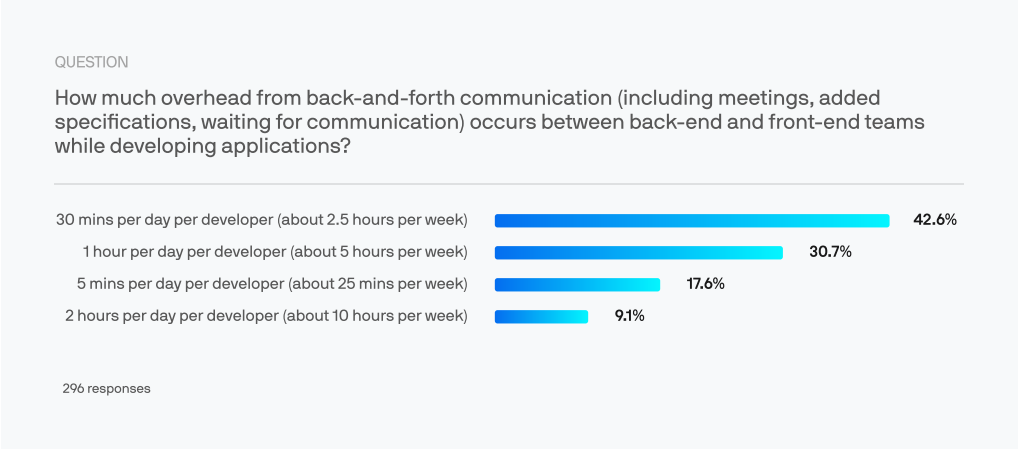
Josh Long,
Spring Developer Advocate

Communication overhead: A shared challenge for developers and leaders

Whether you're leading a project or working hands-on in the code, one reality often surfaces: communication between frontend and backend teams can quietly eat away at development time.

In our combined survey of developers and engineering leaders, this challenge came through clearly:

→ 42.6% of respondents said each developer loses about 30 minutes per day (roughly 2.5 hours per week) to back-and-forth communication
→ 30.7% reported that it's closer to 1 hour per day—equating to a full workday per developer each week
→ 17.6% experienced minimal overhead (around 5 minutes daily)
→ 9.1% said developers spend 2 hours daily coordinating between roles



This includes time lost to meetings, waiting for specs, chasing clarifications, and context-switching between teams—all of which slow delivery and increase cognitive load.

If your current team setup relies heavily on separate frontend and backend roles, this type of friction may already be impacting your velocity—whether or not it's being formally tracked.

"A high level of separation between front-end, back-end, and testing can introduce significant overhead, such as meetings and communication challenges. At the start of a project, having a team of full-stack developers can be highly effective for quickly building a solid core foundation. This approach allows for faster decision-making and reduces coordination bottlenecks/failures. Once the core is established, transitioning to specialized teams for each part of the application can be beneficial, as it enables deeper expertise and optimization in specific areas."

- **Full-stack developer,**
  Custom Software Development/IT Consulting

"It's not just about roles—when you separate frontend and backend teams, the lack of communication creates friction. Full-stack teams cut that overhead significantly."

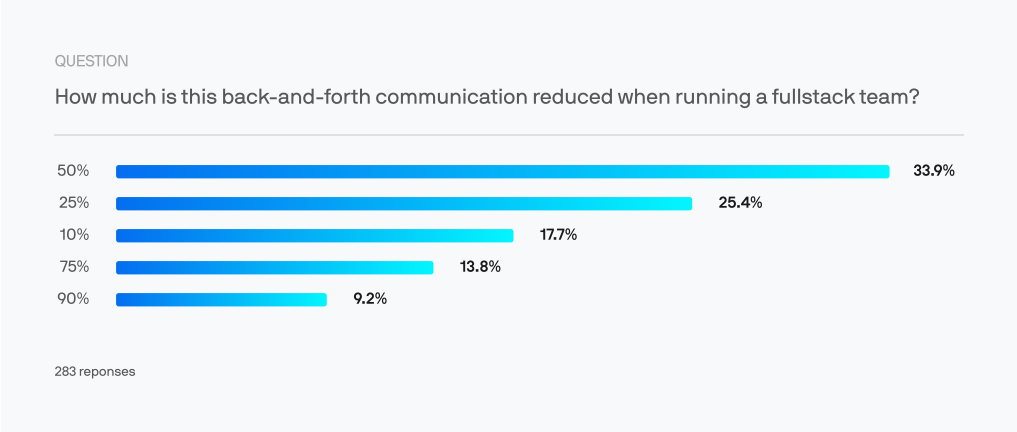
Josh Long,
Spring Developer Advocate

How much can full-stack teams reduce communication overhead?

The good news: many teams report meaningful improvements when shifting to a full-stack structure. When we asked developers and leaders how much communication overhead is reduced by going full-stack, here's what they said:

- → 33.9% said the time spent on coordination drops by 50%
- → 25.4% reported a 25% reduction
- → 17.7% saw a smaller drop of around 10%
- → 13.8% said overhead is reduced by 75%
- → 9.2% saw a dramatic 90% reduction

In total, **56.9% of respondents** reported a **50% or greater drop in communication overhead** after switching to a full-stack structure.



While results may vary by team size and workflow, it's clear that consolidating roles can lead to fewer handoffs, less waiting, and faster feedback loops. If your project is already seeing slowdowns due to cross-role dependencies, rethinking your structure with a full-stack model or hybrid approach could offer a measurable efficiency boost without sacrificing quality.

The cost-based case for full-stack development teams

The majority of developers reported losing 30 minutes to 1 hour per day on coordination tasks—things like chasing specs, waiting for answers, and switching context between frontend and backend teams. That's 2.5 to 5 hours per week per developer, or up to 12.5% of their total working time.

The good news? Teams that have moved to a full-stack setup report meaningful improvements:

→ 33.9% cut coordination time by 50%
→ 25.4% reduced it by 25%
→ 13.8% saw a 75% reduction
→ Some even reported up to 90% savings

Let's put that into real numbers.

Assuming an average developer salary of $100,000/year, just a 50% reduction in coordination overhead (from 5 hours to 2.5 hours per week) saves roughly:

$6,250 per developer, per year

And that's **just** for time recovered.
When you factor in the total cost of a developer—including tools, taxes, management, and overhead—the real savings could easily approach **$10,000 per developer, per year.**

The takeaway? Switching to a full-stack team structure doesn't just streamline delivery—it can unlock thousands in lost time and costs per developer, every year.

According to the 2024 Stack Overflow Developer Survey, the median salary for backend developers was $67,227, while full-stack developers earned a similar $63,332.

With nearly identical salary levels, shifting to full-stack roles can boost delivery speed and reduce coordination overhead—without increasing payroll costs.

"Depending on the size of your application, having a full-stack team (Front to Back to DevOps) can give you way faster results than a compartmentalized team."

- Salvador Pardiñas,
  Team lead @ Antel

"A full stack developer team plays a vital role in the success of any project, particularly when utilizing Vaadin. Vaadin is a powerful framework that simplifies the development of user interfaces by handling many frontend components. This allows developers to focus on creating robust backend systems while ensuring a seamless and responsive user experience. By leveraging the strengths of both the frontend and backend, a full stack developer team can effectively deliver high-quality applications that meet the needs of users and stakeholders alike."

- Barış Şahin,
  Full-stack developer @ License4J

Challenges: Complexity, specialization, and scalability

Of course, full-stack development also comes with its share of challenges—especially as projects grow more complex:

→ **Lack of deep expertise:** 54% say it's difficult to master both frontend and backend technologies
→ **Keeping up with evolving tech:** 47% find it hard to stay current across the entire stack
→ **Increased cognitive load:** 37% feel overwhelmed by the breadth of responsibility
→ **Hiring and onboarding difficulties:** 35% say finding qualified full-stack developers is tough, and 32% report longer onboarding times
→ **Scalability and code quality risks:** 29% warn of knowledge silos, inconsistent practices, and tech debt buildup

"The choice of team structure depends on project needs: a full-stack team is more agile, while a front-end/back-end split allows for deeper expertise. Clear communication is essential to avoid silos and ensure consistency. Testing different approaches and adapting to the company's constraints is often the best strategy. "

- Damlabin Nkamneu Arsene Vivien,
  Full-stack developer @ Université Laval

"Look at the complexity of the project. If it's small enough, use full stack. If it gets to complex, look at dedicated front and back devs. Always include a UI/UX person. Don't over-engineer from the start, but keep possible growth problems in view."

- Full-stack developer

If your project is large-scale or requires high levels of UI polish or infrastructure control, you may need to balance full-stack roles with specialized support to avoid overloading your developers.

Full-stack teams: A leadership perspective

From a leadership standpoint, full-stack teams are often viewed as a strategic move toward agility, faster delivery, and leaner coordination. According to our survey of engineering managers:

→ **62.6%** cited clear responsibility and stronger ownership over deliverables

"Make sure you interlace the responsibilities and clearly communicate them so that nobody can push the responsibility away or to the next guy. Always make sure no silos or build and very team knows the goal of the project. The goal always being the business case."

- **Chris Erlich,**
  VP, C-level @ yellowhippy.com

"Define clear roles and responsibilities for dedicated teams, but there must also be collaboration across teams - this can be greatly aided by your project management/planning tool of choice."

- **Alex Wallon,**
  Full-stack developer, Custom Software Development/IT Consulting

→ **48.5%** said full-stack teams help achieve faster time to market

"Think about the development process, software lifecycle and how you can work efficiently. A cohesive team can be the most effective."

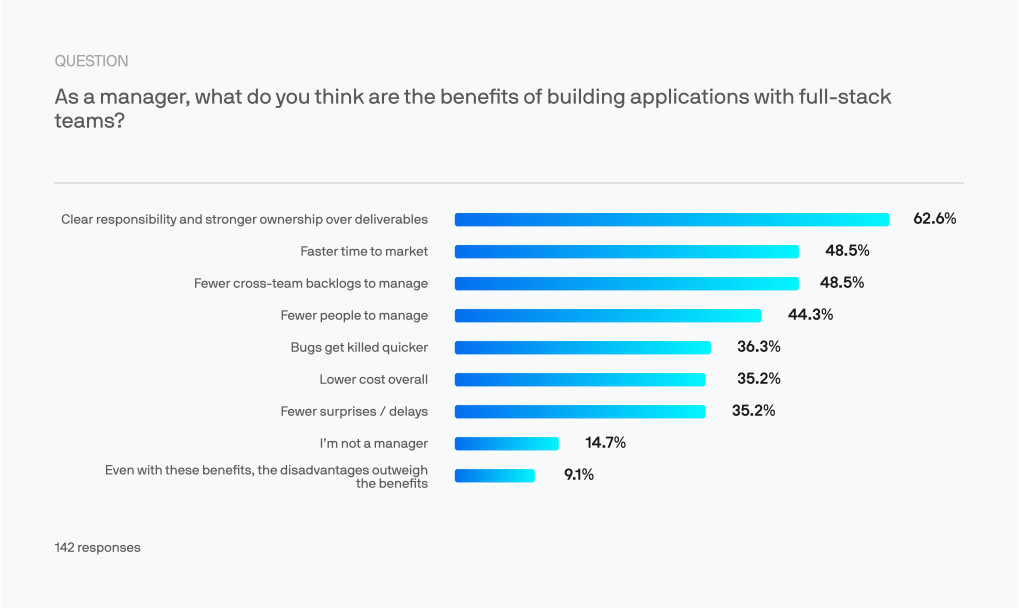
- **Architect,**
  Custom Software Development/IT Consulting

→ **48.5%** also pointed to fewer cross-team backlogs to manage
→ **44.3%** appreciated having fewer people to manage overall
→ **36.6%** reported that bugs get resolved more quickly
→ **35.2%** said they experience fewer surprises or delays
→ **35.2%** saw lower overall costs as a benefit

Only **9.1%** felt that, despite the benefits, the disadvantages outweigh them.

"Depending on the size of your application, having a full-stack team (Front to Back to DevOps) can give you way faster results than a compartmentalized team."

- **Salvador Pardiñas, Team Lead, Antel**



"Go with the structure that best fits your team's skills, project complexity, and company culture. A cross-functional team is great for speed and innovation, while a component-based team can work well for large, specialized systems. No one-size-fits-all—opt for what keeps your team productive, collaborative, and delivering value efficiently."

- **Shaukat Mahmood Ahmad, VP, C-level @ GuCherry Blog by Everestthemes**

The top 5 challenges leaders see in full-stack teams

We know that the benefits are real. It's important to think of ways to address the challenges, so that teams get the most from fullstack development. The top five concerns reported by managers include:

1. **Lack of deep expertise: 49.1%** are concerned that developers can't go deep enough in either frontend or backend, which can impact architectural quality and long-term maintainability.

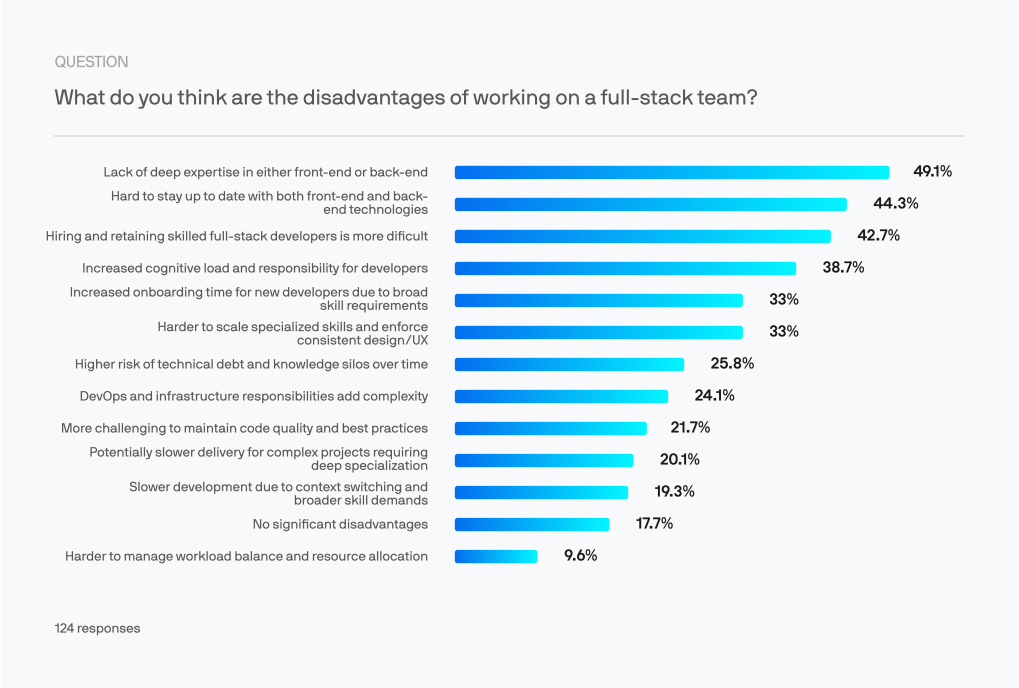
Building on a full-stack platform like Vaadin provides sensible defaults, automates communication, and ensures that all the layers of the stack work seamlessly together.

2. **Staying up to date: 44.3%** say it's difficult for developers to keep up with rapid changes across both ends of the stack.

3. **Hiring and retention: 42.7%** report it's harder to find and keep developers with strong full-stack skills.
   (As the survey creators, we now realize that we should have separated hiring and retention into two separate answers, as each can be resolved with different solutions, and knowing which problem is greater would be interesting!)

4. **Cognitive load: 38.7%** are concerned about the mental overhead that comes with juggling multiple domains and responsibilities.

5. **Longer onboarding: 33%** note that onboarding new developers takes more time due to the broader scope of required knowledge.



If you're planning a full-stack structure, consider how your onboarding, training, and internal support systems will help developers succeed across such a broad surface area.

"If the complexity of your problem allows, try to keep the team as full-stack as possible. This approach provides flexibility, and with staffing becoming easier, recruiting developers with multiple specializations is not as challenging as it was a few years ago."

- **Artur Skowronski, VP, C-level @ Virtus Lab**

Considerations for structuring your Java team

When you're planning a new Java project, team structure will shape your development speed, technical quality, and communication dynamics. Ask yourself:

→ What structure best fits your team's strengths and your project's complexity?
→ Do your developers want to work full-stack—and are they ready for the breadth of responsibility?
→ Is the team using a full-stack development platform that takes care of the details that would otherwise require in-depth expertise?
→ Are backend-focused Java developers equipped to handle frontend technologies and UX expectations?
→ How much communication overhead does your team currently face—and is it slowing you down? Could you do a brief survey on your teams to discover the answer to this?
→ Do you know anyone currently leading, or working on a fullstack team that you could speak with for advice?

At the end of the day, you want to launch, develop, and maintain projects that deliver results for your organization. Team structure is one variable that you have control over, and while it affects budget, team size, and speed, it is also affected by development culture and patterns within your organization.

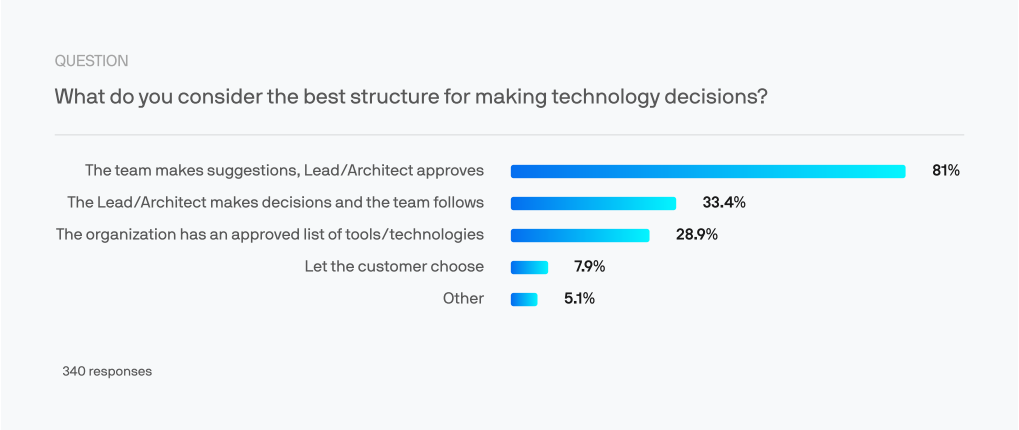
Technology decision-making in Java projects

Right, so you're about to start a new project. You've got some functionality planned, and it's time to choose what tech you'll use to deliver it. Choosing the right technology stack is one of the most consequential decisions in any Java project. It shapes developer productivity, long-term scalability, and how fast you can deliver value. But just as important as what tools you choose is why and how you choose them.

Our survey reveals that most organizations take a collaborative approach to tech decision-making—blending developer input with leadership oversight.

How technology decisions are made:

→ 81% of respondents favor a collaborative model: developers propose technologies, and leads or architects provide final approval.
→ 33.4% follow a top-down model, where the Lead or Architect makes decisions unilaterally.
→ 28.9% rely on a company-approved technology list.
→ 7.9% say customers influence their tech choices.



What matters most in tech decision-making?

When asked what makes a good decision-making model for the technology used in new Java projects, respondents pointed to five recurring themes:

1. **Developer experience & satisfaction** – Choosing tools that feel intuitive and enjoyable can boost productivity and morale, whether familiar or new.

"It's better to have a bottom-up approach of developers being free to choose whatever tools they need for the job, as they're the experts on it. A top-bottom approach only means that managers and architects end up being a blocker on the developer's path."

- Anyul Rivas,
  Team lead, Software/SaaS/Cloud Solutions

"Company-approved list tends to be too rigid and out-of-date. Customer-dictated dev tools even more so. Customer shouldn't need to care what tech is used - as long it meets their needs."

- Architect,
  Government/Public Sector

"Big old companies may have some outdated, like 10-15 years, tech stacks, especially on the front end. Either you care about the tech stack or you don't, find someone who cares and who have to live with it."

- Mikael Fill,
  Team lead

"Team member Bob always wants to use the newest frameworks - and sometimes that's the right move, like when he wanted to start using an opinionated framework that allowed us to simplify cross-app development. Other times, it can be a matter of hopping on the next trend and discovering the grass wasn't really greener - which is where the architect comes in."

- Alex Wollan,
  Full-stack developer @ Kos-mos Solutions

"Sometimes giving the engineers the ability to choose, keeps them happy. Now, let's not get the 0.0.1-BETA version of something. But I'll choose for happy devs over many things."

- AJ Pahl,
  Backend developer @ Bespin Engineering

2. **Impact on scalability & efficiency** – Tech choices directly affect long-term growth and performance.

"Technology decision making is complex and has immense implications to success and sustainability of the organization/ project."

- Architect,
  Technology/IT Services/Consulting

"In a big organization with 100s of engineering teams it simply leads to chaos if teams can decide on frameworks. On the flip side the decision makers in large organizations tend to be so far removed from the technology that decisions are not technically sound and leadership then struggle to understand why the organization has reduced productivity."

- Petrus Viljoen,
  VP, C-level @ Booking.com

"Total freedom does not work; the company ends up with many tools and frameworks. And when the people that thought it was a good idea leave, you still have to maintain it (experienced that). And you maintain software longer than you write it. But locking everything down stiffles innovation (experienced that as well). So the sweetspot is in the balance."

- Tom Eugelink,
  Full-stack developer @ Softworks

3. **Lead/Architect oversight** – Leadership plays a key role in ensuring alignment with business and technical goals.

"Normally, those that develop an application have a good gut instinct for the technology stack that works for them. However, sometimes there is a need to try to use new technologies that might conflict with the customers' needs. Therefore, have an architect that has the final say in those matters really is a good choice."

- Architect,
  Finance/Banking/Insurance

"The people making decision have more experience and knowledge. The best structure is: the lead/architect make decision with a consultation with the team."

- Backend developer,
  Retail/E-commerce/Consumer Goods

"As an Architect your only actual "boss" and commanders should be the requirements. Not the fashion, not the popularity. The requirements, the cost and the maintainability should be the mantra on making your technology decisions. Experienced teams, with a competent Architect Department following the Organization Tech-Culture and operative reality would be my recommended approach. Every technology decision must be linked to the business, market, culture and financial organization goals."

- Bladimir Rondon,
  Architect @ Snap Finance LLC

"The best structure is an open, flat structure where everybody can suggest ideas and improvements, but that still has a clear lead/architect role that provides leadership, standards and guidelines, and takes responsibility for decisions."

- Architect,
  Custom Software Development/IT Consulting

4. **Team involvement** – Encouraging input fosters motivation, ownership, and better adoption of chosen technologies.

"Involving the team keeps the motivation up. If the technology is given from above, the frustration raises. Getting the technology from above or outside should only be used when it's needed by the companies or client needs."

- **Team lead,**
  Custom Software Development/IT Consulting

"Architects usually don't do any development. So I think developers should have their voice when choosing technologies."

- **Veselin Perovic,**
  Full-stack developer @ Dokelldigital

"Ensure the organization has a list of approved stack, based on use case, and enable the entire team to contribute to the approved list, but the lead should make decision on the approved stack."

- **Architect,**
  Software/SaaS/Cloud Solutions

"The team's opinion is crucial and should be taken into account, even if the final decision is taken by the lead/architect. Because they would give their opinion based on their experience, knowledge, or preferences. But of course, the most experienced person on the team should have the last word."

- **Sergio Fernandez,**
  Full-stack developer @ Querry S.A.

5. **Standardization vs. flexibility** – Balancing consistency with adaptability prevents stagnation while allowing innovation.

"Tech decision must be made in a structured way to remove subjectiveness in decision making. A clear assessment has to be made. Making decision be made with the team may lead to teams making decision that are favourable to their core competences at that moment, but this may change as teams change which can lead to technology debt."

- Victor Tichayana Hokonya,
  VP, C-level @ okzim.co.zw

"A structured decision-making process helps mitigate the risks associated with other approaches by providing a clear, objective, and inclusive framework for evaluating technology options. This ultimately leads to better alignment with business goals and more successful technology implementations."

- Tariq Khan,
  Full-stack developer @ Elite Software Tech

"I thinks the decision making process is highly situational. It should be discussed with all parties that are willing to take the responsibility for the decision."

- Team lead,
  Software/SaaS/Cloud Solutions

Consideration: How will you make tech decisions in your Java project?

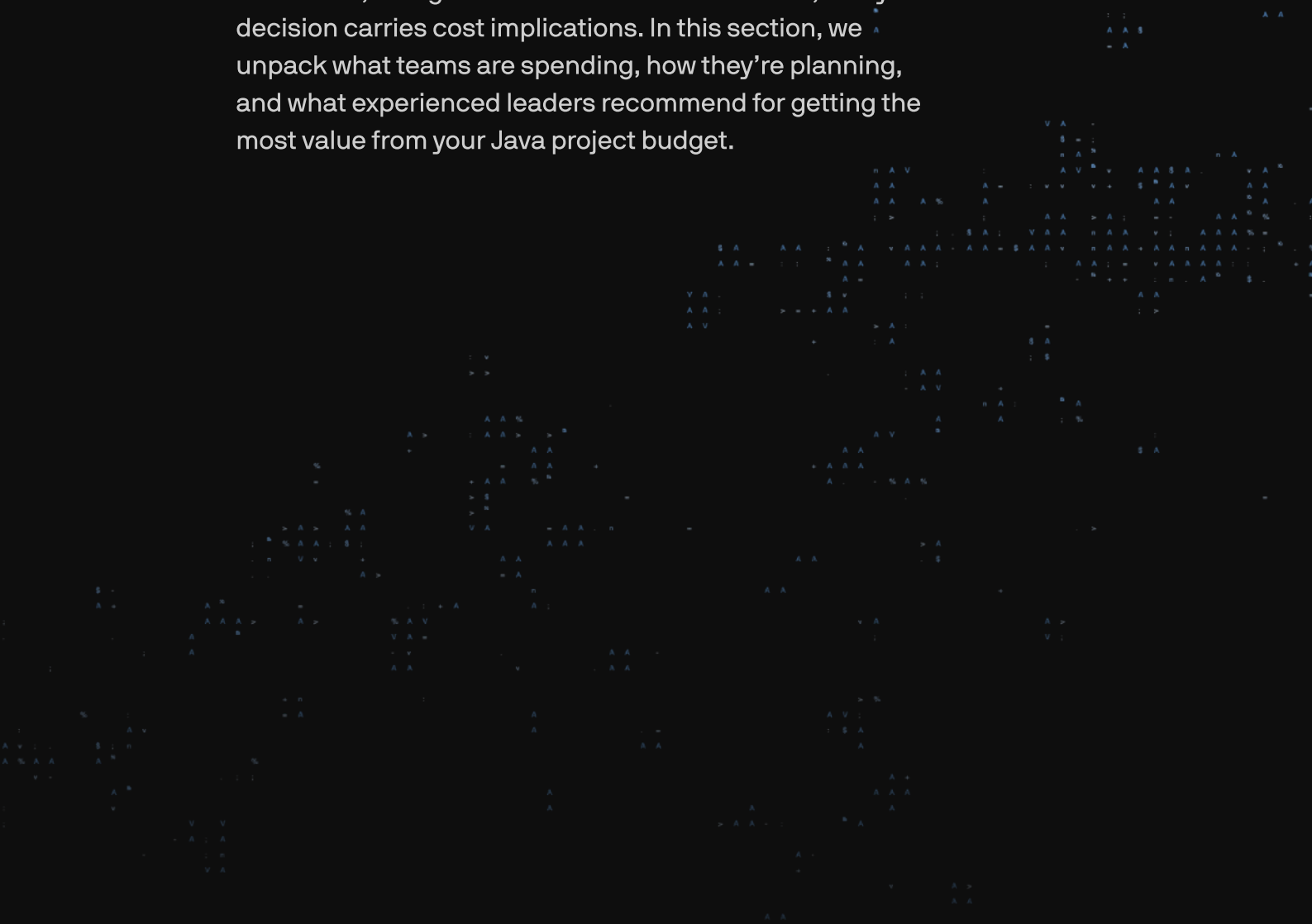
Before kicking off your next Java project, think carefully about your decision-making structure and whether it supports your team and project goals.

→ Are your leads and architects close enough to the tech to make sound decisions?
→ How will you involve developers in choosing frameworks, tools, and libraries?
→ If you rely on a company-approved technology list, how do you balance the desire to standardize technology against productivity? How long does it take a team to get a technology approved through your process? Do you have a list, just to have a list, or do you revisit your list, and the tech stacks of your applications, and use this review to influence your modernization projects?
→ Are your choices scalable and maintainable for the next 5–10 years?
→ How will you evaluate trade-offs between team familiarity and architectural goals?

Technology selection is not just a technical decision—it's a cultural one. Involving your team while keeping alignment with broader goals helps create a stack that your developers want to work with and your business can grow on.

Forecasting & budgeting for Java projects in 2025

As Java teams plan for 2025, budgeting plays a central role in shaping what's possible. From team structure to tech stack, hiring timelines to cloud architecture, every decision carries cost implications. In this section, we unpack what teams are spending, how they're planning, and what experienced leaders recommend for getting the most value from your Java project budget.



The budget landscape: How much are teams working with?

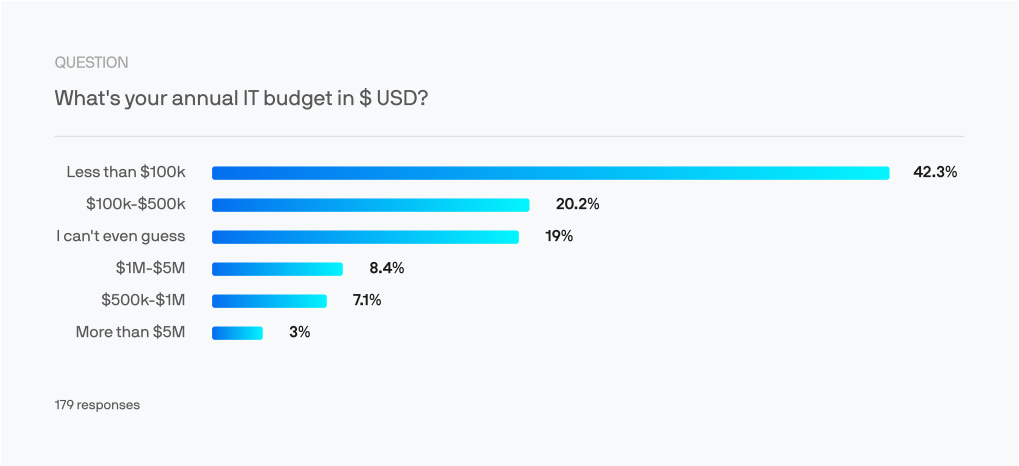
Most Java teams in our survey are working with modest budgets:

→ 42.3% operate with less than $100k/year
→ 20.2% fall into the $100k–$500k range
→ Only 18.5% report $500k+ budgets
→ 19% of respondents aren't sure of their budget

While the reported budgets may seem low—especially given the cost of a single full-time developer—there are several likely explanations:

→ Some respondents may have shared **tooling or project-specific budgets** rather than full IT spend
→ Hobby or maintenance-mode projects with little active investment could skew the numbers
→ Others may simply lack visibility into budget decisions

This reflects the wide range of Java project types today—from side projects and startups to enterprise-scale systems.

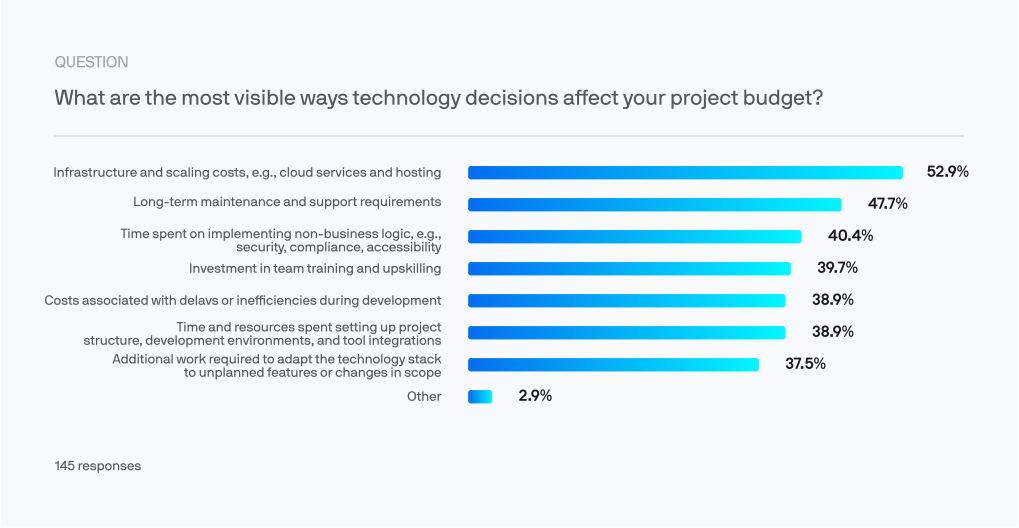


How does your budget compare? Does your plan account for both immediate and long-term costs?

What drives Java project budgets

So what actually eats into your budget? Respondents flagged several recurring issues:

- → **Cloud infrastructure and scaling costs:** 53%
- → **Maintenance and support over time:** 48%
- → **Time spent on non-business logic** – security, compliance, accessibility: 40%
- → **Training and upskilling the team:** 40%
- → **Delays and inefficiencies during development:** 39%
- → **Initial setup and integration work:** 39%
- → **Changing requirements and scope creep:** 38%

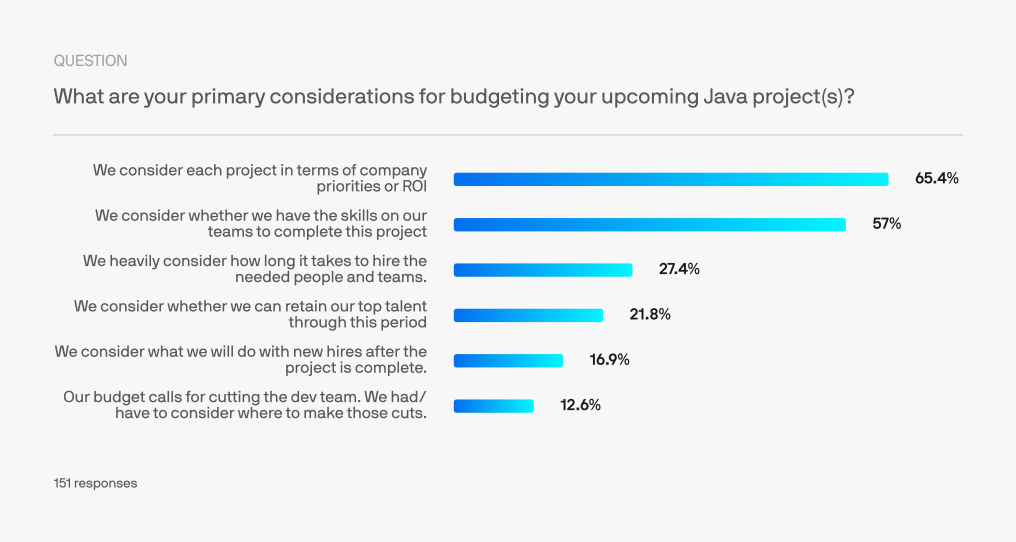


The lesson? It's not just big tools and cloud bills—complexity, misalignment, and rework cost just as much.
Are you planning proactively for these predictable (yet often overlooked) costs?

How teams prioritize during budget planning

When budgeting for Java projects, teams weigh more than just cost—they evaluate readiness, resourcing, and long-term alignment. Here's what respondents said they consider most:

→ 65.4% consider company priorities or ROI first and foremost.
→ 57% assess whether they already have the skills in-house to complete the project.
→ 27.4% factor in the time and difficulty of hiring new team members.
→ 21.8% are concerned about retaining top talent during the project.
→ 16.9% consider what to do with new hires once the project ends.
→ 12.6% are facing budget-driven dev team cuts and must decide where to scale down.

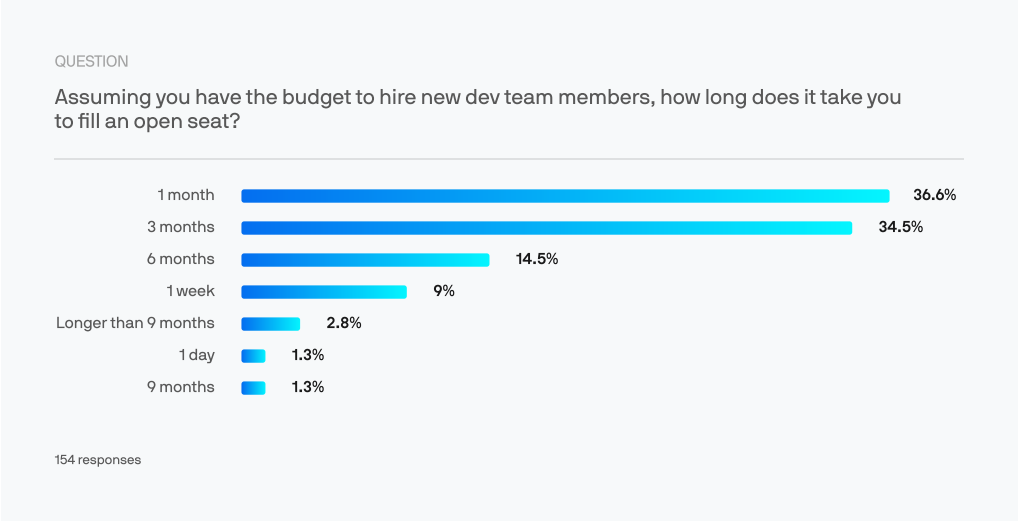


These responses highlight that budget planning is just as much about people as it is about tech—capabilities, timelines, and long-term team health all shape the final numbers.
Are you budgeting for a sustainable team—not just project delivery?

Hiring, team structure, and timeline pressures

Hiring is another major consideration for Java project budgets. We asked teams how long it takes to fill an open dev seat when budget is available:

- → 36.6% said 1 month
- → 34.5% said 3 months
- → 14.5% said 6 months
- → Only 10.3% filled seats in a week or less
- → A small group (4.1%) reported 9+ months



Hiring is rarely instant. This means your budget must cover not just salaries—but time to hire and onboard. Are you accounting for ramp-up time?

In 2025, here's how teams are thinking about growth:

- → 42.1% are focused on using AI and automation to improve efficiency

"Implement AI-powered tools for predictive forecasting, anomaly detection, and real-time budget monitoring."

- **Munezero Sage Alliance,**
  Full-stack developer, Technology/IT Services/Consulting

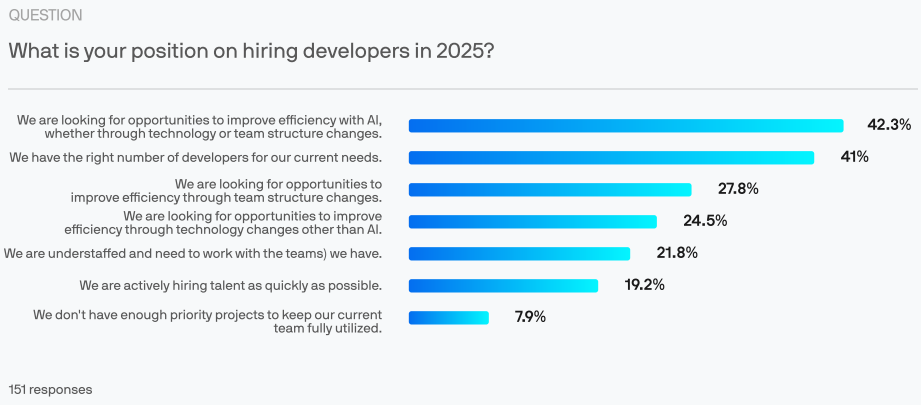
"Learn and plan with AI before budgeting a new project. I think we will need less man power for basic coding and AI for building reports, for instance."

- **Marcelo Castro,**
  Full-stack developer @ TDec Redes de Computadores Ltda

"If you want to fix poor software, fix poor teams. Organize around features—not functions. Have devs, ops, UX all focused on the same domain. Smaller, cross-functional teams—like the two-pizza team model—enable faster iteration and cleaner software boundaries."

Josh Long,
Spring Developer Advocate

- → 40.8% say their team is already right-sized
- → 27.8% are restructuring teams to increase output
- → Only 19.7% are actively hiring quickly
- → Just 7.4% say they don't have enough work for their current team



Most Java teams are trying to do more with what they have—hiring when necessary, but prioritizing efficiency.

Strategic budgeting: What leaders recommend

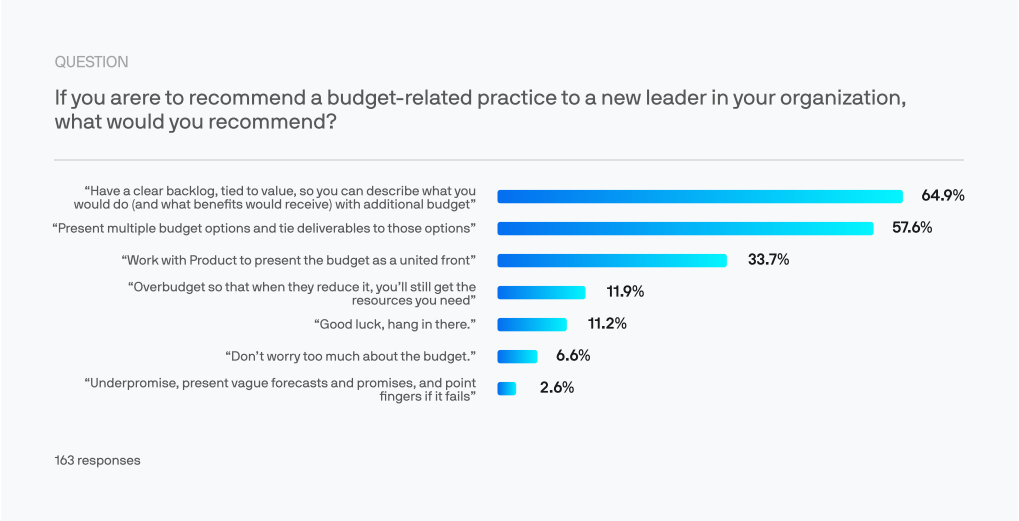
When asked what budgeting practices they'd recommend to newer leaders, the top advice was all about clarity and alignment:

→ 65% said: Tie your backlog to business value. Be ready to explain what additional budget enables—and why it matters.
→ 58% said: Present options. Show what happens at different budget levels with clear deliverables attached.

"Always provide options because a business usually does not understand the full picture from IT perspective, all opportunities and possible outcomes. You need to explain this with numbers and values to gain or lose."

- **Maris Birgers,**
  VP, C-level @ Longo

→ 34% recommended working closely with Product to advocate for shared goals.
→ 11.9% recommended a "contingency buffer" - a planning technique that might imply top-down budget controls, where upper management squeezes their teams. Interestingly, if a respondent selected the number 1 answer, "to have a clear backlog..." they had a 0.3% higher probability to also suggest overbudgeting.



Which of these practices would help you advocate for your project more effectively?

When and how the best budgeting happens

We asked this as an open-ended full text question. We understood that many organizations see internal processes as proprietary, so we intentionally did NOT ask about the budget process of respondents current employer, but instead, about the best budgeting process that they encountered during their career so far. This led to a variety of responses.

It was clear that there were two main differentiators in responses: software development service providers who build projects on contract, and in-house software development teams.

That said, we believe that similar considerations are at play:

→ **Project-based budgeting:** Budgets are often allocated on a per-project basis, considering specific project needs and goals. One difference here, is that in-house teams tend to assign extra value to long-term strategic projects, where service providers tend to think in terms of winning a pitch for a project.

→ If you are an in-house technical team, it pays to understand the long term strategic value of the project you're being asked to build - before deciding on technologies to use or team sizes.

The best budgetting process was one where the organization was split in two: dedicated product teams, and project teams. The latter had to purchase the product from the first as part of the project. Each could focus on their totally different budgets. -

- Tom Eugelink,
  Full-stack developer @ Softworks

→ **Early planning:** Budgeting process starts months in advance to allow ample time for planning and discussion.

"1) Begin almost 6 months before budget tabling 2) Clear business values attached to projects 3) Early discussions between business and IT 4) RFI for complex projects."

- VP, C-level,
  Finance/Banking/Insurance

"The best budgeting process I experienced was structured yet dynamic, allowing for flexibility and collaboration. It kicked off mid-year with strategy discussions where finance set initial guidelines based on company goals and market trends. Departments then submitted draft budgets, justifying key expenses and growth plans. From there, an iterative review process took place, with finance and leadership refining projections through open discussions rather than rigid approvals. Adjustments were made based on evolving priorities, ensuring alignment without unnecessary constraints. By year-end, the budget was finalized with a clear understanding of trade-offs and opportunities, making the process both strategic and adaptable."

- VP, C-level,
  Finance/Banking/Insurance

"Basically the budget process started with the planning phase where we come up with budget proposals based on the current activities, upcoming replacements and upgrades, and future innovations. These inputs are made in consultation with the relevant departments. The budget process starts in the 3rd quarter of the financial year. To get the budget approved, it must be within the budget limit for ICT, e.g. 1 % of projected revenue. If the budget goes above the limit then different projects need to be prioritised for spending and the lowly ranked one removed till budget is within limit."

- Victor Tichayana Hokonya,
  VP, C-level @ okzim.co.zw

"The budgeting process started early, allowing ample time for planning and discussion. We developed multiple budget proposals, which were reviewed collaboratively with the entire management board and the customer. Through multiple iterations and meetings, we refined the plans until we agreed on a direction. However, the budget remained flexible throughout the project, with small adjustments made as needed to accommodate unforeseen changes. In reality, a fully finalized budget only emerged toward the end of the project, reflecting the dynamic nature of the work."

- Full-stack developer,
  Unspecified industry

→ **Flexible budgeting:** Allows for adjustments and changes throughout the project lifecycle to accommodate unforeseen needs.

"I'd make it even more dynamic—less rigid annual planning, more quarterly reviews to adapt to changing needs. Also, faster approval cycles for smaller, high-impact investments would keep innovation moving without unnecessary delays."

- **Shaukat Mahmood Ahmad,**
  VP, C-level @ GuCherry Blog by Everestthemes

"In our company, budgets are not allocated on a project-by-project basis. Instead, our primary focus is ensuring that company revenue exceeds expenses. This approach helps us maintain a constant focus on profitability and efficiency. We have a fixed team whose members take on different roles based on the needs of various projects. This flexibility allows us to optimize resource management and leverage the expertise of team members across multiple initiatives. Time management in our company is based on allocating team members to projects. This method enables us to quickly assign resources according to project priorities and needs, minimizing wasted time and effort. Additionally, when a new product is added to our software portfolio, we give it special attention. This means allocating the necessary resources and focusing entirely on its development and success. This approach allows us to achieve sustainable growth and profitability while maintaining flexibility."

- **Team lead,**
  Technology/IT Services/Consulting

→ **Strategic alignment:** Budgets are aligned with company goals and strategic objectives to ensure relevance and effectiveness.

"The budget was dynamically allocated and recalculated based on the projected ROI backed by multiple business cases."

- **Chris Erlich,**
  VP, C-level @ yellowhippy.com

"At my previous company, the budget process was collaborative, transparent, and aligned with strategic goals. It started in August/September for the next fiscal year, finalized by mid-December. Leadership shared financial targets, departments drafted budgets, and finance reviewed them with feedback. Quarterly check-ins allowed flexibility. Transparency, early planning, and clear communication made it effective. To get approval, I aligned requests with company goals, explained the "why," consulted stakeholders early, stayed open to compromise, and used data to support my case. It was structured yet adaptable—my best budgeting experience!"

- **Samuel Fuentes,**
  Full-stack developer, Software/SaaS/Cloud Solutions

""Efficient" budgets can present as a thin bench, but they're much better than a boom/bust hiring/layoff cycle. The best organizations I've worked at have prioritized developing skills and processes over increasing headcount. We only opened reqs when there was a clear, long-term business requirement, but we retained a subset for, say, contractors and consultants to step in where necessary. Developing consensus around when to hire is necessarily a time-consuming process entailing a comprehensive view of the roadmap, the economic conditions, etc. We would plan out infrastructure and headcount spending separately and align them with the business goals."

- **Josiah Haswell,**
  Architect @ Sunshower.io

→ **Transparency:** Clear communication and visibility into the budgeting process to build trust and understanding.

"Best experiences about budget come when the CXO team has a clear vision of exactly what they want and clear vision about ROI. Also they have faith in the team (backed by proven track record within the organization) that the plans will get executed."

- VP, C-level,
  Technology/IT Services/Consulting

What does your current budgeting process look like—and where could it improve?

Improving even the best budgeting processes

We asked a question about the best budgeting processes, and then followed up by asking what could be improved in those processes. Overall, even seasoned leaders see room for improvement:

→ **Negotiate per-project flexibility:** Adjust budgets for each project and negotiate flexibility margins with clients.

"To improve the budgeting process, I'd suggest setting a predefined flexibility margin to allow minor adjustments without formal approvals, reducing delays and admin work."

- Full-stack developer,
  Custom Software Development/IT Consulting

"Be honest with customers—share your best estimates but clarify that projects rarely stay within exact boundaries. New features and unexpected issues always come up."

- Architect
  @ IZUM

→ **Educate stakeholders:** Educate management and sales teams on realistic estimates and the impact of under-budgeting.

"Keep the company's needs and bottom line in mind. If possible, involve other departments to combine budgets and achieve shared goals."

- **Jay Whaley,**
  Full-stack developer @ Dunbarton

"Never lose sight of the projected outcome and always be prepared to demonstrate the value of your current state. Make sure that on all decision-making levels there are people who value the project and see the benefit."

- **Chris Erlich,**
  VP, C-level @ yellowhippy.com

→ **Adopt phased or agile budgeting** that aligns with sprint cycles

"Instead of a fixed budget, consider using an agile budgeting approach that allows for flexibility. This can involve allocating budgets in phases or sprints, enabling adjustments based on project progress and changing requirements."

- **Tariq Khan,**
  Full-stack developer @ Elite Software Tech

→ **Technology and innovation investment:** Focus on investing in new technologies, skilled developers, and user-friendly budgeting software.

"Selecting the most effective and budget-friendly tools is essential for achieving optimal results. It's important to carefully evaluate the features, quality, and pricing of each option to ensure they meet your specific needs without exceeding your budget. By making informed choices, you can maximize productivity while minimizing costs."

- Barış Şahin,
  Full-stack developer @ License4J

"To improve even the best budgeting process, I would suggest more real-time adaptability and collaborative forecasting. Instead of a fixed annual cycle, a rolling budget approach with quarterly reviews could help adjust for market shifts and new opportunities. Additionally, integrating real-time financial tracking tools would allow departments to see budget variances early and make informed adjustments. More cross-functional collaboration between finance and business units would also ensure budgets are not just numbers but strategic enablers. Lastly, adding a feedback loop post-budget approval—where teams discuss what worked and what didn't—would refine the process year after year."

- VP,
  C-level, Finance/Banking/Insurance

→ **Contingency planning:** Always include a contingency buffer to handle unforeseen expenses.

Take risks into account with sufficient emphasis."

- Architect,
  Custom Software Development/IT Consulting

Samuel Fuentes, a Full-stack developer in the Software/SaaS/Cloud Solutions sector, shared a thoughtful, step-by-step suggestion to help refine your budgeting process:

"Even the best budgeting process I've experienced could be improved with a few tweaks:

1. More Real-Time Data Access: Provide teams with real-time financial data and tools to track spending against budgets throughout the year, not just during quarterly reviews. This would help identify issues earlier and allow for quicker adjustments.

2. Simplify the Tech: While we had good systems in place, the tools were sometimes clunky or hard to navigate. Investing in more user-friendly budgeting software could save time and reduce errors.

3. Increase Cross-Department Collaboration: While there was collaboration within departments, more cross-departmental discussions could help align priorities and avoid silos. For example, regular workshops or shared platforms where teams can see each other's goals and challenges might lead to better resource allocation.

4. Shorter Feedback Loops: The review process involved multiple rounds of feedback, which sometimes slowed things down. Streamlining this with clearer initial guidelines or faster decision-making could make the process more efficient.

5. Post-Year Review: After the fiscal year ended, a formal review session to evaluate how well the budget worked (or didn't) would be helpful. This could include lessons learned and specific recommendations for the next cycle, ensuring continuous improvement. These changes would make the process even more dynamic, transparent, and efficient while keeping everyone aligned on shared goals."

- **Samuel Fuentes,**
  Full-stack developer in the Software/SaaS/Cloud Solutions sector

One piece of advice regarding budgeting that we didn't see in the open-ended questions was surprising. We didn't see a single person say "build a team of advisors" or "speak with people in other organizations about their budgeting process". From years of experience, we've found that building relationships with people at different levels of our various organizations as well as having friends in different organizations is incredibly helpful.

Budgeting is one of those areas where every organization is trying to do their best. It would be shocking if your organization were unwilling to learn from the experience of others. So don't be afraid to reach out and connect with other people in your roles, in other places. There's a good chance you'll both benefit from the experience!

Budgeting advice from the Java community

When asked what advice they'd give to others budgeting for upcoming Java projects, respondents shared these key themes:

→ **Budget realistically for developer costs:** Attracting and retaining good talent pays off.

"Factor in realistic developer staff costs to attract good developers."

- Architect,
  Software/SaaS/Cloud Solutions

"Budget for critical skills retention, as skills can kill a project ."

- Victor Tichayana Hokonya,
  VP, C-level @ okzim.co.zw

→ **Choose tools and frameworks carefully:** Prioritize productivity and long-term value.

"Establish a detailed budget that accounts for every element involved, clearly outlining the costs associated with each tool and resource. It's crucial to remain committed to the initial selections made; changing tools or software mid-project to save costs can lead to inefficiencies and complications. By sticking to the original plan and investment choices, you will help ensure a smoother workflow and mitigate the risk of unintended expenses arising from transition issues."

- Barış Şahin,
  Full-stack developer @ License4J

"Wise choice of technologies and tools. Always plan a budget reserve (20-30%) because actual costs are often higher than expected."

- Michał Szczygieł,
  Engineering manager @ Firma Kams

→ **Don’t skip testing or maintenance:** Future you will thank you.

“Always keep in mind the future of your project: scalability, maintainability, vulnerabilities, modernization.”

- Team lead,
  Technology/IT Services/Consulting

“Calculate testing with about 40% and deployment with about 20% of the development costs.“

- Team lead,
  Custom Software Development/IT Consulting

“When budgeting for an upcoming Java project, consider the following key factors: - Java applications often require long-term support, so allocate resources for future updates, refactoring, and performance optimization. - If deploying to the cloud, factor in hosting, database, and scaling costs. Unexpected infrastructure expenses can quickly add up. -Skimping on testing or security can lead to higher costs down the line. Ensure your budget includes robust testing frameworks, code reviews, and security audits.”

- Full-stack developer,
  Custom Software Development/IT Consulting

“Plan for the long term—don’t just budget for development, but also for maintenance, scaling, and security. Leave room for unexpected changes, prioritize must-haves over nice-to-haves, and invest in automation to save costs down the road.”

- Shaukat Mahmood Ahmad,
  VP, C-level @ GuCherry Blog by Everestthemes

→ **Keep the team motivated:** Budget for success, not just survival.

"Try to be realistic about the number of developers based on the technical effort required by the project."

- Team lead,
  Custom Software Development/IT Consulting

"Define objectives clearly, provide incentives to motivate teams and, above all, celebrate the success of the project with appropriate recognition."

- Full-stack developer,
  Technology/IT Services/Consulting

→ **Build in contingency buffers:** Surprises happen.

"Budget more than you think for known tech debt—and then add extra for the unknown tech debt."

- Alexander Wollan,
  Full-stack developer @ Kosmos Solutions

"Allow for some overbudget. Bugs will occur, refactorings will have to be made."

- Architect.
  Finance/Banking/Insurance

→ **Optimize for performance and scalability:** Early planning prevents expensive rework.

"Learn and plan with AI before budgeting a new project. I think we will need less man power for basic coding and AI for building reports, for instance."

- Marcelo Castro,
  Full-stack developer @ TDec Redes de Computadores Ltda

→ **Define project scope clearly:** A focused project is easier to budget.

"When budgeting for a Java project, focus on clear scope, efficient resource allocation, and future scalability. Plan for developer costs, infrastructure (cloud, CI/CD, databases), security, and compliance. Factor in testing, maintenance, and a 10-20% contingency fund for unexpected changes. Leverage new technologies like AI for automated testing, performance optimization, and cost analysis to streamline development. Regularly review and adjust the budget to stay agile and optimize spending while ensuring long-term growth."

- **Samuel Fuentes,**
  Full-stack developer, Software/SaaS/Cloud Solutions

"Determine your goals and can you monetize project during development. Then determine dependencies of other projects. Good goals will save you lot of time when you are full speed as with clear goals in mind you can pick correct tools, determine how project should be split between teams, how devops will work, what platforms you will use. But don't be afaird to create proof of concepts (pocs) and test to determine correct tools, technologies and solutions early on. More complex your project is more you will save when poc and test early on different solutions. If you find late in project that something will not work, fixing it will cost much more compared to early pocs and tests."

- **Full-stack developer,**
  Logistics/Transportation

→ **Focus on solving real customer problems:** Let outcomes guide investment.

"Budgeting for upcoming Java projects requires careful planning and consideration of various factors to ensure that resources are allocated effectively."

- **Tariq Khan,**
  Full-stack developer @ Elite Software Tech

→ **Leverage open source when possible:** Solutions like **Vaadin** can cut costs and accelerate delivery. While open source frameworks offer great value, it's important to remember that not all are entirely free. For example, **Vaadin provides a free, open-source core, along with optional commercial tools and support** for teams that need more. Planning for this upfront helps avoid surprises during budgeting.

"Implement AI-powered tools for predictive forecasting, anomaly detection, and real-time budget monitoring."

- **Munezero Sage Alliance,**
  Full-stack developer, Technology/IT Services/Consulting

Which of these ideas resonates most with you?
What will you take into your next planning session?

Considerations: Budgeting for a new Java project

→ Are your budget estimates tied to actual business outcomes?
→ Have you considered the cost of complexity—setup, compliance, training, and long-term maintenance?
→ Can your team realistically deliver with your current headcount and skillset?
→ How long will it take you to hire or ramp up if needed?
→ Are you giving stakeholders clear options and trade-offs when asking for budget?
→ Is your budget flexible enough to handle scope changes or tech stack challenges mid-project?
→ What investments can you make now (in tooling, automation, architecture) to reduce costs later?

Budgeting isn't just a financial process—it's a strategy for success. Are you setting yourself up to deliver value, adapt quickly, and grow sustainably?

Choosing technologies for new Java projects

Technology and framework decisions are foundational to every Java project. While many reports list tools by popularity, we hope to focus on the reasoning behind technology choices. We understand that choices are subjective to project-based use cases, so we intend to highlight the decision-making process and considerations for making technology decisions: so that when you are making your decisions, you know that you're covering your bases. Some considerations include the priorities driving those decisions, and the balance between developer preferences, leadership oversight, and long-term sustainability.

Top priorities when choosing new technologies for new projects

Firstly, with every new project, you want to list your top priorities. We asked respondents to rank each possible answer from Unimportant to Most Important. We left it up to you to decide the difference between each ranking, such as the difference between Very Important and Most Important.

We asked "You're considering using a new technology on your upcoming project. How important are each of these factors when making your decision?" When selecting technologies for new projects, Java teams consistently prioritize security, stability, and long-term support. Security emerged as the top concern, with 40.2% of respondents ranking it as the 'Most Important' factor when evaluating new technologies.

Many of our respondents used open source technology in their applications. 89% of respondents said that having "Long-term maintenance available" was either Most Important or Very Important.

> "When choosing a framework, prioritize project needs—scalability, communication, support, compatibility, and security. Pick tools that boost developer productivity and integrate cleanly with your stack, and consider AI-powered features for optimization and automation."
>
> 
> Josh Long,
> Spring Developer Advocate

"In our experience, when considering a framework we must mainly consider the maturity, stability and future support of the tool. There are very good tools that stop being maintained and that puts a project in serious trouble."

- Marco Castillo,
  Full-stack developer @ nube4.com

"Full stack. Ease of use. Maturity. Resources. LTS."

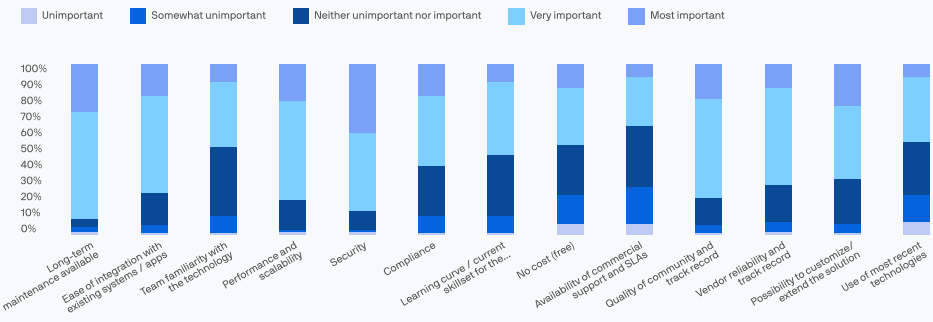
- **Architect,**
  Government/Public Sector

"Long-term maintenance from a trustworthy vendor, no/few bugs and problems, provides the features we need, performance, compliance, security, great developer experience, open source with a license that we can use."

- **Architect,**
  Custom Software Development/IT Consulting

QUESTION

You're considering using a new technology on your upcoming project. How important are each of these factors when making your decision? (5 point scale from not important to very important)



255 responses

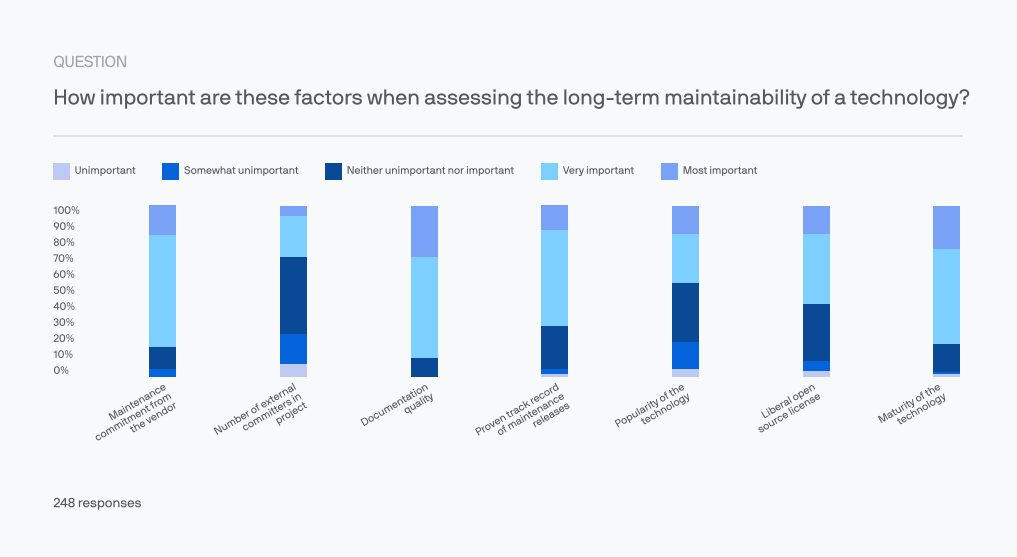
What matters most for long-term maintainability?

When starting a new Java project, your tech choices shouldn't just be about what's hot—they should be about what will still hold up five years from now. We asked developers which factors they consider most important when assessing the long-term maintainability of a technology.

The top response by far? Documentation quality—with 88% rating it as very or most important. Good docs mean faster onboarding, fewer bottlenecks, and easier handovers down the line.

Next up: maintenance commitment from the vendor (82%) and maturity of the technology (79%). Developers clearly favor stability, predictability, and tools with a solid track record over flashy new frameworks.

Surprisingly, popularity and community size ranked much lower. The takeaway? When it comes to long-term success, teams value clarity, stability, and support over hype. Cchoose tech that's well-documented, actively maintained, and built to last.



Popular frameworks and version trends in Java projects

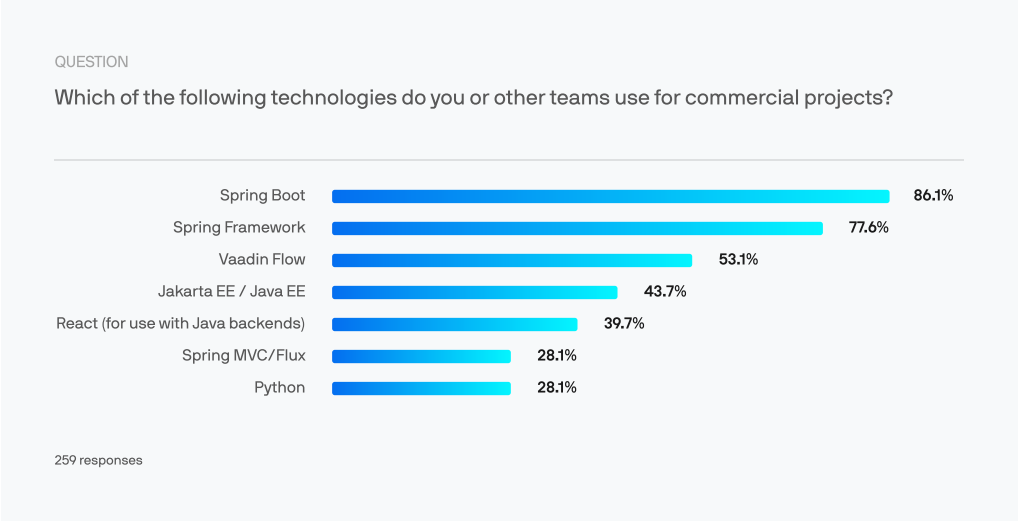
Choosing the right frameworks and runtime version is one of your earliest, most impactful decisions in a new project. Our survey data reveals where most teams are leaning—and what's influencing those choices.

Spring continues to dominate Java projects

In 2025, Spring Boot (86.1%) and Spring Framework (77.6%) remain the backbone of enterprise Java development. Their continued popularity highlights Spring's reliability, strong community, and wide adoption across industries.

If you're building a new Java application, it's likely you're already considering Spring. But are you also thinking about how tightly you want to couple your business logic to framework conventions, or how to balance ease of use with long-term flexibility?

On the front-end side, React (39.7%) and Angular (26.3%) are the most commonly used JavaScript frameworks in combination with Java backends.



Note: Because this survey respondents included Vaadin community, the tech stack usage is more weighted to Vaadin Flow users than it might otherwise be.

Criteria for choosing front-end technologies

When selecting a frontend technology, respondents prioritized:

→ Developer productivity – 50.6%

"When choosing a framework, prioritize project needs, scalability, community support, compatibility, and security. Opt for frameworks that boost developer productivity and integrate well with your tech stack. Consider AI-powered tools for automation and optimization. A well-supported, future-proof framework ensures efficiency and long-term success."

- VP,
  C-level, Finance/Banking/Insurance

→ Availability of UI components – 45.7%
→ Team familiarity – 30.7%
→ Full-stack security – 29.8%

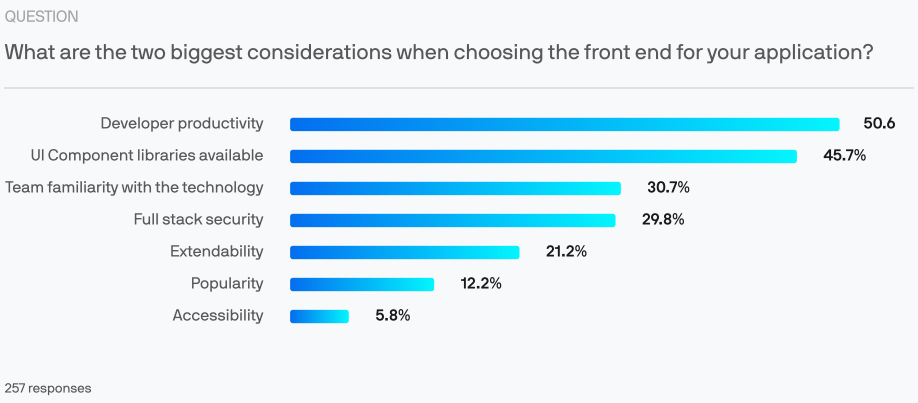
Vaadin is a comprehensive, fully integrated, end-to-end framework combining the backend and front end into a single coherent solution. It helps you build applications with proven, secure architecture and long-term maintainability.
This is one of the key reasons companies choose to build mission-critical applications with Vaadin.

"Stability, security and productivity"

- **Full-stack developer, Healthcare/Pharmaceuticals/Biotechnology**

"Compatibility, ease of use/integration, documentation and online help, open source, the framework is continually maintained and not a dorment project, feature rich, great performance."

- **Full-stack developer, Technology/IT Services/Consulting**



"When selecting technology and frameworks for a project, understanding the learning curve is crucial. The learning curve refers to the amount of time and effort required for team members to become proficient in using a particular technology or framework. It encompasses various factors, including the complexity of the technology, the availability of resources and documentation, and the prior experience of the team. A steep learning curve can lead to increased delays during the initial phases of development, potentially impacting project timelines and costs. Therefore, it is essential to assess how quickly team members can adapt and start delivering value with the chosen technology, ensuring that it aligns with their skills and the project's requirements. Properly evaluating the learning curve can ultimately lead to more efficient development processes and better overall outcomes."

- **Barış Şahin,**
  Full-stack developer @ License4J

"The choice of a framework depends on its compatibility with the project's architecture, performance, and ecosystem. Maintenance ease, documentation, and community support are also crucial."

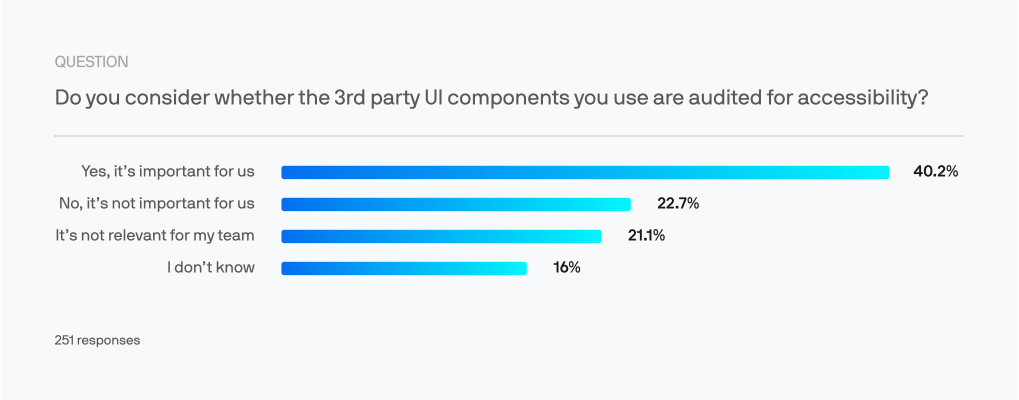
- **Damlabin Nkamneu Arsene Vivien,**
  Full-stack developer @ Université Laval

Accessibility: An opportunity and a gap

Does accessibility matter when building your application? Is it important for your audience? Or based on the location of your users?

When asked about accessibility in third-party UI components:

→ 40.2% of respondents said accessibility audits were important
→ 43.8% said accessibility was either not relevant or not a priority
→ 15% were unsure



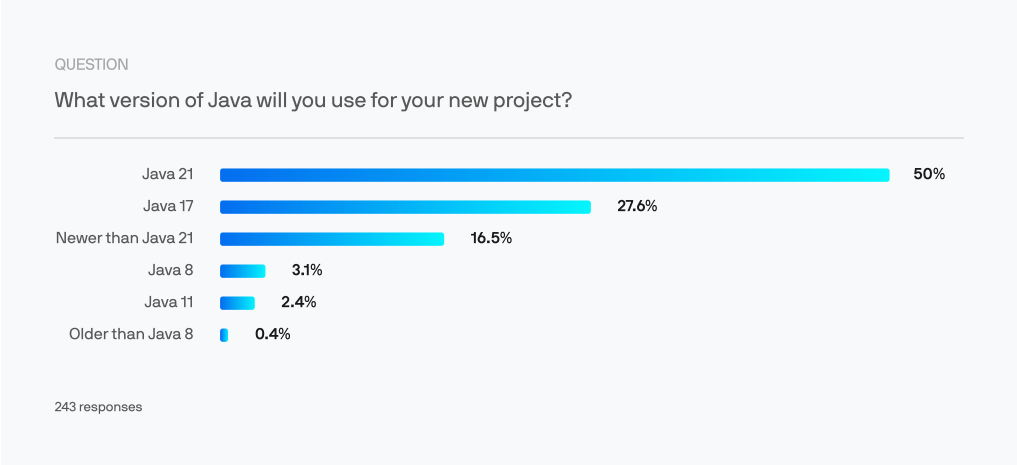
This suggests a knowledge or planning gap around accessibility—and a potential risk area for public-facing or regulated applications, especially for organizations operating in Europe as the European Accessibility Act will come to effect in June 2025.

If your team prefers staying within the Java ecosystem without taking on a separate JavaScript stack, Vaadin Flow enables you to build modern UIs in pure Java.

Java 21 adoption & upgrade challenges

Choosing a Java version might seem like a technical detail, but it has deep implications for security, performance, maintainability, and future upgrade paths. For teams planning a new Java project, this decision is a strategic one.

Java 21 is the clear favorite for upcoming projects. Half of all respondents have selected Java 21 as their preferred version. Java 17, the previous long-term support (LTS) release, still sees significant usage at 27.6%, while 16.5% of teams plan to adopt a version newer than Java 21—indicating a growing interest in staying on the cutting edge of the Java platform.



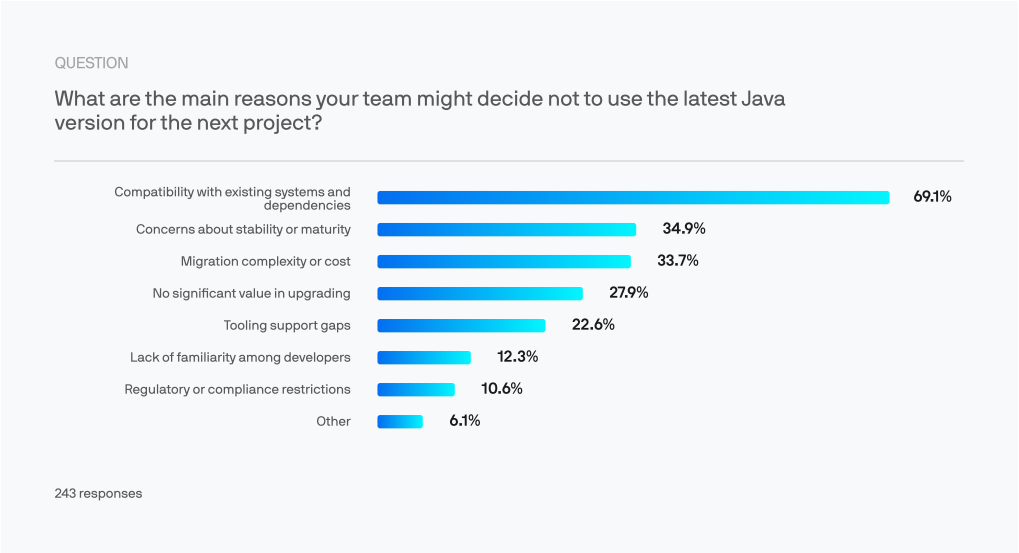
These numbers point to a broader trend: Java teams are increasingly moving toward modern, supported releases that offer enhanced security, developer productivity, and future compatibility.

What's holding some teams back?

Despite strong interest in newer Java versions, many teams still face real-world constraints. The biggest barrier is compatibility with existing systems and dependencies, cited by 69.1% of respondents.

Other notable challenges include:

- → Concerns about stability or maturity – 34.9%
- → Migration complexity or cost – 33.7%
- → A perceived lack of significant value in upgrading – 27.9%



These insights show that for many teams, upgrading isn't just about flipping a version number—it requires planning, resources, and justification.

Considerations: Choosing the right technologies for your Java project in 2025

Starting a new Java project today means making decisions that will shape your architecture, team productivity, and ability to scale for years to come. While popular frameworks and tools can be a helpful signal, your choices should be driven by the real needs of your project and your team.

Before committing, ask yourself:

→ Does this technology align with the team structure and skills you're building?
→ Can you rely on long-term support, active maintenance, and security updates?
→ How easily can it integrate with your infrastructure and existing systems?
→ Will it meet your compliance and accessibility requirements, especially with the EU Accessibility Act coming into effect?
→ Is the frontend framework tightly integrated with your backend—or are you managing separate stacks?
→ Does your team have the capacity to manage a frontend/backend split, or would full-stack Java simplify development?
→ Are you starting greenfield, or inheriting legacy constraints that limit your options (e.g., Java version compatibility)?

→ Are you planning for upgrades? Does your stack choice make it easier—or harder—to stay current?
→ How strong is the documentation, community support, and onboarding experience?
→ Is the framework too heavy—or not opinionated enough—for the size and scope of your application?
→ Are you building for today's requirements, or planning for long-term maintainability, performance, and evolution?

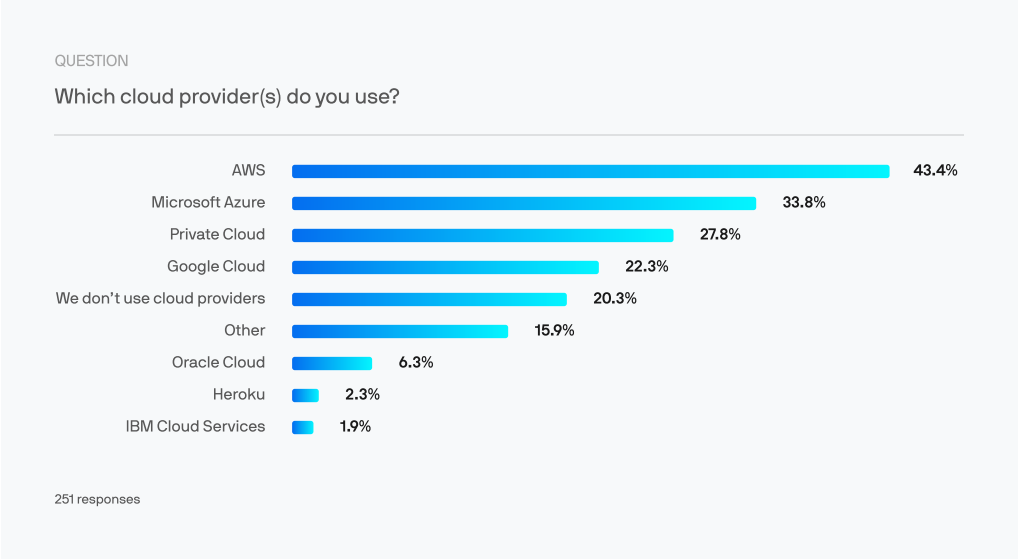
The most successful Java projects in 2025 will be built not just with modern tools—but with thoughtful, future-focused decisions.

Infrastructure & cloud adoption

Infrastructure is one of the first big decisions you'll face when starting a new Java project. Most teams today are leaning into cloud, containerization, and automation—but not everyone is moving at the same pace. Some still stick to more traditional setups, often for reasons like simplicity, security, or internal expertise. There's no one-size-fits-all answer here.

Cloud provider choices

When it comes to choosing a cloud provider, AWS and Azure are still leading the pack—used by 43.4% and 33.8% of teams respectively. Private cloud options (27.8%) are still going strong, especially for teams with strict compliance needs or more control over their environments. Google Cloud is used by 22.3% of teams, and interestingly, **about 1 in 5 Java teams aren't using cloud at all yet.**



So how are teams making their decisions? We asked respondents "What advice would you give to others when it comes to choosing a cloud provider?" and to provide text-based answers, after analysing the answers through AI, responses fell into the following categories.

From what we heard, the top factors include:

→ **Scalability and performance** topped the list. Many teams emphasized the importance of choosing a provider that can scale with your application and offer performance optimization tools.

"When choosing a cloud provider, focus on cost, scalability, security, and integration with your tech stack. Compare pricing models (pay-as-you-go vs. reserved instances) and performance (latency, reliability). Ensure strong security, compliance, and disaster recovery options. Look for AI, automation, and analytics tools to optimize resources. Finally, consider multi-cloud flexibility to avoid vendor lock-in and ensure resilience."

- VP,
  C-level @ Finance/Banking/Insurance

"The largest cloud service providers are often regarded as the most reliable and effective options in the market. These companies typically offer a wide array of services and features, catering to various needs ranging from computing power to storage solutions. They often have competitive pricing models, allowing businesses of all sizes to take advantage of advanced technology without breaking the bank. Furthermore, these major providers invest heavily in infrastructure, ensuring high availability, robust security measures, and constant innovation, which can greatly benefit users seeking scalability and flexibility for their operations."

- Barış Şahin,
  Full-stack developer @ License4J

→ **Cost considerations** came in close behind. Teams advised looking beyond the headline pricing—think long-term costs, scaling charges, and hidden fees.

"Pay careful attention to price of your cloud provider. Nearly all of them support the ability of scaling the database, and scaling the core functionality to respond by firing up additional processors, and doing so with Open source and free and fully tested components."

- Full-stack developer,
  Custom Software Development/IT Consulting

"Look up if it fit your demands. Consider the costs and be aware that in case of problems these costs can raise up especially when using provider's support"

- Full-stack developer,
  Industry not specified

"Decide what services are really required for the job and do not over-architect the solution. Have observability in mind, both operational and cost-wise. Make an up-front cost estimation. Then think again and repeat the process."

- Backend developer,
  Industry not specified

"We have tried various combinations over the years and the cost seems to average out when you factor in downtime and problems that come from the cheaper options."

- Terry Packer,
  VP, C-level @ Radix IoT

"Cost plays a major role, especially when deciding between major providers like Google Cloud, Microsoft Azure, and AWS. Evaluate not just initial pricing but also long-term costs, including scaling, storage, and data transfer fees. While private cloud solutions offer more control, flexibility, and potentially better performance, they also come with significant overhead. Maintaining infrastructure, ensuring security, and achieving resilience require dedicated resources and expertise."

- Full-stack dveloper,
  Custom Software Development/IT Consulting

→ **Private vs. public cloud** is still a conversation. Some respondents recommended private clouds for cost or performance reasons—but cautioned to test thoroughly before migrating.

Advocating for private cloud

"Self hosting will keep crucial expertise close at hand. The extra people you'll employ; security analysts, infrastructure experts and so on, will be there onsite talking to the dev team, sharing knowledge and ideas. We don't realise the value of those casual interactions which contribute fundamentally to initiatives, features processes. An Azure contract is the path of least resistance, but consider the hidden costs."

- Jimmy B,
  Full-stack developer, Education/Research

"Use a solution that you can run on premises and measure the cloud costs. If the cloud costs are higher than running on premises, maintenance and problem-solving costs included, run the solution on premises."

- Architect,
  Custom Software Development/IT Consulting

"While many move into the cloud because they think, it makes life easier - but they do not consider higher costs, possible dangerous dependencies (providers tend to make the way back to on-premise much harder) and slower applications. It may be a good choice if you want to provide systems to the masses but there is probably a future for decentral solutions."

- Martin Wildman,
  Full-stack developer @ Phactum

Advocating for public could:

"You don't migrate to the cloud for saving cost, but to provide elasticity and scalability to your system. Have this always in mind and optimize for cost from the beginning."

- Anyul Rivas,
  Team lead, Software/SaaS/Cloud Solutions

"In the beginning can use one of the big ones: AWS, Azure, Google. It will be faster and easier to start and scale. Later to cut costs and improve performance can look at private cloud if it makes sense. For private clouds it is always better to test them first with some machines before migrating all. Usually it is not always there as advertised."

- Maris Birgers,
  VP, C-level @ Longo

"Always read the fine print and make sure everybody in the team understands that it is not their job anymore to operate the hardware. This is a significant MindShift that needs to happen in order for a successful shift to the cloud."

- Chris Erlich,
  VP, C-level @ Yellowhippy.com

One respondent stood out by taking a more neutral stance, weighing the pros and cons of both options:

"Cost plays a major role, especially when deciding between major providers like Google Cloud, Microsoft Azure, and AWS. Evaluate not just initial pricing but also long-term costs, including scaling, storage, and data transfer fees. While private cloud solutions offer more control, flexibility, and potentially better performance, they also come with significant overhead. Maintaining infrastructure, ensuring security, and achieving resilience require dedicated resources and expertise."

- Full-stack developer,
  Custom Software Development/IT Consulting

→ **Avoiding vendor lock-in** was also a consideration for a few respondents. Developers want the flexibility to move between providers if needed, and prefer cloud-agnostic solutions.

"Most cloud providers are aiming to lock you in to their ecosystem, and ultimately they trade off capital costs and risks for operational costs and standards. You should not assume today's costs will be tomorrow's costs and take future geopolitical and other concerns into your TCO measurements. For example, recent politics are resulting in change costs. "

- Ryan Deschamps,
  Education/Research professional

"Avoid getting married with provider, don't trust that provider offers everything you need check what they are actually offering. Even if they say they support something it may not be what you need and some other provider offers solution that works much better for you."

- Full-stack developer,
  Logistics/Transportation

→ A few also pointed out the need to **match provider choice with the project's specific goals**—there's no single right answer.

"Choosing a cloud provider depends on the needs of each project. In our experience, all cloud providers already have very high quality standards, which has made them commodities. We mainly use infrastructure, not PaaS, so we have sought out the best prices."

- Marco Castillo,
  Full-stack developer @ nube4.com

"When choosing a cloud provider, consider cost, scalability, and ecosystem compatibility with your tech stack. AWS offers the most services and flexibility, Azure integrates well with enterprise and Microsoft tools, while Google Cloud excels in data analytics and AI. Evaluate pricing models, especially for compute, storage, and networking costs, and ensure regional availability aligns with your user base. Prioritize security, compliance (GDPR, SOC 2), and managed services to reduce operational overhead. Finally, assess multi-cloud or hybrid cloud options if vendor lock-in is a concern."

- Ash-Ura Empty,
  Backend developer @ IT Consortium

→ And of course, **security and compliance** were flagged as must-haves, especially for teams working in regulated industries.

"Always keep in mind that by using a 3rd party provider you're opening yourself to possible security issues outside of your control. It doesn't matter how big and popular your provider is, they are still human and make mistakes. Try to consider whether the size of your project necessitates using a 3rd party provider instead of building an in-house solution."

- Stefan Dimitriu,
  Full-stack developer, Software/SaaS/Cloud Solutions

As one developer put it succinctly:

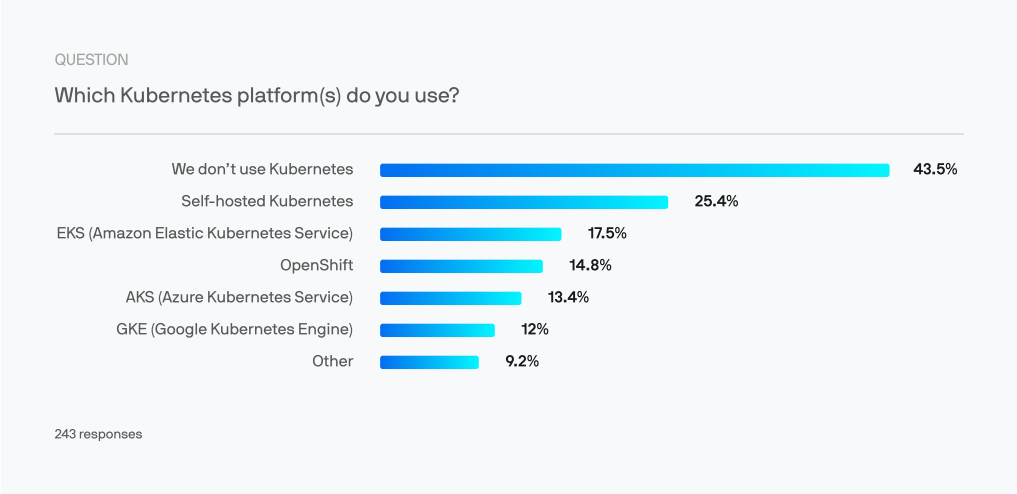
> “Pick a provider that aligns with your tech stack, scalability needs, and budget. Avoid lock-in. Reliability and pricing transparency matter more than flashy features.”
>
> — **Shaukat Mahmood Ahmad, VP, C-level @ GuCherry Blog by Everestthemes**

→ If you’re starting a new Java project, it’s worth stepping back and asking: **Which provider gives your team the best mix of control, scalability, and long-term value—without locking you in?**

Kubernetes & containerization

75.2% of respondents reported using some form of containerization. Docker without Kubernetes (39.2%) is the most common choice, showing that many prefer simpler, lightweight deployments. Kubernetes usage stands at 36%, and of those:

→ Self-hosted Kubernetes is most common (25.4%)
→ Amazon EKS leads among managed solutions (17.5%)
→ Followed by OpenShift (14.8%), Azure AKS (13.4%), and Google GKE (12%), white
→ 43.5% of respondents do not use Kubernetes at all currently.



Containers, Kubernetes & how Java teams are deploying in 2025

If you're planning to use containers in your next Java project, you're in good company——nearly 73% of respondents say they're already using Docker, Kubernetes, or both in their projects.

Here's how it breaks down:

→ 39% use Docker (but not Kubernetes)
→ 33.9% use Kubernetes
→ 12.4% are exploring containerization
→ 13.1% are not using containers at all
→ 1.6% selected Other

Among Kubernetes users, adoption is split across both self-managed and cloud-managed platforms:

→ 25.9% use self-hosted Kubernetes
→ 16.4% use Amazon EKS
→ 13.9% are on OpenShift
→ 12.7% use Azure AKS
→ 10.6% run on Google GKE

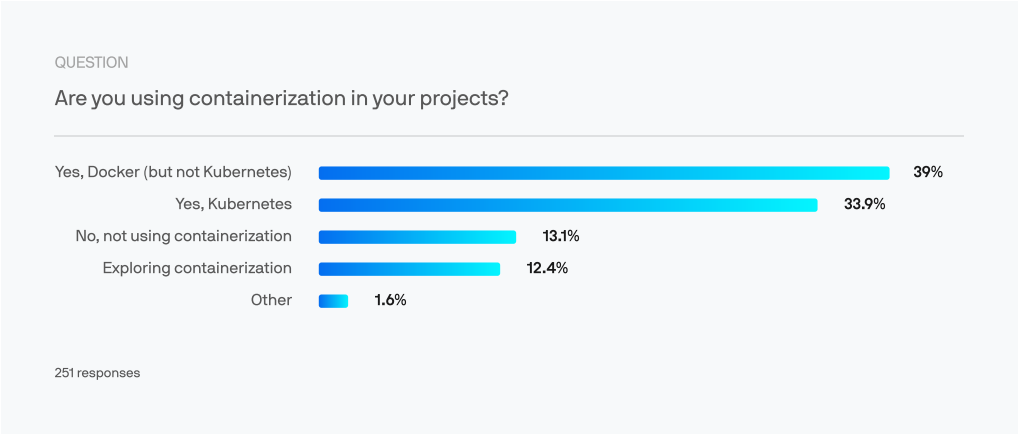
And while Kubernetes adoption is growing, it's important to note that **45.2% of respondents said they're not using Kubernetes at all**, highlighting that many teams still prefer simpler setups—or haven't made the jump yet.

"When choosing a Kubernetes platform, prioritize ease of use, scalability, and compatibility with your existing tools and workflows. Evaluate managed services (e.g., GKE, EKS, AKS) for reduced operational overhead, check for strong community support and documentation, ensure robust security features, and consider cost transparency. Test for seamless integration with your CI/CD pipelines and monitor performance to align with your long-term goals."

- Samuel Fuentes,
  Full-stack developer, Software/SaaS/Cloud Solutions

"When selecting a Kubernetes (k8s) platform as a software architect and Java expert, start by deciding between managed services and self-managed options. Managed services, such as Amazon EKS Amazon EKS and Java, Google GKE, and Microsoft AKS, offer less management overhead with automatic updates and cloud integrations, but may involve vendor lock-in and higher costs. Self-managed k8s, on the other hand, provides more control and flexibility, though it requires more expertise and resources for maintenance. Ensure the platform supports Java well, with features like pre-built Java images and easy deployment mechanisms. For example, Red Hat OpenShift OpenShift and Java is known for strong Java support, especially for enterprise applications. Evaluate scalability, security, and manageability to meet your application's needs, and consider the community and support available for troubleshooting."

- Bladimir Rondon,
  Architect @ Snap Finance LLC



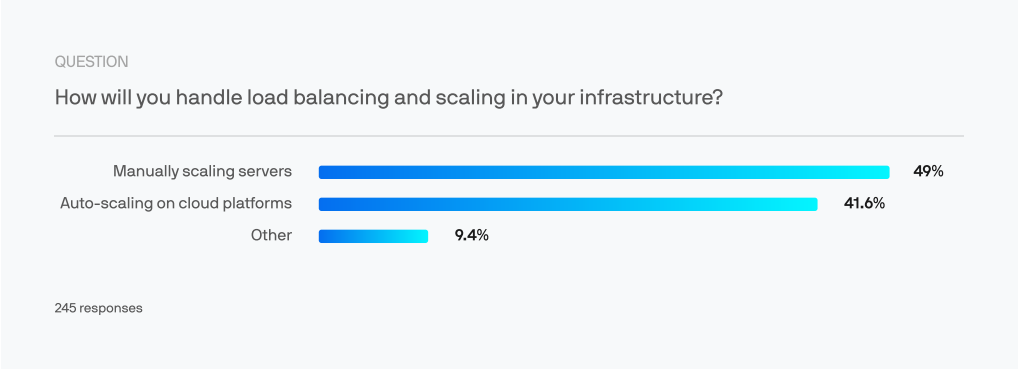
"Kubernetes has become a major topic in modern infrastructure, making it difficult to pinpoint a single best practice. However, the key considerations are similar to those when choosing a cloud provider: cost, functionality, and ease of integration. It's essential to evaluate whether a managed Kubernetes service (such as AWS EKS, Google GKE, or Azure AKS) suits your needs or if a self-managed solution is necessary. Managed services reduce operational overhead, while self-hosted solutions offer more control but require significant maintenance and expertise."

- **Full-stack developer,**
  Custom Software Development/IT Consulting

Deployment & scaling strategies

When starting a new Java project, how you scale and deploy your application will have a direct impact on how fast your team can move—and how easily you can adapt to changing demands.

Despite the increasing availability of automation tools, manual scaling is still the norm for many teams. According to our survey, 49% of respondents are still manually scaling servers, while 41.6% have adopted auto-scaling solutions on cloud platforms. The remaining 9.4% use other strategies, which may include container-based or hybrid approaches.



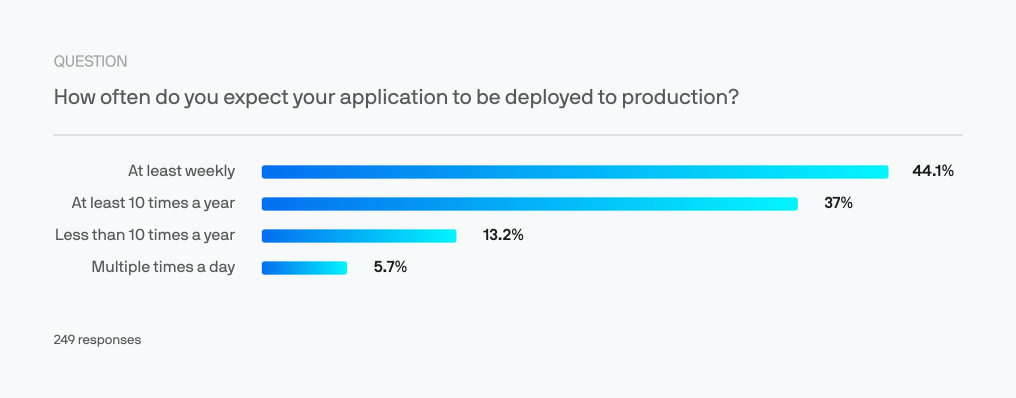
"If speed of iteration is the goal, you need deployment, security, and observability to be platform-based and baked in. The tools shouldn't slow you down."

Josh Long,
Spring Developer Advocate

While automation is gaining traction, this data suggests that many teams are still in transition—either due to legacy infrastructure, team expertise, or a desire for greater control over scaling behavior.

Deployment frequency paints a similar picture of steady, but not aggressive, iteration:

- → 44.1% deploy at least weekly
- → 37% deploy at least 10 times per year
- → 13.2% deploy less than 10 times a year
- → Only 5.7% deploy multiple times per day



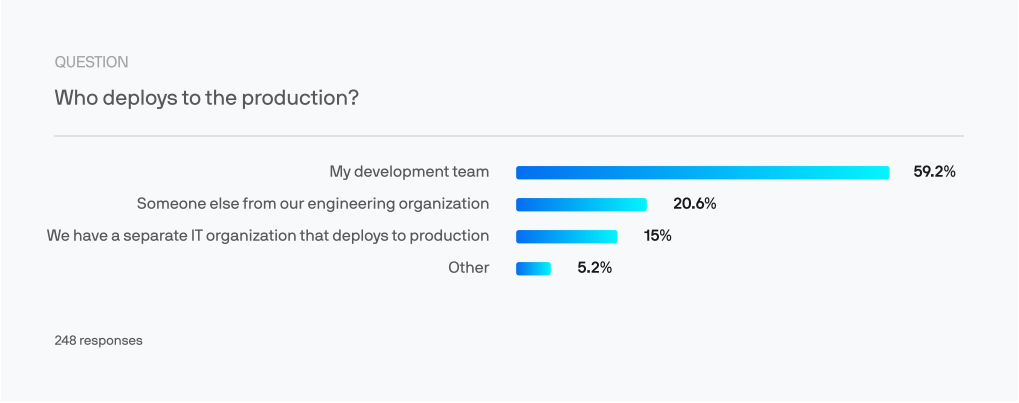
Most Java teams are following agile or CI/CD-driven practices, but ultra-frequent deployment is still relatively rare. This reflects a balance between stability and speed—where teams are modernizing, but not necessarily deploying dozens of times a day.

Who owns deployment & tooling decisions?

When starting a new Java project, it's worth thinking about who will actually own the deployment process and decide on the infrastructure tooling.

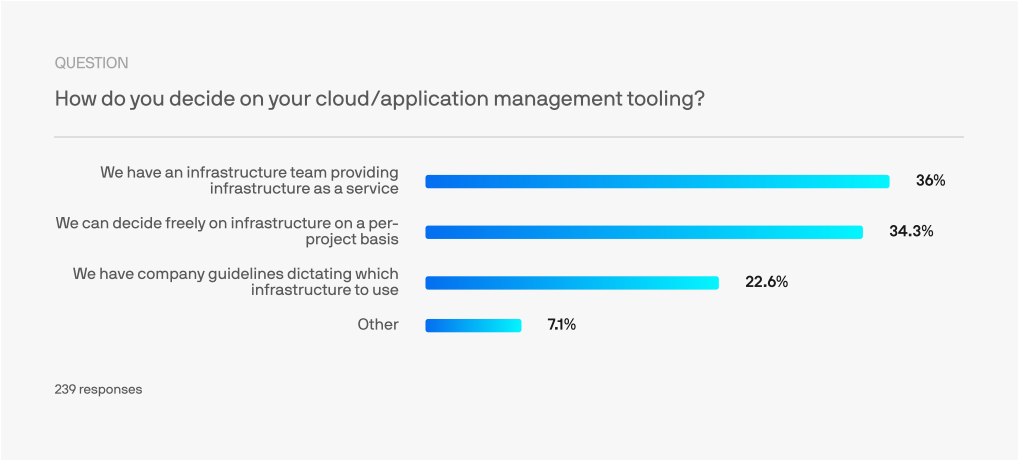
In many teams today, deployment isn't just the responsibility of a centralized IT team anymore. According to our survey:

→ **59.2% say deployment is now handled directly by development teams**, reflecting a broader shift in modern teams—where developers are increasingly owning not just the code, but how it's delivered and run.
→ 20.6% by other engineering teams,
→ 15% still rely on centralized IT,
→ and 5.2% said "Other."



When it comes to choosing cloud and infrastructure tools, a similar pattern emerges:

→ 36% of decisions are guided by infrastructure teams,
→ 34.3% are made at the project level,
→ 22.6% follow company-wide standards,
→ and 7.1% chose “Other.”



In short: developers are gaining more control, especially when it comes to choosing the tools they use every day. But there’s usually still some oversight from infrastructure or IT to make sure things stay secure, compliant, and manageable.

This shift reflects the growing role of developers in DevOps and cloud-native environments, where rapid iteration, continuous delivery, and tool ownership are key to moving fast and staying flexible.

If you’re kicking off a new Java project, it’s a great time to ask: How much freedom will your team have to choose its tools—and how will you balance that with consistency and governance across the organization?

Considerations when planning the infrastructure of a new Java project

Choosing the right infrastructure in 2025 means balancing performance, flexibility, and long-term sustainability—while also empowering your team. Here are some key questions to help guide your decisions:

→ **Cloud or on-premises?** Does your project require the scalability of public cloud, or do cost, compliance, or control needs make private cloud or on-prem a better fit?

→ **Which provider fits best?** Evaluate each cloud provider based on your tech stack, pricing transparency, regional availability, support offerings, and the potential for vendor lock-in.

→ **Are you building cloud-native from the start—or migrating over time?** Factor in how that affects your architecture, skills needed, and maintenance overhead.

→ **Do you need full Kubernetes orchestration or is Docker enough for now?** Keep in mind that many teams are still using Docker alone—and tools like Vaadin Control Center can simplify Kubernetes adoption when you're ready.

→ **How will you handle scaling?** Will your app scale automatically based on demand, or will your team manage scaling manually? What tools or platforms will support that?

→ **What's your deployment rhythm?** Are you aiming for continuous delivery, or more structured release cycles? And does your tooling stack support that cadence?

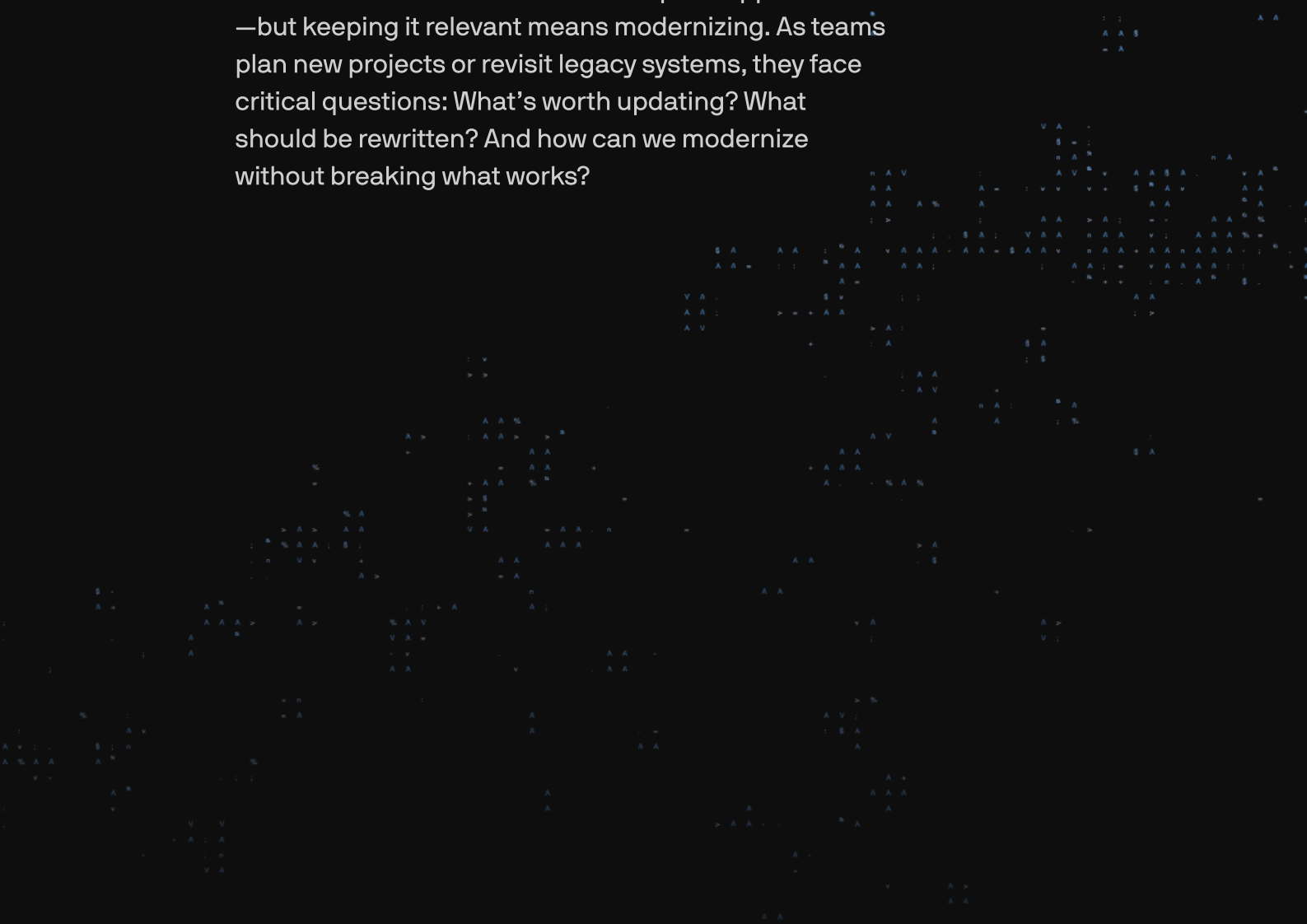
→ **Who makes the calls?** Will infrastructure and deployment decisions be made by developers, infra teams, or at the project level? Define ownership early to avoid misalignment later.

→ **Do you need a hybrid or multi-cloud setup?** If avoiding lock-in is a priority, think about portability and cross-cloud tooling from the beginning.

→ **Is security and compliance baked into your infrastructure plan?** Especially important in regulated industries, where provider trust and certifications can be a deciding factor.

Additional considerations for modernization projects

Java remains the backbone of enterprise applications—but keeping it relevant means modernizing. As teams plan new projects or revisit legacy systems, they face critical questions: What's worth updating? What should be rewritten? And how can we modernize without breaking what works?



How organizations are approaching Java modernization

55.5% of respondents said that their upcoming java project is a modernization project. 43% are starting within the next 3 months, and 24.5% are not sure when they'll get started on it. The top two answers indicate that it's either urgent, or it's a can getting kicked down the road. It's not a priority until it is, and when it is, it's a serious priority.

And nearly every organization has something that needs to be modernized:

→ 37% of respondents said their organization has 1–2 applications in need of modernization.

→ 28% said they have three or more Java applications that require updates.

→ 16% report having 10 or more outdated applications to modernize.

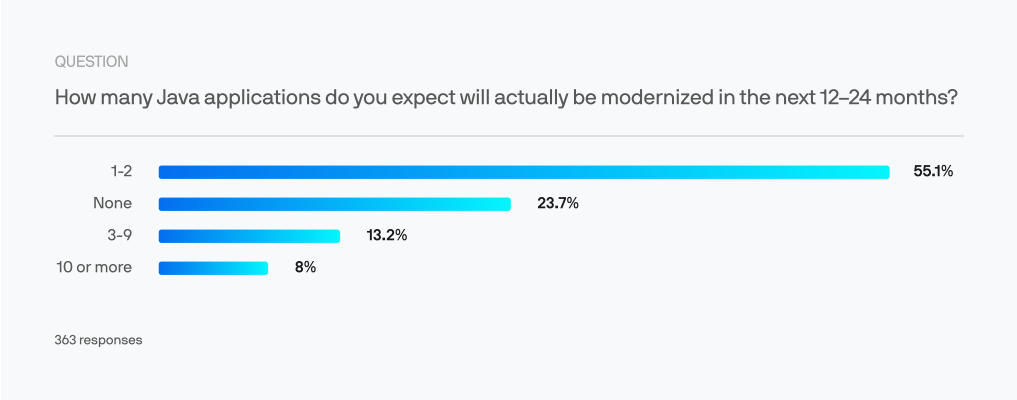
Despite this, most are taking a phased approach:

→ 55% plan to modernize only 1–2 applications in the next 12–24 months.

→ Only 13% plan to modernize 3–9 applications.

→ Just 8% expect to modernize 10 or more.

→ 24% do not expect to modernize any applications in that time.



"Refactoring isn't free, but neither is keeping legacy software simply running. Modernization will have a cost, but this is often offset by reducing/eliminating the costs of maintaining legacy software."

- David Nebinger,
  Architect @ Liferay

Insights from Java professionals on application modernization

Starting a Java modernization project is more than a tech upgrade—it's a chance to rethink how your applications deliver value to users and the business. Done right, modernization sets the stage for faster development, better UX, and long-term maintainability.

In our survey, we asked development professionals who have undertaken Java modernization projects to share their advice in their own words: "What advice would you give to someone considering modernizing an application?"

Using AI-powered analysis to categorize qualitative responses, we identified recurring themes across the dataset. The results show that successful modernization relies on **incremental approaches, thorough planning**, and maintaining a **strong business focus**. Technical considerations are important, but the most successful projects balance technical choices with business value, user experience, and team capabilities.

> If you're modernizing a Spring Boot app, it's often cleaner to start fresh, generate a new project, and migrate the code—especially when using tools like OpenRewrite. You'll know exactly what's broken, and you can fix it fast.
> There should never really be a reason to be left behind. Tools like Moderne can auto-upgrade entire organizations."
>
> 
> Josh Long,
> Spring Developer Advocate

The main keytakeaways include:

Approach modernization incrementally

The most frequently shared advice focused on taking an incremental approach to modernization—breaking efforts into smaller, manageable steps rather than attempting a full overhaul at once:

“If possible, try a break-down approach and modernize business functions one by one. Try to uncover undocumented business knowledge hidden in the legacy code and deal with it appropriately.”

- Rainer Bieniek,
  Senior lead developer, Custom Software Development/IT Consulting

“Do it in a phased manner. If possible decompose the application into it's components and modernize one piece at a time.”

- VP,
  C-level, Technology/IT Services/Consulting

"First of all, evaluate whether it's worth modernizing instead of rewriting. Then proceed carefully, step by step—never try to change everything in a few big steps."

- Full-stack developer,
  Other industry

“Modernisation is a continuous process. It is important to have clear goals of the modernisation exercise after assessing the current strengths and weaknesses of the application. Data migration is also very key, there is need to ensure data integrity and minimise downtime during transition.”

- Victor Tichayana Hokonya,
  VP, C-level @ Retail/E-commerce/Consumer Goods

“Before starting a modernization effort, think about the users and the processes. Modernization isn’t just about changing the framework—it’s about optimizing the processes within the application."

- Engineering manager,
  Technology/IT Services/Consulting

"To modernize an application, I recommend working in iterations. The application should be modernized piece by piece. This means making the old code coexist with the new code."

- Jérôme Ruillier,
  Consultant @ Université Laval

Make smart technology choices

Respondents advised doing your homework when selecting technologies—prioritizing maintainability, scalability, and organizational context over popularity:

"Think ahead. Don't try to support old and inflexible framework for longer than is good for your IT landscape. Be courageous when discussing necessary changes. Find someone in the higher echelons that supports you."

- Architect
  @ Provinzial

"Do not follow the mainstream. There are a lot of tools, frameworks, and even programming languages that come and go shortly with the fashion movement. Try to use the technologies your organization and the requirements need. "Netflix" (or someone else) is using that technology, will never be a good decision-making approach. Let the requirements command the decision process, focusing on your organization's reality, environment, and culture. Are your team members more focused on the Java side? So, use effective and efficient Java tools to solve your needs or business cases. Evaluate the maintenance costs, the team availability and proficiency in the proposed technologies, and how confident you feel with your providers and supporters for using some specified technology or tool. Keep in mind, as main directions, the security, testability, performance, and the challenges of distributed applications for operating in the cloud in an effective and trustworthy way. Matured APIs, clear business rules, and a solid, whole backend will ensure the general project's stability and future success."

- Bladimir Rondon,
  Architect @ Snap Finance LLC

"Deeply analyze the stack you choose. If you rush the decision and later realize that the stack has significant limitations or isn't the most suitable choice months or years into development, switching can be extremely difficult. Also, for project managers—listen to your development team."

- Full-stack developer,
  Technology/IT Services/Consulting

"When modernizing legacy ERP desktop applications, Vaadin should be your first platform choice—whether they're built in Java (Swing/JavaFX/SWT) or Microsoft technologies (VC++/VB/C#). In one project, we modernized a C#/WinForm healthcare CRM using Vaadin and Spring faster than estimates for Angular/.NET Core. We first consolidated business logic that was scattered across forms and singletons into Spring services/components, eliminated stored procedures, revised locking policies, and migrated SQL to Spring JPA. This restructuring would have been necessary even with .NET Core, and the switch from C# to Java didn't affect timelines. Vaadin Flow allowed us to preserve desktop functionality while cutting development time by approximately 70%."

- Full-stack developer,
  Technology/IT Services/Consulting

"Swing is useful, but in today's world, everything is web-based. When developers lack experience in web and front-end development, choosing the right web framework can be challenging. With so many options available in web, CSS, and JavaScript frameworks, it can be overwhelming to select one and learn it effectively. However, with Vaadin, we can continue developing in Java, and Vaadin takes care of transforming our application into a web application. This means there's no need to learn about various web, CSS, or JavaScript frameworks."

- Barış Şahin,
  Full-stack developer @ License4J

"Do your research. Don't choose your framework, based primarily on its current popularity. Consider long-term support you'll need to provide to your users and need yourself from other providers."

- Architect
  @ IZUM

Interestingly, a few respondents also suggested leveraging AI and automation in Java modernization efforts to help accelerate the process:

"Use Vaadin to modernize Swing apps, but also try to adapt your architecture to the web."

- Full-stack developer,
  Software/SaaS/Cloud Solutions sector

"Follow best practice and clean architecture. Use AI and Automation tools."

- Rafek Shawki,
  Full-stack developer, Technology/IT Services/Consulting

What this means for you:

→ Break your modernization into manageable phases
→ Consider a modular architecture that allows for piece-by-piece modernization
→ Don't upgrade multiple frameworks simultaneously
→ Leverage AI and automation tools to refactor legacy code faster and reduce manual effort

Vaadin Flow was recommended as a framework choice for modernizing Java applications, particularly when transitioning from desktop to web:

"If your desktop app is made with JavaFx or Swing, and you are planning to modernize it into a web application, Vaadin is a great option. It's easy to understand and you will feel comfortable working with it from the first moment."

- **Sergio Fernandez,**
  Full-stack developer @ Querry S.A.

Planning beats rushing

Many respondents emphasized the importance of thorough preparation and planning before diving into modernization efforts:

"Take your time to prepare before actually starting. Knowing the new modernization tools as well as the existing application can save time in the short and long run."

- Team Lead

"Do a proper analysis prior estimating budget and choosing proper technology. The devil is in details."

- Backend developer, Other

"Before modernizing an application, it is essential to analyze the current state and define clear objectives (performance, security, scalability). Choosing the right approach (refactoring, migration, rebuilding) while ensuring compatibility and test automation is key. Finally, involving teams and adopting a gradual modernization process helps minimize risks."

- Damlabin Nkamneu Arsene Vivien, Full-stack developer @ Université Laval

"Balance innovation with stability and avoid adopting new technologies just because they're trendy. Focus on actual business benefits rather than following trends. Don't rush to upgrade without a clear need. Do not chase a new tool, just solve a problem. Answer the question: What will we improve for the customer? Do not overcomplicate the architecture. Choose technologies with strong ecosystem and community support. Migrate in smaller steps."

- Survey respondent, Education/Research sector

Planning beats rushing

"Understand the goal - modernization is not always about replacing the old systems, but bringing flexibility to most often changed parts."

- Artur Skowronski,
  Solution architect @ Virtus Lab Sp

"Modernizing an application can be a game-changer, but it requires careful planning. Check performance, security, scalability, and maintainability issues. Evaluate dependencies and legacy components. Do you want better performance, scalability, security, or a new UI/UX? Prioritize features based on business needs and user experience. Improve code structure, adopt microservices, or API-first approach."

- Wassim Khazri,
  Backend developer, IT Services/Consulting

"Think business first, tech second. Start small—clean up what you can, break things down if needed. Consider cloud, but don't rush. Prioritize security and efficiency. Roll out changes gradually, like taste-testing a new recipe. Sometimes starting fresh beats endless patches. Remember, tech should serve your goals, not dictate them."

- Shaukat Mahmood Ahmad,
  VP, C-level @ GuCherry Blog by Everestthemes

What this means for you:

→ Conduct a detailed analysis of your current application
→ Map dependencies and potential bottlenecks before starting
→ Create a roadmap with clear milestones

One respondent offered particularly actionable advice:

"Start with a small stand-alone prototype that explores how the technology choices work together and proves that they actually do work well and have great developer experience."

- Developer
  @ Smartworks Ltd

Business value must lead the way

Respondents also cautioned against 'modernization for modernization's sake' and emphasized the importance of aligning modernization efforts with business goals and customer needs:

"Make sure modernization actually makes sense. Don’t assume microservices or the latest trend is the right answer—evaluate the business impact first."

- VP, C-level @ Celias

“Solve the problem that [the] organization and the market needs.”

- VP,C-level @ Alper Aslan

“First find out the pain points and performance bottlenecks in your existing application and identify the areas which needs to be improved. I'd suggest going through some case studies of modern applications and try to improve the UX design. I'd highly recommend the use of modern frameworks, libraries and tools that can enhance development speed and application performance. Focus on security. Engage with stakeholders throughout the modernization process to gather feedback and ensure that the changes align with business needs.”

- Tariq Khan, Full-stack developer @ Elite Software Tech

What this means for you:

→ Start by identifying specific business problems to solve
→ Quantify expected benefits in business terms
→ Get stakeholder buy-in early and maintain communication

Focus on user-centric design

Modernization efforts should be guided by the needs of real users. As many respondents pointed out, design choices should reflect how the application is used in practice:

"Become the user of your application ."

- George Gionis,
  Full-stack developer @ camatech.it

"The user experience and good architecture are the key to success."

- Mohamed Ellaouzi,
  Full-stack developer @ Fekra Systems

"Reduce technical debt. Focus on the end user. Always think about the business process and don't get lost in functionality."

- Christian Wawrzinek,
  VP/C-level @ yellowhippy.com

"I suggest that you first start with the proposed use of the application. Put your mind in the place of the end user, use logic to make the application easier to use then work backwards from there. I'm a fan of user-centric design and I believe that this is a great way to proceed!"

- Phill Borm,
  Team lead, Technology/IT Services/Consulting

"Modernizing the application is not only about using new technology; it is mainly about figuring out what improvements in user experience and usability this new application can provide to customers that were not possible/feasible due to the limitations of legacy technology."

- Backend developer,
  Finance/Banking/Insurance

"Explore Vaadin Flow framework for interactive and captivating UI components."

- **Full-stack developer, Technology/IT Services/Consulting**

What this means for you:

→ Start with the user's perspective before making technical decisions
→ Simplify workflows by aligning features with real business processes
→ Use modern UI frameworks to create intuitive, engaging interfaces
→ Don't just adopt new tech—ensure it brings tangible value to the end user

Testing provides safety nets

To minimize risk and ensure stability, many respondents emphasized the importance of a strong testing foundation throughout the modernization process:

"Improve continuously, in small increments, test and release often."

- Rafal Borowiak,
  Backend developer @ Logistics/Transportation

"Make sure you have solid regression tests (preferably automated) for the existing app before you start, as it will help define expectations for the new app and a starting point for discussions about variation from that starting point."

- Full-stack developer,
  Technology/IT Services/Consulting

"When modernizing a Java application, it is essential to carefully assess any interfaces or APIs that may require downward compatibility. You must evaluate which components interact with legacy systems to ensure that new changes do not break existing functionality. This involves implementing robust testing and, if necessary, using adapter patterns to maintain the integrity of the system. Equally important is the selection of core frameworks, as these will form the foundation of your application's future development. Take the time to research and choose frameworks that offer scalability, flexibility, and strong community support."

- Full-stack developer,
  Custom Software Development/IT Consulting

What this means for you:

→ Implement comprehensive testing before starting modernization
→ Use automated tests to verify that functionality remains intact
→ Consider parallel running of old and new systems during transition

Team skills and knowledge transfer are critical

Last but not least, a few respondents highlighted the importance of not overlooking the human element in modernization projects:

"Make sure all stakeholders are involved early on. You don't want changing requirements weeks before go-live because you forgot to ask someone how they actually use the application."

- Architect,
  Custom Software Development/IT Consulting

"When modernizing a system, it is important to create a documentation about all transactions happened when migrating the data from legacy to modern apps, this will provide a reference for developers when doing root cause analysis or searching of patterns from the legacy data."

- Full-stack developer
  @ Collaberadigital

What this means for you:

→ Assess your team's skills relative to target technologies
→ Plan for knowledge transfer from legacy system experts
→ Budget for training and potential external expertise, or look for solutions that offer vendor-backed support

The modernization roadmap

Based on practitioner experiences, we put together a pragmatic roadmap as a baseline for your next Java modernization project:

1. **Assessment phase**
   a. Analyze current application architecture and pain points
   b. Gather undocumented business rules embedded in legacy code
   c. Establish clear modernization goals and success metrics

2. **Planning phase**
   a. Choose appropriate technologies with proven track records
   b. Develop a phased approach with quick wins identified
   c. Create comprehensive test suites for the existing application

3. **Execution phase**
   a. Start with a small proof of concept
   b. Implement in incremental slices of functionality
   c. Maintain parallel systems during transition

4. **Validation phase**
    a. Thorough testing of each modernized component
    b. User acceptance testing with actual end users
    c. Performance comparison with legacy system

5. **Transition phase**
    a. Gradual cutover to the new system
    b. Monitor your application closely for issues
    c. Document new architecture and knowledge gained

Common pitfalls to avoid

Survey respondents also highlighted several traps that can derail modernization efforts:

Trying to do too much at once:

"Don't change too much at once. We changed java, WildFly and Vaadin versions in one go and that was a bit much." -

- Architect
  @ Quorum Software

Ignoring business knowledge in legacy code:

"if possible, try a break-down approach and modernize business functions one by one. Try to uncover undocumented business knowledge hidden in the legacy code and deal with it appropiazely."

- Rainer Bieniek,
  Backend developer, Custom Software Development/IT Consulting

Chasing technology trends

"Don't follow the mainstream. There are a lot of tools, frameworks, even programming languages that come and go shortly and with the fashion movement."

- Bladimir Rondon,
  Architect @ Snap Finance LLC

Java modernization doesn't have to be overwhelming. The collective wisdom from our survey suggests that success comes from being pragmatic, incremental, and business-focused.

As one respondent succinctly put it:

"Everything big that works well started as something small that worked well."

- **Full-stack developer @ FDC Solutions, Inc.**

Start your modernization journey with clear goals, a solid plan, and the right balance of ambition and caution.

Considerations: Aligning modernization with business value

→ Does management know the costs of maintaining applications that require modernization?

→ How can you best convey the risks associated with apps that require modernization?

→ Other than risk-management, how do you describe the benefits of modernization a) to users b) to management?

→ Have you developed metrics to score modernization-target-applications regarding improvements to business value, reduced risk, ease of updating?

→ Do you have a long-term modernization roadmap?

→ Are you balancing modernization efforts with new feature development?

Summary

Java remains a powerful and trusted choice for building business-critical software—but how you approach your project will make all the difference. Across the nearly 1,000 practitioners we surveyed, a few key trends stand out:

→ **Modernization is accelerating.**
Over half of Java teams are focused on modernizing existing systems—often in parallel with new development. Incremental, phased upgrades are the preferred path.

→ **Team structure matters.**
Full-stack development is on the rise, helping reduce handoffs and speed up delivery. But success also relies on strong collaboration with QA, DevOps, and UX roles.

→ **Tech stack decisions shape long-term outcomes.** Security, maintainability, and documentation quality consistently rank above popularity. Choose tools your team enjoys working with—because productivity and morale go hand in hand.

→ **Cloud strategy is not one-size-fits-all.**
Teams are choosing infrastructure based on real-world factors like compliance, cost, and existing systems—not trends. Hybrid and containerized setups are growing, but many still rely on private or on-prem solutions.

→ **Developer input drives better results.**
Collaborative tech decision-making is now the norm—and teams that involve developers early tend to move faster and build smarter.

→ **Budgeting is about more than cost.**
The most effective teams tie budgets to outcomes, plan for hiring and skills gaps, and build in flexibility for change.

If you're starting a Java project in 2025, this report should give you both a sense of where the ecosystem is heading—and practical insights from people doing the work today. Plan with your business goals in mind, structure your team for speed and sustainability, and choose technologies built to last.

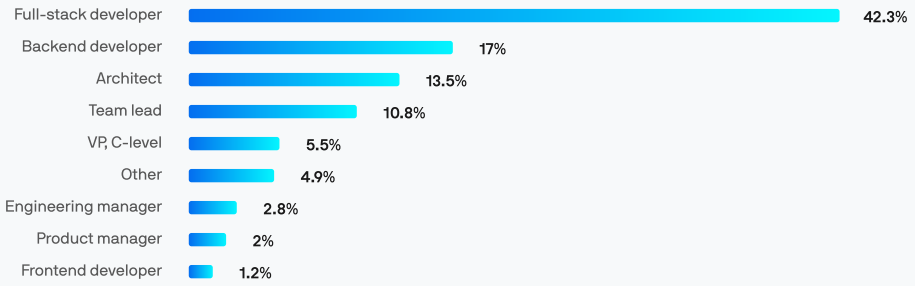
The tools are there.
The talent is there.
The key is starting smart.

Appendix: Demographics

This report is based on a global survey of 1,000 technology leaders and practitioners in organizations of all sizes across a broad set of industries.

QUESTION

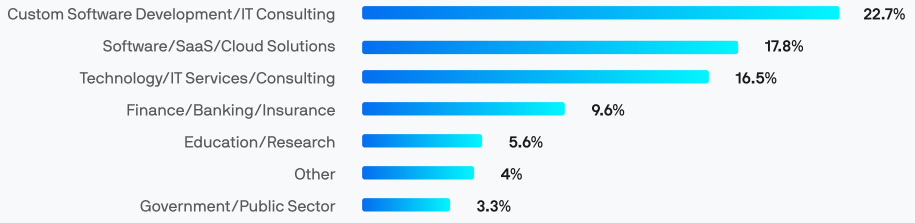
Which of the following best describes your role in the organization?



631 responses

QUESTION

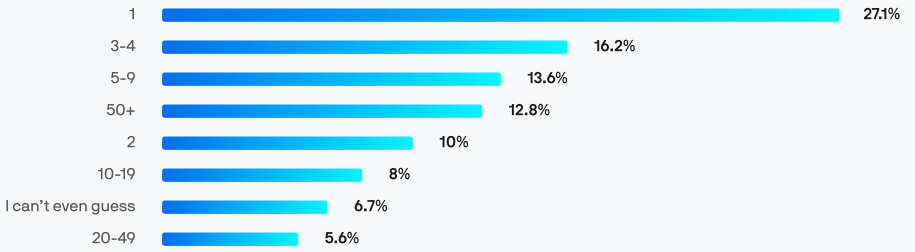
Industry of your company



559 responses

QUESTION

Approximately how many development teams are working on Java based applications in your organization? (it's ok to estimate)



546 responses

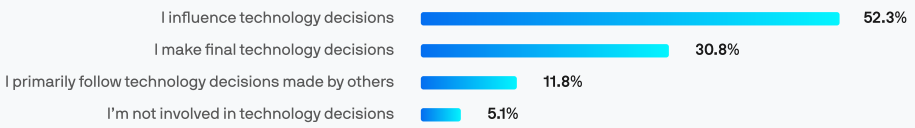
QUESTION

Do you have influence over the budget decisions for upcoming Java projects?

540 responses

QUESTION

Do you make technology decisions, influence them, or primarily follow decisions made by others?



534 responses

In what country is your organization's headquarters?

Germany	16%	91 responses
United States	11.1%	64 responses
Italy	6%	34 responses
France	3%	17 responses
Russian Federation	3%	17 responses
Spain	3%	17 responses
Switzerland	2.8%	16 responses
Canada	2.8%	16 responses
India	2.7%	15 responses
Poland	2.3%	13 responses
Brazil	2.3%	13 responses
United Kingdom	2%	12 responses
Finland	2%	12 responses
Netherlands	1.8%	10 responses
Slovakia	1.8%	10 responses
Argentina	1.6%	9 responses
Austria	1.6%	9 responses
Mexico	1.6%	9 responses
Hungary	1.4%	8 responses
Norway	1.4%	8 responses
Indonesia	1.3%	7 responses
Denmark	1.3%	7 responses
Romania	1.3%	7 responses
Sweden	1%	6 responses
Czech Republic	1%	6 responses
Pakistan	1%	6 responses
Ukraine	0.9%	5 responses
Turkey	0.9%	5 responses
Australia	0.9%	5 responses
Greece	0.9%	5 responses

Algeria	0.7%	4 responses
Latvia	0.7%	4 responses
Colombia	0.7%	4 responses
Philippines	0.7%	4 responses
Iran	0.7%	4 responses
Luxembourg	0.7%	4 responses
Kenya	0.7%	4 responses
Morocco	0.6%	3 responses
Japan	0.6%	3 responses
Croatia	0.6%	3 responses
Estonia	0.6%	3 responses
China	0.6%	3 responses
Guatemala	0.6%	3 responses
Uruguay	0.6%	3 responses
Belgium	0.6%	3 responses
Portugal	0.6%	3 responses
Israel	0.6%	3 responses
Zimbabwe	0.4%	2 responses
Bangladesh	0.4%	2 responses
Bulgaria	0.4%	2 responses
Chile	0.4%	2 responses
Ecuador	0.4%	2 responses
Ghana	0.4%	2 responses
Nigeria	0.4%	2 responses
Saudi Arabia	0.4%	2 responses
Singapore	0.4%	2 responses
Taiwan	0.4%	2 responses
Tunisia	0.4%	2 responses
Vietnam	0.4%	2 responses
Jordan	0.2%	1 response
Paraguay	0.2%	1 response
Peru	0.2%	1 response

Ivory Coast	0.2%	1 response
Ireland	0.2%	1 response
Iraq	0.1%	1 response
Ethiopia	0.1%	1 response
El Salvador	0.1%	1 response
Rwanda	0.1%	1 response
Belarus	0.1%	1 response
Angola	0.1%	1 response
Dominican Republic	0.1%	1 response
Slovenia	0.1%	1 response
South Africa	0.1%	1 response
Cyprus	0.1%	1 response
Sri Lanka	0.1%	1 response
Cuba	0.1%	1 response
Congo	0.1%	1 response
Syria	0.1%	1 response
Uzbekistan	0.1%	1 response
Tanzania	0.1%	1 response
Albania	0.1%	1 response
Bosnia and Herzegovina	0.1%	1 response
Bolivia	0.1%	1 response
United Arab Emirates	0.1%	1 response
Moldova	0.1%	1 response
Montenegro	0.1%	1 response
Malasya	0.1%	1 response
Malawi	0.1%	1 response
Benin	0.1%	1 response
Lebanon	0.1%	1 response

Total: 572 responses

Vaadin is an open-source platform for building modern web apps in Java.

With a powerful set of accessible UI components, two proven frameworks—**Vaadin Flow** (100% Java) and **Hilla** (a full-stack React framework for Java)—plus design-to-code and modernization tools, Vaadin helps teams build secure, scalable, and maintainable business applications faster and with less frontend complexity.

More than 1,500 companies worldwide—from startups to global enterprises—rely on Vaadin to build modern, user-friendly Java applications.

Ready to build your next Java project?

Discover how Vaadin can support your team—whether you're starting fresh or modernizing an existing Java application. Get started in seconds at **start.vaadin.com**, or **contact us** for more information.