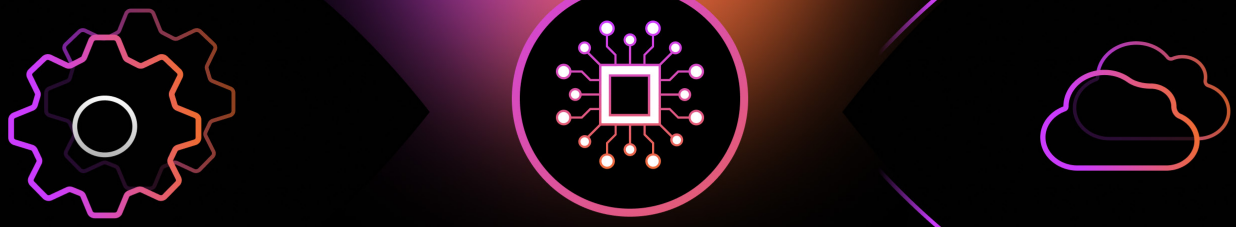
AI PULSE REPORT

AI adoption and its impact on developer productivity

2026



Octopus Deploy

AI Pulse Report

AI adoption and its impact on developer productivity

Ed.1.2026.1

Executive summary

AI is transforming how software teams and their organizations develop their operational strategies. Integration of AI tools into developer workflows is occurring more rapidly every day as organizations learn how to implement and use AI effectively. The AI Pulse report combines primary survey data, industry data, and an extensive review of current literature. Our survey included a worldwide sample of technology professionals across various industries.

Organizations are currently experimenting with AI to determine where it works best and how to implement it effectively. During this phase, organizations often prioritize short-term productivity, which can be detrimental to their longer-term outcomes. One impact we predict is a shortage of skilled senior developers.

In this report, we provide evidence of a three-phase trend in the software delivery workforce.

Phases of AI adoption impact on software development talent pipeline

- **Phase 1** (2024-2025): *Experimentation*
  The current era, during which organizations have broadly adopted AI across a variety of development tasks, assessing its most beneficial attributes, and current perspectives of AI's impact in the industry.
- **Phase 2** (2025-2027): *Hiring freeze*
  An emerging junior hiring freeze as companies pursue the advancement of their AI strategy, implementing it further, while also retaining senior developers.
- **Phase 3** (2027-2030): *Skills shortage*
  A senior developer shortage due to the reduction in hiring junior developers.

This report presents an analysis combining primary survey data with leading third-party research. We look at where AI excels and where it has limitations for individuals and organizations. You'll also learn about the impact of AI on hiring trends, the likely long-term impact on the software industry, and recommendations for a sustainable path forward.

The decisions you make today will have a significant effect on the future of the software industry.

The AI productivity promise

The software industry has adopted AI with remarkable enthusiasm, delivering significant productivity gains across numerous organizations and industries. AI productivity is measured by how much employees and organizations accomplish with AI tools, including impacts on performance and well-being[1]. According to the 2025 **AI at Work report**:

- 74% of executives view AI as critical to their company's success
- 60% say their company is aggressively using AI to innovate in their industry
- 77% have already implemented formal AI training programs

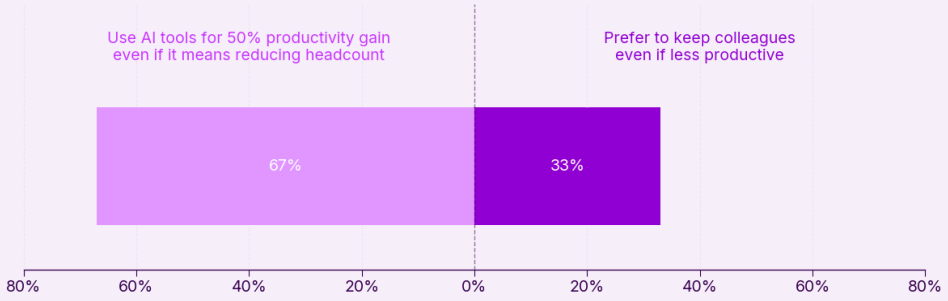
Only 1% of executives reported no AI use[2] and 92% of companies plan to increase their investments in AI over the next 3 years[3].

Executive perspectives on AI adoption. Source: 2025 AI at Work report

Organizations are investing heavily in AI tools for internal productivity, spending $500 to $1,000 per developer annually, with some allocating 1-8% of their total revenue, which means they are spending more on AI than on marketing[4].

These investments result in developers reporting increased productivity, workflow efficiency, job satisfaction, and code quality while reducing individual burnout[5, 6]. A case study by **GitHub CoPilot** found that 60-75% of users report feeling more fulfilled in their jobs and less frustrated when coding, and 88% feel generally productive[7].

When asked to choose, 67% of executives said they would prefer to use AI tools to increase productivity by 50%, even if it means reducing their workforce. In comparison, 33% would rather maintain their current team size, even without the productivity gains[2].



Executives choose AI productivity over team retention. Source: 2025 AI at Work report

When using AI, organizations benefit from enhanced documentation quality, faster code generation, and reduced technical debt[5, 6]. Despite these gains, only 1% of leaders rate their companies as "mature" on the deployment spectrum, suggesting a significant gap between promise and reality[3].

One cause of this gap is when AI's capabilities don't match the organization's needs. AI is capable of tasks junior developers traditionally handle (like generating boilerplate code, writing unit tests, debugging, and answering routine questions)[8]. However, it struggles with senior-level responsibilities, including architectural decisions, evaluating business context, subtle code reviews, and responding to production incidents[9].

The convergence of efficiency pressures with AI's capabilities are reshaping how work is done and blurring traditional role boundaries. Using signals on AI adoption, historical workforce patterns, and skill development reveals risks in the current industry landscape, and highlights the importance of Continuous Delivery in unlocking the benefits of AI.

Theory of constraints

Many applications of AI result in local efficiencies that don't lead to an associated improvement in the end-to-end flow of work. The Theory of Constraints tells us that systems are limited by the presence of one or more constraints. Optimizations made after the bottleneck have no benefit, and those made before it often degrade the system's performance by overloading the constraint. You can only improve a system by working on the constraint.

Applying the theory of constraints provides a framework for understanding why AI-driven productivity gains don't translate into proportional delivery improvements when used outside of a constraint[10]. This can manifest as limited development resources, slow feedback loops, inadequate testing infrastructure, or dependencies on external systems[11].

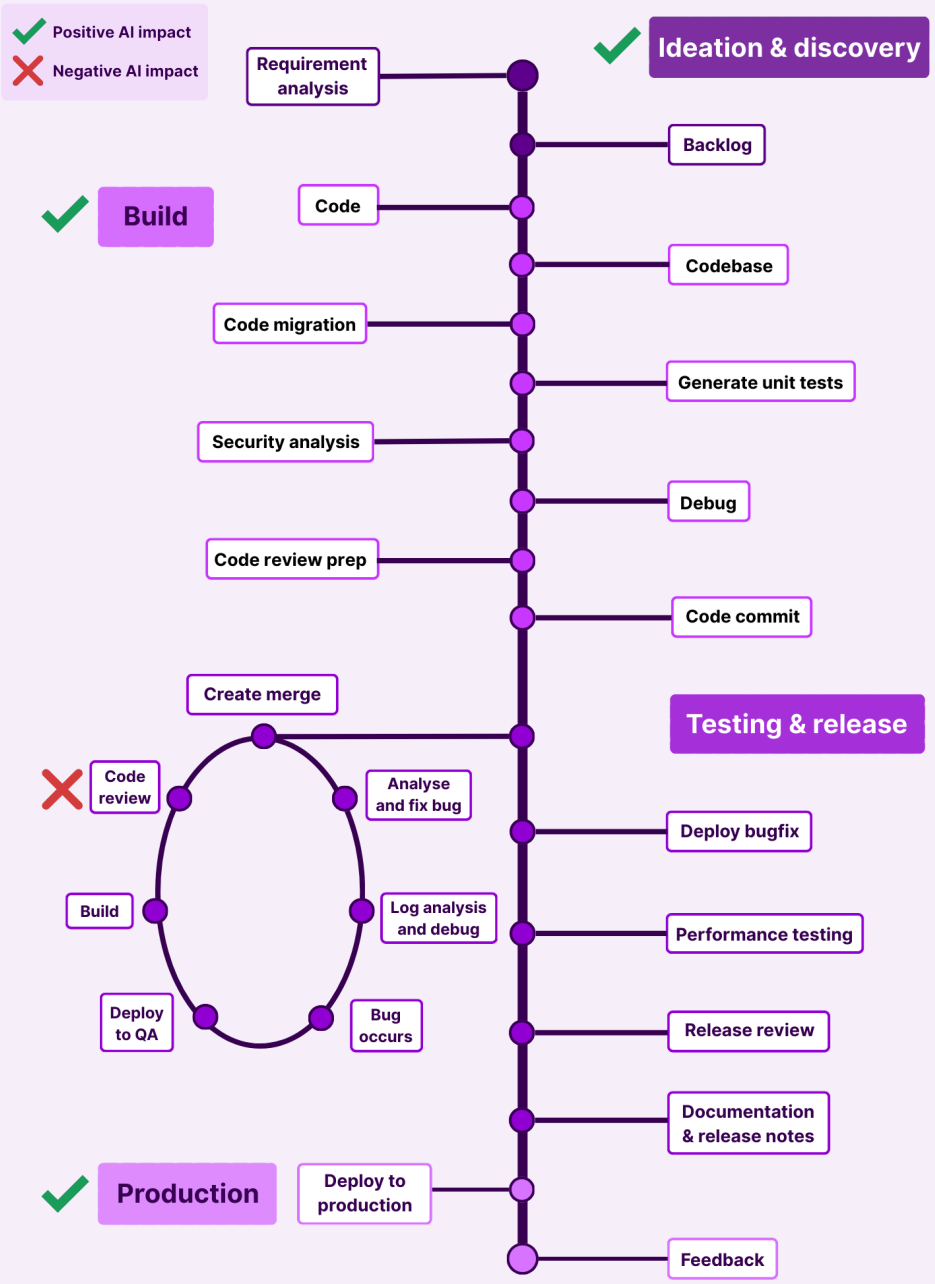
Value stream management

Value stream management (VSM) is an approach for visualizing, evaluating, and refining the flow of work through a system. Teams map out tasks collaboratively to document the entire workflow from start to finish to reveal the system's underlying structure and pathways. This baseline visualization uses systematic analysis to identify bottlenecks and eliminate waste[6].

The **DORA 2025 State of AI-assisted Software Development report** demonstrates that effective VSM implementation enhances team performance, allows more valuable work allocation, and improves overall performance. Through VSM, organizations can identify where collection costs accumulate, where delays and rework occur, and, crucially, where the system fails to make the most of AI-assisted acceleration[6].

We can model the system-level impact when AI is integrated into the value stream (like generating code, testing, release, and production processes), considering what AI excels at (such as high-volume routine tasks, pattern matching, and generating boilerplate code)[8].

The example value stream below models AI integration throughout the software delivery pipeline, demonstrating expected constraint dynamics and accelerated tasks for individual processes, which in turn creates longer queues at unchanged bottlenecks. This model has been scaffolded from multiple sources[6, 12, 13, 14, 15 & 16].



Overview of software delivery pipeline value stream with AI integration

AI implementation reduces task hours for all individual task areas of the software delivery pipeline, except for code review processes. This pattern consistently emerges across organizations, with AI accelerating code generation and all processes involved in the build/development stage. Still, the review process, constrained by the availability of senior developers, becomes an increasingly severe bottleneck as the volume of AI-generated code overwhelms the review capacity.

Code review constraint

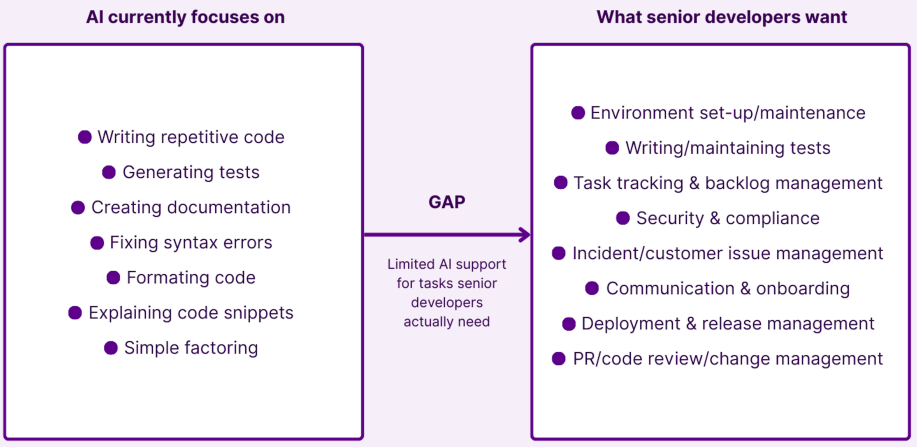
Say you have 3 senior developers supporting 12 team members. Each senior developer can effectively review 10-15 items a day, creating a maximum theoretical daily throughput of 30-45 reviews.

When AI lets developers produce code faster, the review queue grows proportionally while review capacity remains fixed. The result is a system where local optimization (faster code creation) degrades overall performance (longer delivery times). AI produces larger change sets, which increases the size of code reviews, often making them conceptually too large to be understood well.

The **DORA 2025 State of AI-assisted Software Development report** found that 72% of developers use AI for writing new code, while only 56% use AI for code reviews. Increasing the size and volume of code reviews adds more pressure to the code review process.

The focus should not be on speed, but instead on flow, focusing on using AI to improve the code review process itself, rather than generating more code, which exacerbates the bottleneck[6].

This finding is consistent with a study by DX, which found that senior developers, while appreciating faster code generation, actually desire automation for non-coding tasks like code review, compliance, security, and communication. These are areas where AI currently offers limited support, seen in the figure below[9].



Current AI focus vs. developer needs

When we consider what developers actually need (like environment setup/maintenance, automated testing, and deployment automation), a more sustainable approach emerges, which is to strengthen the underlying delivery infrastructure through automation.

Automation as the foundation for AI

AI scales the groundwork, which means that organizations with developed, automated pipelines see AI accelerate their delivery; and those with manual processes see it amplify existing issues. AI thrives on structure, with automation surfacing as a very real solution to fulfilling AI's full potential.

Research supports this pattern, with **MIT's Enterprise AI maturity study** finding that organizations in the early stages of AI adoption performed below their industry average. The distinction was whether organizations have the automation infrastructure AI needs to function effectively[17]. **The Widening AI Value Gap** research report by BCG found a similar trend, with 74% of companies struggling to scale AI value, with only 21% of pilots reaching production. The other 5% generating real returns had first built fit-for-purpose technology architecture and data foundations[18].

74%

of companies struggle to scale AI value

These findings highlight the importance of Continuous Delivery, which is the ability to release changes to production safely, quickly, and sustainably. The ultimate goal of Continuous Delivery is to make deployments predictable and on demand by keeping code continuously deployable, which eliminates traditional integration, testing, and hardening phases post-development[19]. DORA research has consistently shown that high-performing teams achieve results through practices of Continuous Delivery (like CI automation, build automation, deployment automation, observability, and auditability)[5, 6].

Continuous Delivery creates the structured, repeatable foundation that AI requires to deliver value. Some Continuous Delivery practices can be enhanced by AI (like code review, advanced testing, and scanning tools). Others, like deployment automation, require high determinism, though AI can still assist when deployments fail.

The **DORA Impact of Generative AI in Software Development** report found that as AI adoption increased by 25%, delivery throughput decreased by 1.5% and delivery stability dropped by 7.2%. This decline was not due to AI failing, but because organizations lacked the foundational knowledge and practices necessary to implement AI effectively[5]. By 2025, AI adoption increased throughput as organizations matured these practices[6].

Implementing AI without the necessary practices and maturity processes will not deliver the expected outcomes.

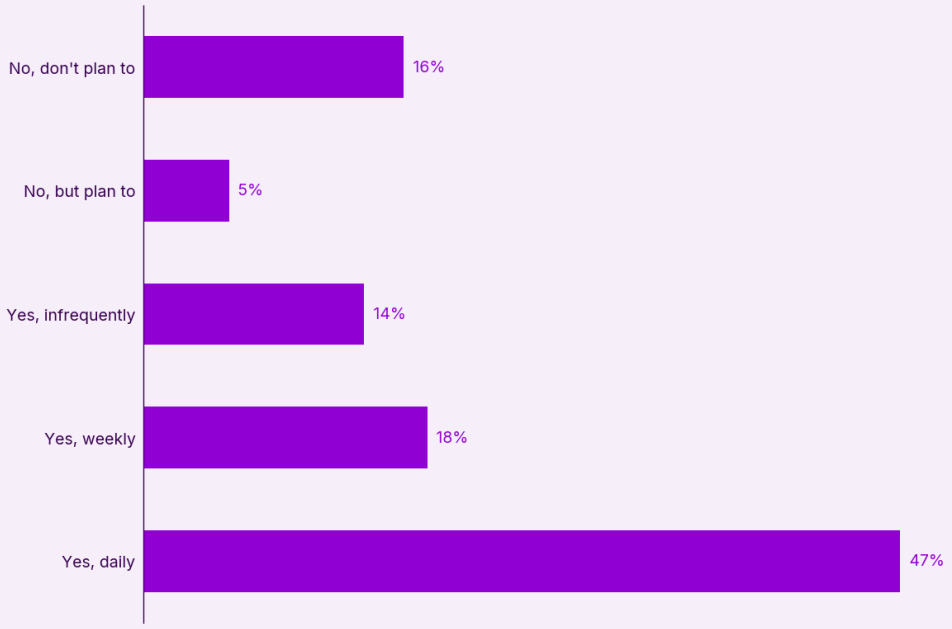
Organizations with highly automated pipelines are better positioned to let the benefits of AI take shape. Speed at scale requires a solid foundation, and those relying on manual processes or DIY scripts may struggle to convert AI developer productivity into tangible business outcomes.

Phase 1: Experimentation era

Current AI integration represents rapid experimentation and learning across the industry. This section examines AI adoption patterns, differential use across developer experience levels, perspectives, and challenges where AI assistance falls short.

AI adoption

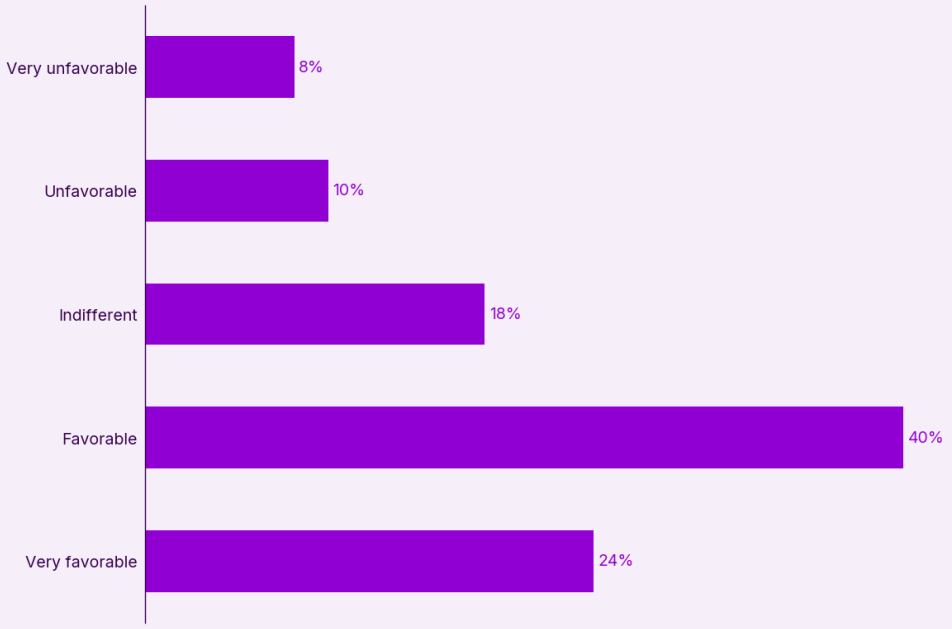
The **DORA 2025 State of AI-assisted Software Development report** found that 90% of technology professionals use AI at work, with 60% reporting reflexive AI use for problem-solving, which means they turn to AI tools as their first response when encountering challenges[6]. This is aligns with data from the **2025 Stack Overflow survey**, which found that 47% of respondents use AI tools daily in their development process, with another 32% using them somewhat frequently[20].



Do you currently use AI tools in the development process? Source: 2025 Stack Overflow survey

Many developers have moved past the trial stage and now use AI daily or even hourly. More than two-thirds of developers have tried ChatGPT or GitHub Copilot for coding, and now 49% use these tools regularly[21].

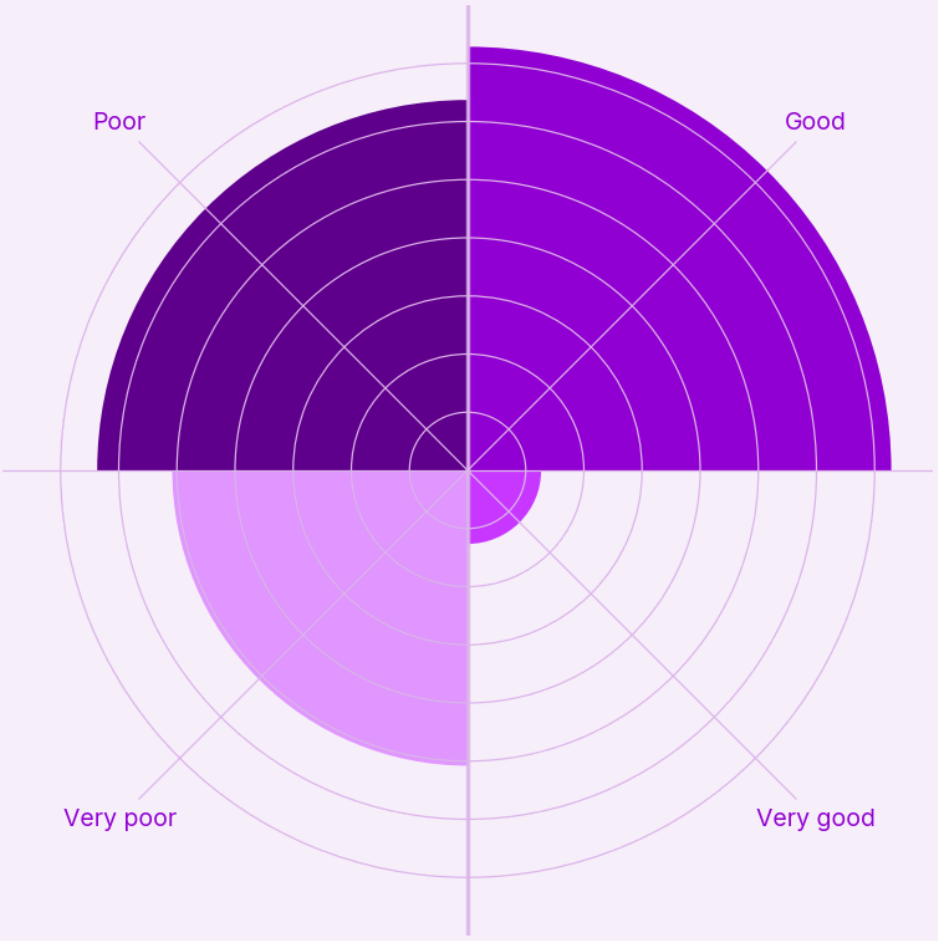
Trust in AI is growing, corresponding to its rapid adoption. In 2024, 39% of professionals had little to no trust in AI. By 2025, that number dropped to 30%[5, 6]. The trend is corroborated by the figure below, showing that 64% of respondents have a favorable view of AI integration in their development workflow.



How favorable is your stance on using AI as part of your development workflow? Source: 2025 Stack Overflow survey

The rise in confidence coincides with real productivity gains. The **2024 State of Developer Ecosystem report** found that 67% of respondents believe they spend less time searching for information and 58% believe they code faster when using AI tools[21].

Performance varies significantly, however, with task complexity. The **2025 Stack Overflow survey** found that when handling complex tasks, 57% of respondents rated AI performance as "poor" or "very poor", while 36% reported that AI is "good" at handling complex tasks. Only 6% rated it as "very good" at complex tasks.



How well do the AI tools you use in your development workflow handle complex tasks? Source: 2025 Stack Overflow survey

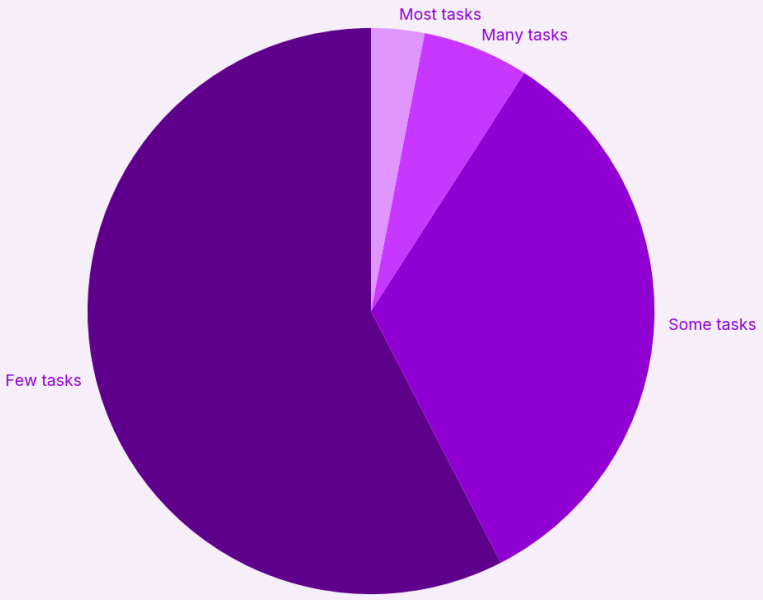
This suggests that while AI is proficient at routine tasks, its application to complex context-driven challenges remains inconsistent.

Practitioners report that successful use of AI depends on appropriate preparation steps.

> You need to ensure it has excellent context and a solid plan, and then it can do a lot of the heavy lifting.
>
> *Lead software engineer, Technology industry*

There are different adoption levels based on the task-level effectiveness of AI. A developer may use AI to write unit and integration tests, but not for exploratory testing. This demonstrates that AI can be effective in well-defined context-driven circumstances, but is less helpful for tasks that require creative problem-solving and intuition.

Additionally, how well AI handles tasks of different complexities is a contributing factor in how many tasks can be assisted by AI. A LinkedIn poll found that 57% of respondents report that AI can only handle a "few tasks" that make up their job role, with another 33% saying it handles just "some tasks".



What percentage of work could AI realistically handle today? Source: LinkedIn Poll

This indicates that AI is less effective for the majority of daily tasks. This is crucial to inform organizations not to overrate AI's readiness for broader adoption yet, based on its ability to tackle isolated, complex tasks.

Despite the heavy investments organizations are making in AI, structured implementation strategies are unclear. Only 11% of companies altogether ban third-party AI tools, while 29% let developer teams use AI however they want[21]. This unstructured approach helps teams learn quickly, but suggests companies are still developing their strategy for AI-integration[21].

As one industry expert explains:

> A lot of companies spent the first few years just getting tools into developers' hands. The shift in the last six months has been toward agentic-style systems, which lend themselves to more structured organizational implementation.
>
> *David Stenglein, Strategic Advisor & Platform Consultant at Missing Mass LLC*

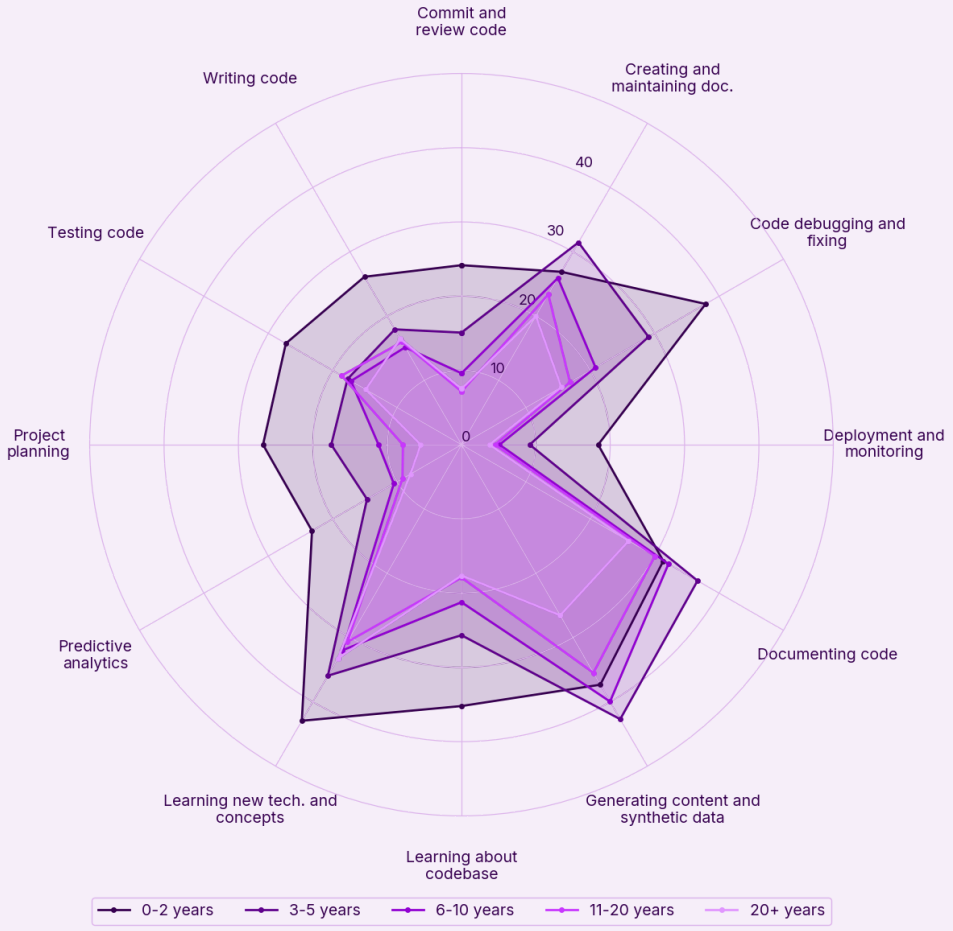
While broad experimentation with AI is essential for learning, long-term success comes from offering teams a curated set of approved tools and focusing implementation on specific areas where AI demonstrates clear end-to-end value.

Experience and AI use

The **2024 Stack Overflow Annual Developer survey** indicates that senior developers spend 30% of their time on coding, with the remainder allocated to communication, mentoring, technical decision-making, and project coordination[22]. In contrast, junior developers spend 40% of their time implementing straightforward features, 25% writing tests, and 20% on bug fixes, tasks where AI provides more assistance[22].

The figure below illustrates the current use of AI for tasks across various experience levels, as measured by years of coding experience. The task option "search for answers" was included in the **2025 Stack Overflow survey**. We removed it from our analysis as it represents a step that is required to accomplish another task, like coding assistance, finding documentation, or debugging, rather than being an end goal in itself.

Developers with 0-2 years of experience demonstrate the highest use across most tasks, with particularly high peaks for "learning new technologies and concepts", "learning about codebase", and "code debugging and fixing". These activities are foundational to skill acquisition[23, 24]. When AI mediates these learning experiences, it raises questions about the depth of knowledge transfer compared to direct experience.



Current AI use for tasks across experience levels (years of coding). Source: 2025 Stack Overflow survey

Developers with 3+ years of experience display relatively consistent AI use patterns across most tasks. Notably, the group with 20+ years of experience exhibits lower overall AI use, except for "learning new technologies and concepts" and "writing code".

Both the newest and most experienced developers rely on AI for learning, though likely for different reasons. Juniors use it to acquire foundational knowledge, while most experienced developers use it to stay current with emerging technologies.

Despite widespread adoption across all experience levels, its impact is disproportionately concentrated among junior developers (0-2 years), potentially affecting their skill development. Experts in the software development industry have offered another explanation for high AI adoption among juniors:

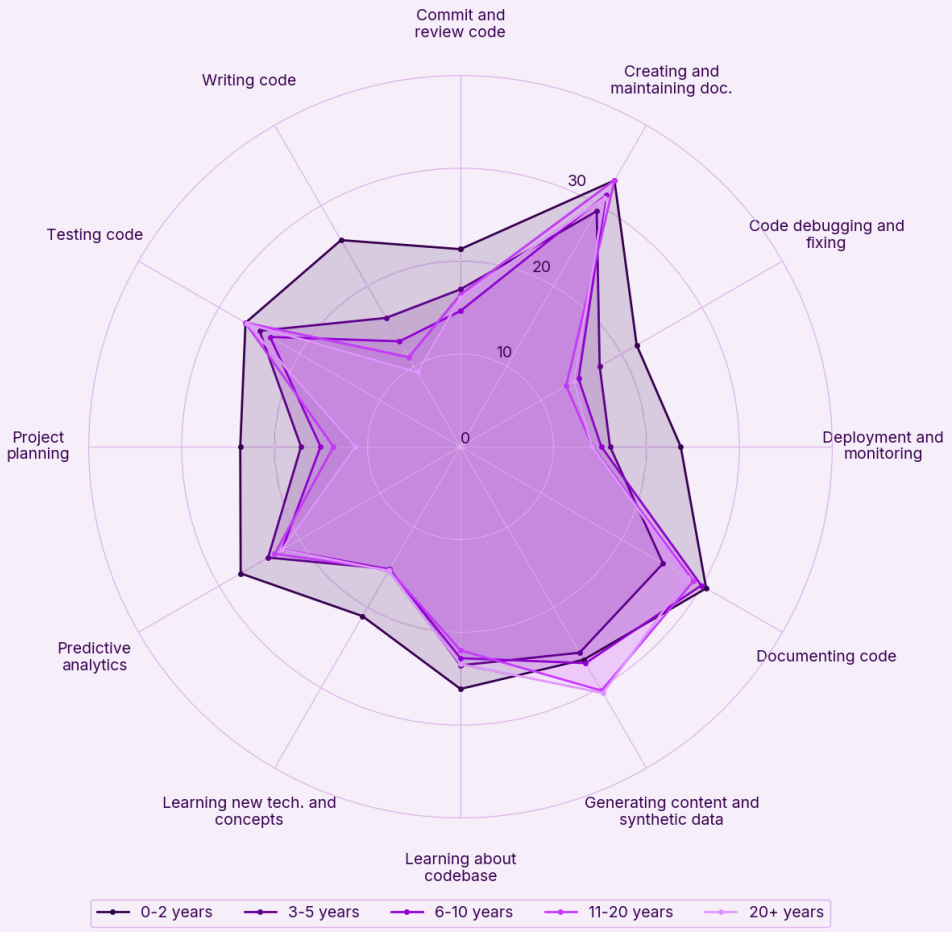
> Juniors are open-minded. And this open-mindedness also comes from the fact that they haven't seen everything in this development world and haven't picked up biases.
>
> *Ivan Krnic, Director of Engineering at CROZ*

This suggests that junior developers readily adopt AI, not because of resistance to traditional methods, but rather because they lack the baseline experience necessary to critically evaluate its outputs.

Analysis of planned AI use reveals a shift in patterns, as seen in the figure below. Developers with 3+ years of experience plan to focus AI use on support tasks like "testing code", "creating and maintaining documentation", "generating content and synthetic data", and "documenting code", suggesting that this lets them focus on higher-level activities[25].

Developers with 20+ years of experience show higher planned use across 8 task categories compared to other experience groups, suggesting that this group anticipates increasing their AI adoption. This could signal that experienced developers, having evaluated and familiarized themselves with AI's capabilities, see potential for broader applications.

Junior developers (0-2 years) show the highest planned AI use across most tasks, beyond coding to advanced tasks like "predictive analytics", "deployment and monitoring", and "project planning". These activities typically require a deep understanding of system behavior and production environments, which are skills usually gained through hands-on experience[26]. This trajectory could indicate juniors' view AI as a stepping stone as their expertise grows.



Planned AI use for tasks across experience levels (years of coding). Source: 2025 Stack Overflow survey

Research from GitHub shows junior developers experience the most significant productivity gains from AI (i.e., Copilot), completing tasks up to 55% faster[7]. IBM corroborates this finding, showing that less-experienced programmers gain more speed and learning velocity from AI than their senior counterparts[27].

This differential impact has profound implications for the workforce, where junior developers appear highly productive despite the possibility of not fully understanding the underlying processes of their work. This perceived productivity is deceptive because both juniors and their managers mistake AI-generated output for genuine competence. Some studies show developers believe they save 20-24% of their time with AI, when in fact they're actually 19% slower on complex tasks[28].

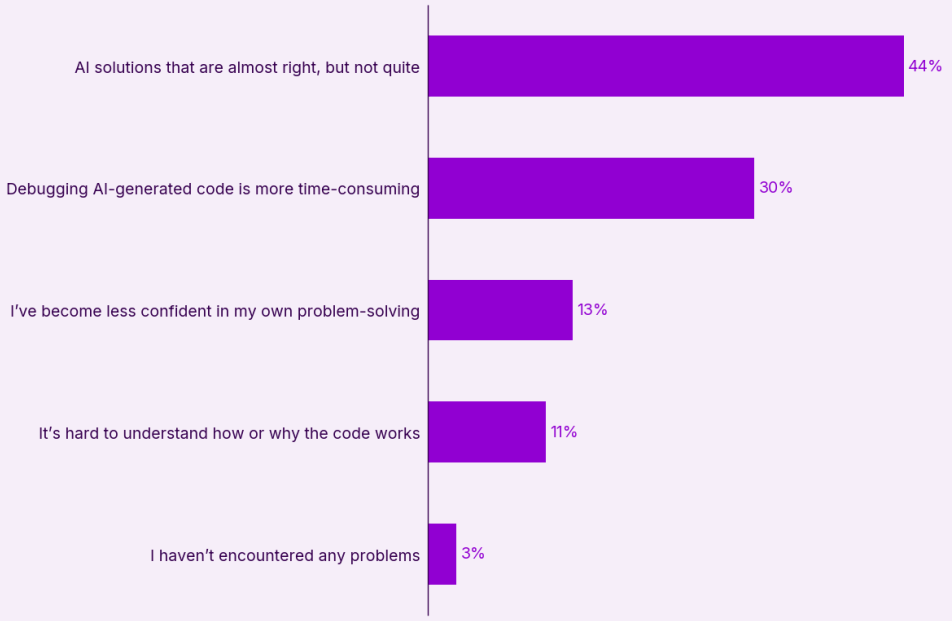
19%

slower on complex tasks with AI

The result is a hidden burden on senior developers, who must review the increased code volume and clean up AI-generated code. This false confidence may result in a skill deficit, which will only surface when complex problems require genuine understanding.

Current perspectives on AI

The data from the **2025 Stack Overflow survey** shows 44% of respondents are frustrated with "AI solutions that are almost right, but not quite", and 30% report that "debugging AI-generated code is more time-consuming". The size of these responses is interesting, as previously we have seen that AI is being used extensively for code debugging and fixing across all experience levels.



When using AI tools, which of the following problems or frustrations have you encountered? Source: 2025 Stack Overflow survey

Using AI also leads to some respondents reporting a reduction in confidence in their problem-solving abilities (13%). This signals an over-reliance on AI that may hinder skill development in developers and suppress skill retention among more experienced developers.

Beyond a skills gap, this suggests a shift in developer mindset, creating a feedback loop in which developers increasingly depend on AI systems that require human judgment while feeling less confident in providing it. AI remains an assistant, not a solution, and as AI introduces additional maintenance and complexity, gaps in these skills may negatively affect software quality and innovation.

Phase 2: Junior hiring freeze

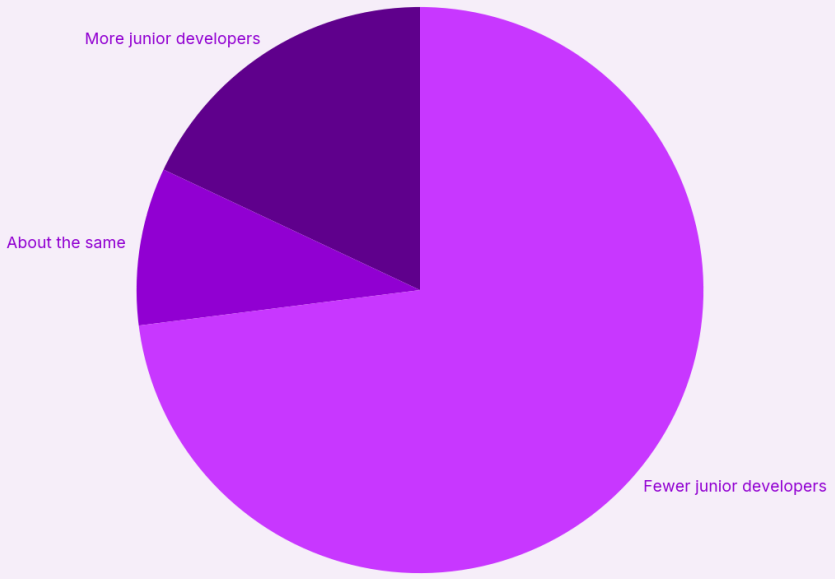
Organizations are reevaluating the value of junior developers, by reducing the entry-level positions available as teams restructure their operations around AI capabilities. This section examines the shift in hiring patterns, the "seniors with AI" strategy, and how AI is transforming traditional communication models.

Reducing junior positions

As AI tools take over tasks traditionally performed by entry-level developers, there is a general trend among companies to reduce the number of junior developers they hire. Despite a 47% growth in entry-level software engineering job postings in December 2024 compared to 2023, a significant downward shift has occurred since then[29].

This is a part of a broader trend across the industry, with big tech companies leading the decline. Google and Meta have reduced new graduate hiring by approximately 50% compared to 2021 numbers[30]. New graduates currently represent just 7% of big-tech hires, down 25% from 2023[30, 31].

The figure below shows that 73% of organizations have reduced the number of junior developers over the past two years, highlighting a shift in how organizations perceive junior talent in the AI era.



How has the ratio of junior developers changed in your organization over the past 2 years? Source: LinkedIn Poll

Stack Overflow's annual surveys confirm this trend is widespread. Developers with 0-4 years of professional coding experience dropped from 32.8% of survey respondents to 24.8% between 2024 and 2025[32]. The 2024 spike, followed by the 2025 crash, suggests that companies briefly resumed junior hiring post-pandemic, then stopped as AI capabilities emerged[32].

Full-time employment for all developers also declined from 75.3% to 69.8% between 2024 and 2025, but junior developers are bearing the brunt of the reduction[32].

AI adoption is driving this decline, alongside the rise of Platform Engineering. Platform Engineering creates pressure for smaller, more senior teams who can build and maintain complex systems that support developers. In combination, these factors suggest that junior developers are an unnecessary expense in the short term.

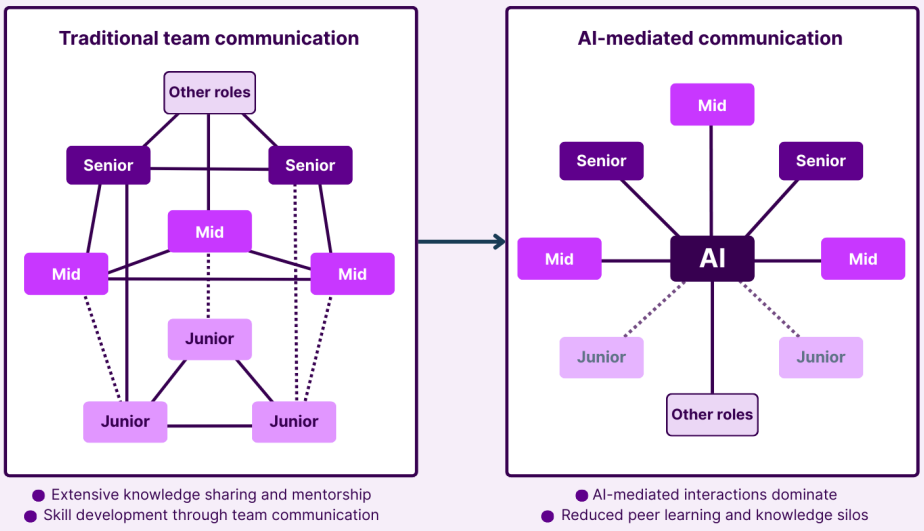
Economic rationale

> Seniors with AI = Seniors + Juniors

Organizations see cutting junior positions as a straightforward financial decision. Senior developer salaries are typically 1.4 to 2.1 times higher than junior developer salaries, plus juniors need a significant investment in training and mentoring[32]. AI tools offer a greater return on investment, currently ranging from $500 to $1,000 per year per developer, driving the "seniors with AI" strategy[4]. The belief underpinning this move is that experienced developers using AI can effectively replace the work of entire teams.

The figure below illustrates the differences between traditional team communication and AI-mediated communication. The traditional model highlights collaboration and knowledge sharing across all developer experience levels and other roles, promoting learning and skill development through direct interactions.

In contrast, the AI-mediated communication model suggests a fundamental shift in interaction patterns. AI tools become the primary interface for problem-solving, thereby reducing the need for communication between team members. This change creates isolated expertise, limiting knowledge transfer, and undermining collaborative learning that juniors need to progress into senior roles.



Traditional communication transitioning into AI-mediated communication

Organizations benefit from a marketplace of experienced developers, but aren't incentivized to train and develop juniors. It's easier to recruit individuals who have already acquired the skills elsewhere. When one organization cuts junior staff to increase its profitability, others will likely follow, creating a chain reaction that results in an entire industry cutting back on the training of new talent exactly when it needs them most.

Phase 3: Senior talent crisis

The current hiring patterns for junior developers presents a significant challenge to the availability of future senior talent. This section explores the perceptions of role-specific vulnerabilities, career transition patterns, and the organizational implications of these workforce dynamics.

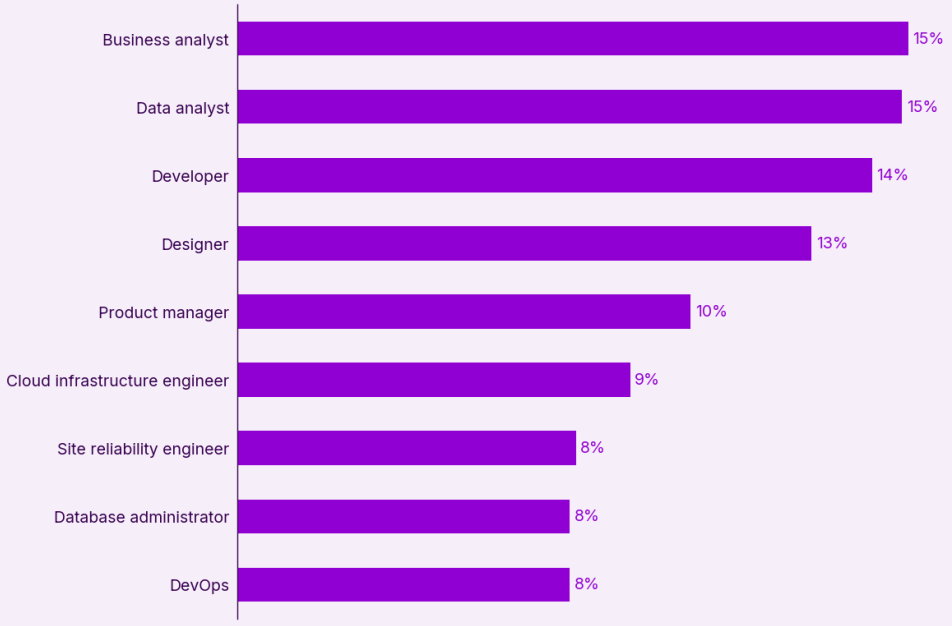
Talent development pipeline

Developer progression generally follows fixed timelines, with 5-7 years of experience and mentoring required to reach a senior level[33]. For specialized roles, 10+ years of experience is needed[33].

There is no shortcut in this process, as expertise requires time and a wide range of varied experiences. Juniors become mid-level when they work independently and understand "why", not just "how." They reach a senior level when others seek their guidance, and they balance technical work with strategic and creative thinking[34]. This progression is fundamental to building capable teams, and while AI can provide support, it can't replace it.

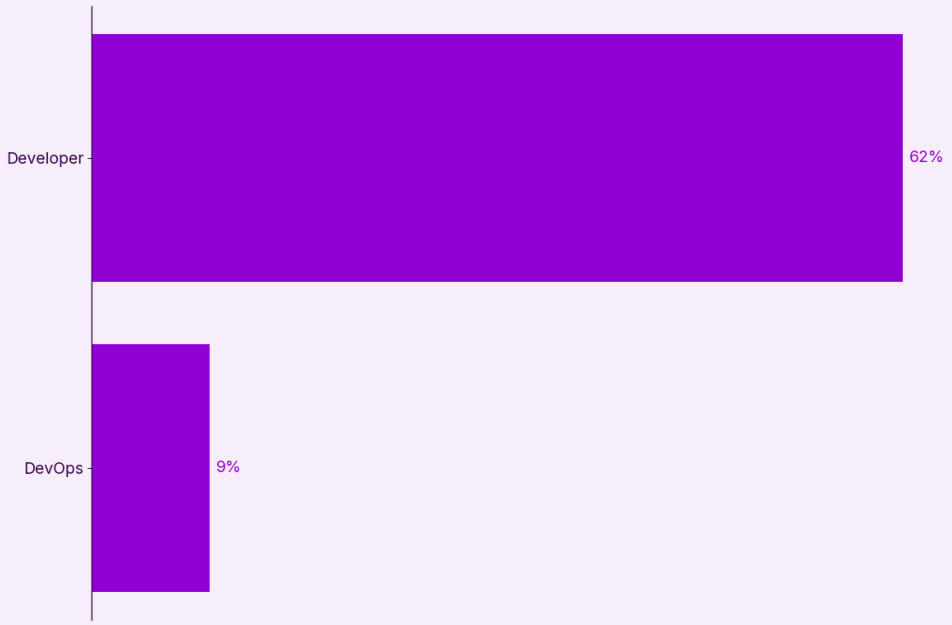
The software industry has historically experienced similar talent pipeline disruptions, with predictable consequences. The dot-com downturn (2000-2002) and the post-pandemic period (2022-2023) led to significant decreases in junior hiring, followed by senior talent shortages and wage inflation 2-3 years later[35].

To understand the perceived impact of AI, we asked respondents which roles they expect to be negatively impacted by AI advancements and integration. In the figure below, we found that *Business analysts* and *Data analysts* are perceived to be most vulnerable to AI adoption (15% each), followed closely by *Developers* (14%) and *Designers* (13%), suggesting that this vulnerability spans across analytical and creative technical roles.



Which roles do you think will be negatively affected by AI in the future? Source: Primary survey data

The most striking finding emerged from role-specific analysis, where 62% of *Developers* and 9% of *DevOps* professionals reported concern for their own careers, with no other professional group showing this level of self-awareness.



Self-concern of negative AI impact on job role. Source: Primary survey data

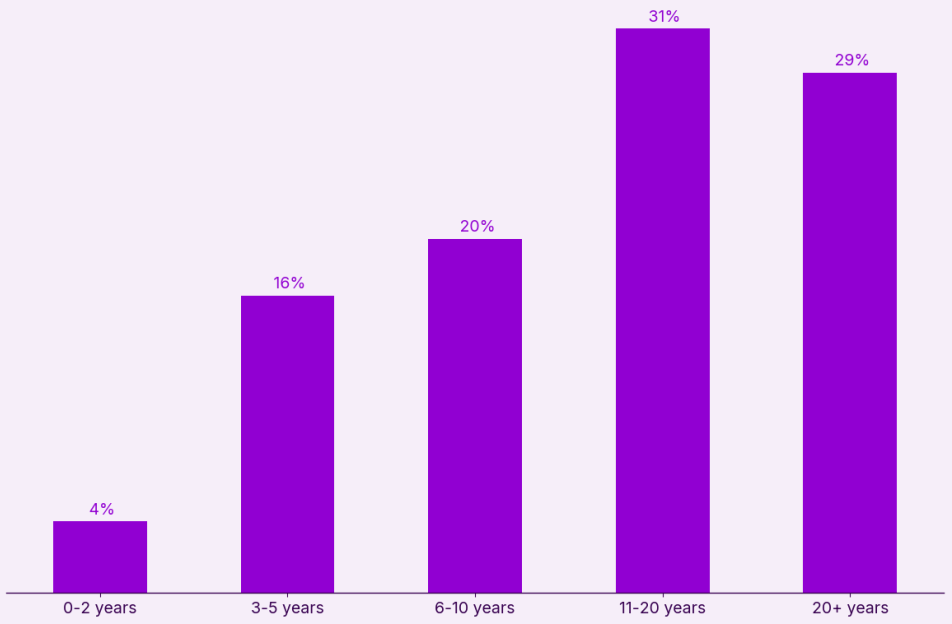
This is particularly interesting as these professionals, who are closest to building and deploying AI systems, recognize their own risk, while other roles may be unaware or in denial about AI's potential impact. Given that those best-positioned to evaluate AI's potential acknowledge the risks they face, it raises questions about how disruptive AI will be to roles more separated from the technology.

Effects and consequences

Cutting junior developers may trigger a chain of organizational problems. This may include outdated documentation as seniors lack the time to update it, and a reduction in innovation, as AI provides conventional solutions rather than fresh perspectives.

Senior burnout may accelerate as they handle both strategic and routine tasks without options for delegating to junior developers. With all decisions in fewer hands, technical debt accumulates, and the initial productivity gains are reversed as these actions compound.

The figure below shows the distribution of developers by years of work experience, with the 11-20 years (31%) and 20+ years (29%) groups together representing 60% of all respondents.



Developers in industry across experience levels (years of coding). Source: 2025 Stack Overflow survey

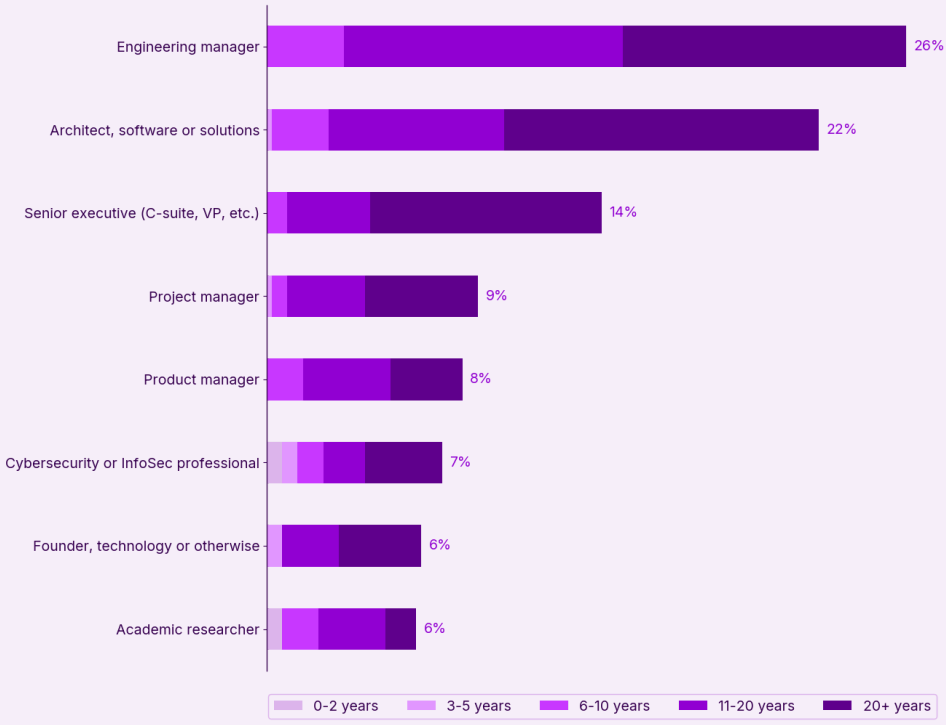
As senior developers naturally progress into management roles, the technical talent pool shrinks further. The timeline issue is fundamental here, as it takes 5-10 years to mature into a senior developer[33]. When organizations recognize the projected shortage (for 2027-2030), they will compete for the same limited talent pool, driving unsustainable salary inflation.

When asked about role transitions, 46% of developers have not considered changing roles, while 44% are "somewhat" or "strongly" considering transitions. Notably, those contemplating a role change cluster in the mid-career brackets (3-5, 6-10, and 11-20 years of experience), which are those who traditionally bridge the gap between early career learning and senior leadership.

	Percentage of developers
Not considered	46%
Somewhat considered	29%
Strongly considered	15%
Transitioned voluntarily	9%
Transitioned involuntarily	2%

Have you considered a career change or transitioned into a new role in the past year? Source: 2025 Stack Overflow survey

This mid-career uncertainty translates into actual career transitions from technical roles. The figure below illustrates the most common career transitions for developers, with 26% of former developers transitioning into *Engineering manager* roles, 23% into *Architect* roles, and 14% into *Senior executive* roles.



Top 8 roles that developers transition into. Source: 2025 Stack Overflow survey

These transitions are natural, but they draw experienced developers away from hands-on work, creating vacancies that accelerate the loss of foundational skills and institutional knowledge. Without a pipeline to fill them, hiring alone can't solve this gap.

The knowledge gap risk

AI excels at foundation-level tasks and, while this does accelerate output, it bypasses critical learning processes[36]. Studies have found that debugging experiences and error resolution contribute to skill development and pattern recognition capabilities[37, 38]. Without these, progression from foundational to expert knowledge will be disrupted.

Developers using AI to generate entire modules often overlook understanding why solutions work, when they fail, and which trade-offs matter in specific contexts[38]. Habits may form where developers generate code without a thorough understanding of what they have created, and when they encounter issues, they apply AI-generated solutions without identifying the root causes[38].

The decision-making required for software development is based on context, which AI cannot yet replicate (like balancing performance with maintaining systems, speed against technical debt, best practices against pragmatic constraints). These decisions depend on the project context, team capabilities, and business knowledge.

Those who rely on AI may not develop a comprehensive understanding of details relevant to their specific uses[39]. Organizations expecting to develop senior-level expertise within 3-5 years may find themselves with less qualified and capable candidates.

The AI/ML Engineering has emerged as a new career trajectory as the industry adapts to AI, with increased demand for developers who bridge the gap between traditional programming skills and AI-assisted development. Junior developers can evolve into these positions by combining conventional skills with AI expertise[40].

The path forward

The challenge facing the software industry is not whether to adopt AI, but how to integrate it to enhance the human experience while preserving talent pipelines. This section outlines recommendations at the industry, organizational, and individual levels.

Industry and organizations

Focusing on adapting to the evolving workforce and implementing internal process can include:

- Establishing AI-assisted training standards/programs with mentorship for evaluating AI generated output.
- Redefining role requirements and expectations for changing workforce demands (such as having juniors work alongside AI, new roles emerge).
- Developing internal best practices for AI implementation.
- Strengthening underlying infrastructure with automation through Continuous Delivery to leverage AI's full potential.

Individual developers

Focusing on what developers can do to stay relevant can include:

- Using AI strategically to automate routine tasks, while developing independent problem-solving skills.
- Developing fundamental and irreplaceable skills (like system design, performance optimization, complex debugging, business knowledge, and communication).
- Prioritizing continuous learning and seeking mentorship opportunities that provide exposure to senior-level decision-making.

We recognize that these recommendations are not new, and many organizations will already have adopted one or more of these strategies. Our aim is to offer a framework/perspectives for measured AI integration.

Our take

As we collectively stumble in our attempts to bring a new generation of tools into our work, things can feel quite bleak. Our early achievements disappear when we scale up from experimental tasks to real work. This shouldn't dishearten us as it's part of the learning process.

To make progress, we need to pay attention to a finding that has come up time and again in research and practice: this idea around flow and bottlenecks. We can draw on ideas from the theory of constraints and value stream management to help with this, but, in software development in particular, we can draw on a method built with this in mind: Continuous Delivery.

The mechanisms baked into Continuous Delivery are designed to lay a smooth, safe pathway for changes, because, even before AI, developers weren't the bottleneck; the deployment pipeline was.

Teams with high levels of automation will be better able to convert individual and team productivity gains into software and organizational outcomes. At the same time, those burdened with manual stages and quality gates will lose all developer productivity gain in the queues and reviews that follow.

The AI Pulse report highlights how a common bottleneck comes from code review. Once you solve this, the constraint moves elsewhere, and you'll have a new problem to solve. Continuous Delivery is the culmination of practices you'll need to tackle each bottleneck you'll encounter as you seek to improve the end-to-end process of delivering software.

Conclusion

The software development industry is currently facing critical directional decisions. AI is driving productivity gains for individuals and organizations, however, current implementation patterns focus on code generation rather than addressing review and validation processes.

This approach indicates that the challenge lies in how organizations structure their development workflows rather than in AI capabilities. Continuous Delivery provides a framework for achieving this balance by automating repetitive tasks while preserving the context-based work that encourages developer expertise.

Reducing junior developer positions across the industry and in organizations, may lead to senior talent shortages in the next 3-5 years. This could also be exacerbated by senior developer burnout, unsustainable salary demands, limited scaling capacity, and architectural fragility. Organizations with firm plans to upskill junior developers alongside AI adoption are better positioned for long-term talent availability and development capacity.

Using AI to enhance developer capabilities rather than replace skill development and knowledge growth, is a sustainable path forward. By capturing efficiency gains while preserving learning opportunities, organizations can achieve immediate efficiency and a robust talent pipeline for the future.

Contributors

Charlotte Fleming

Researcher

Developer relations researcher, neuroscientist, and research lead for The AI Pulse report.

Steve Fenton

Principal DevEx Researcher

A long-time software delivery practitioner, author, and researcher.

Ivan Krnic

Director of Engineering (**CROZ**)

Sociotechnical architect, author of BizTech Evolution, and host of the 0800-DEVOPS podcast.

David Stenglein

Strategic Advisor, Platform Consultant (**Missing Mass LLC**)

Platform engineering consultant helping enterprises build high-performing teams and platforms.

John Bristowe

Principal Developer Advocate

Software engineer, DevOps practitioner, host of Deploy on Friday.

Methodology

We employed a mixed-methods approach, combining primary data with an analysis of existing industry research to understand the current impact of AI adoption in the industry, with a specific focus on the developer talent pipeline.

We included analysis of data from established industry surveys (**AI at Work report**, **DORA Impact of Generative AI in Software Development report**, **DORA State of AI-assisted Software Development 2025 report**, **Stack Overflow 2025 Developer Survey**, **2024 State of Developer Ecosystem report**) to research baseline patterns of AI adoption, capabilities, use across experience levels, and role transitions[2, 5, 6, 20 & 21].

Industry data

We analyzed data from the **Stack Overflow 2025 Developer Survey**, specifically examining AI capabilities, AI adoption, and how these are perceived differently across experience levels, categorized by years of coding experience. We also examined the number of developers considering transitioning into other roles, and the most common roles developers move into by years of coding experience.

Primary data

We designed our primary survey to validate emerging patterns and explore aspects of current perceptions of the roles in which AI will have a negative impact on. This survey included a range of open-text, multiple-choice, and Likert-type questions, with an "other" option allowing individuals to add entries not included in the options provided. The survey was distributed through **Typeform** and then shared via email, social media, and a research panel used in previous studies over a period of 2 months.

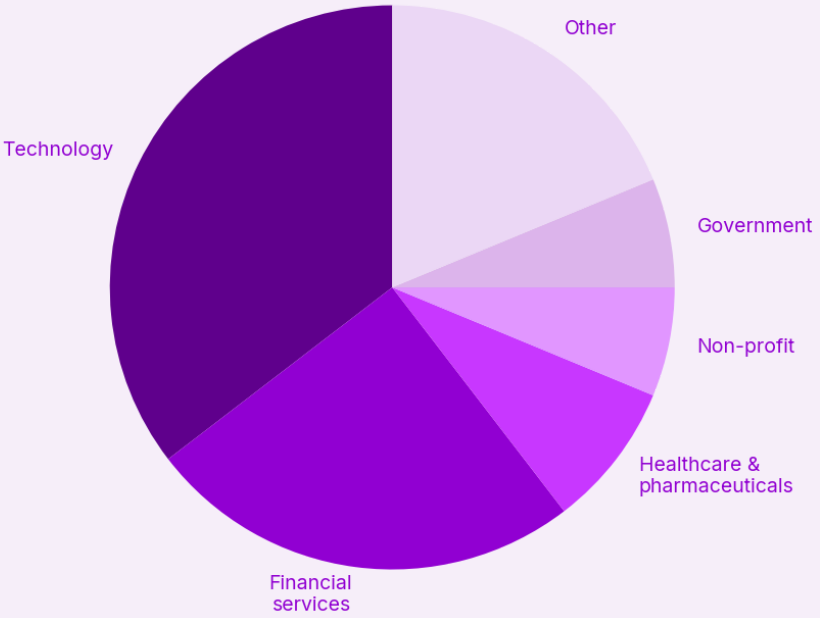
LinkedIn polls were conducted targeting technology professionals, focusing on junior developer hiring trends and AI capabilities, to gather a real-time industry perspectives. This provided quick validation of patterns observed in primary and industry survey data.

Analysis

We used Microsoft Excel and Python for analysis of both primary and industry data. For the role-specific analysis, the roles perceived to be impacted by AI integration were found, then matched this list against their own role to determine whether they had selected their own role. The percentage of “self-concerned” respondents was then calculated. The findings in the industry data, along with our own, serve as a baseline for further exploration to track these patterns in future studies.

Firmographics

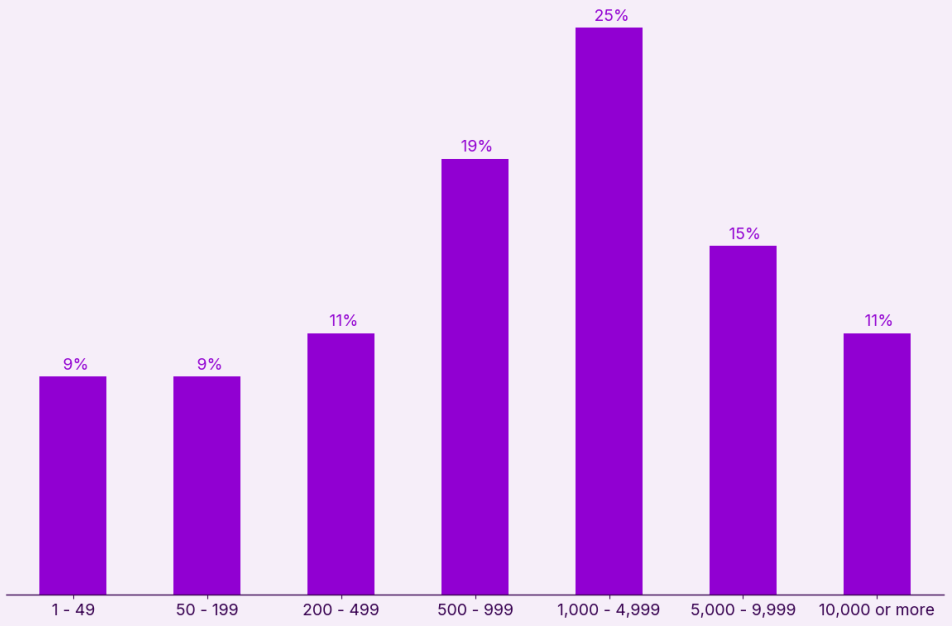
Industries



Industries. Source: Primary survey data

Overall, more than 10 industries were represented in our primary survey, with the top 2 industries accounting for over half of the responses. The top 5 industries were Technology (35%), Financial services (25%), Healthcare & Pharmaceuticals (8%), Government (6%), and Non-profit (6%).

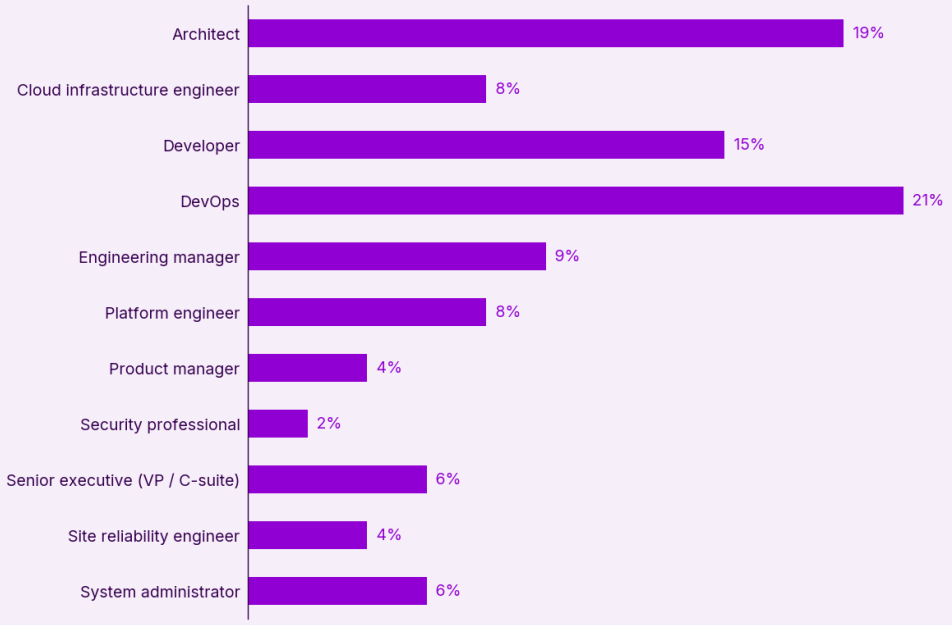
Organization size



Organization size. Source: Primary survey data

Responses were received from organizations of all sizes. Comparable to industry research, we found that organizations with 1,000 to 4,999 employees are the largest representative group in this study.

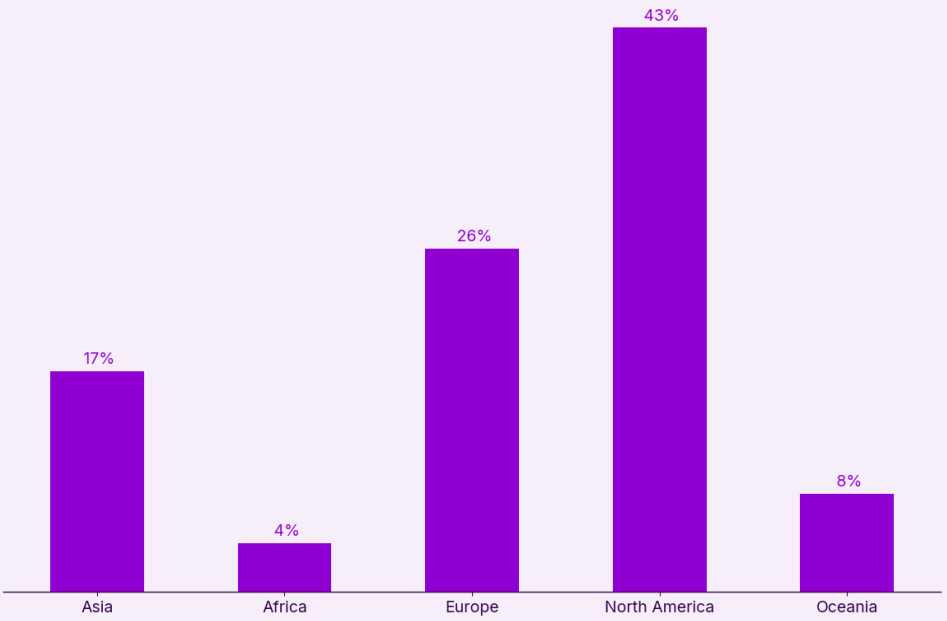
Job roles



Job roles. Source: Primary survey data

Over 60% of responses came from just three roles: *Architect* (19%), *DevOps* (21%), and *Developer* (15%). Two of these roles (*DevOps* and *Developer*) were of particular interest for this study.

Location



Job roles. Source: Primary survey data

The survey reached people in almost every continent, with most responses coming from North America (43%), followed by Europe (26%), and Asia (17%).

Sponsors

References

1. Kassa, B.Y., Worku, E.K. (2025). The impact of artificial intelligence on organizational performance: The mediating role of employee productivity. Journal of Open Innovation: Technology, Market, and Complexity, volume 1, issue 1**https://doi.org/10.1016/j.joitmc.2025.100474**
2. **https://www.globalization-partners.com/resources/2025-ai-at-work-report/?__hstc=207525708.5bf10ccafab06a6424be8ed7c68710f3.1761013132000.1761013132000.1761277604949.2&__hssc=207525708.1.1761277604949&__hsfp=3753239694&submissionGuid=ef051503-23ec-47e3-bd02-32308317e769**
3. **https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/superagency-in-the-workplace-empowering-people-to-unlock-ai's-full-potential-at-work**
4. **http://getdx.com/podcast/planning-2026-ai-tooling-budget/**
5. **https://dora.dev/ai/gen-ai-report/report/**
6. **https://dora.dev/research/2025/dora-report/**
7. **https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/**
8. **https://dora.dev/research/2024/dora-report/**
9. Orouji, M. (2016). Theory of constraints: A state-of-the-art review. *Accounting 2*, (45-52). https://**10.5267/j.ac.2015.12.004**
10. **https://mikecarruego.medium.com/the-theory-of-constraints-in-software-development-7e37cb0911db**
11. **https://www.linkedin.com/pulse/how-long-does-take-build-software-realistic-timelines-d4vkc/**
12. **https://ijcsm.researchcommons.org/cgi/viewcontent.cgi?article=1114&context=ijcsm**

13. **https://www.sciencedirect.com/science/article/pii/S0950584923002471#b8**
14. **https://axify.io/blog/devops-value-stream**
15. **https://ijcsm.researchcommons.org/cgi/viewcontent.cgi?article=1114&context=ijcsm**
16. Kumar, S., Goel, D., Zimmerman, T., Houck, B., Ashok, B., Bansal, C., (2025). Time Warp: The Gap Between Developers' Ideal vs. Actual Workweeks in an AI-Driven Era, *Software Engineering in Practice*. **https://doi.org/10.48550/arXiv.2502.15287**.
17. **https://mitsloan.mit.edu/ideas-made-to-matter/whats-your-companys-ai-maturity-level**
18. **https://www.bcg.com/publications/2025/are-you-generating-value-from-ai-the-widening-gap**
19. **https://continuousdelivery.com/**
20. **https://survey.stackoverflow.co/2025/**
21. **https://www.jetbrains.com/lp/devecosystem-2024/**
22. **https://survey.stackoverflow.co/2024/**
23. **https://arxiv.org/html/2509.22420v1**
24. **https://arxiv.org/abs/2407.20792**
25. **https://www.techtarget.com/searchenterpriseai/feature/Insights-on-generative-AI-for-automation-vs-augmentation**
26. **https://medium.com/@sanjeevanibhandari3/why-juniors-are-learning-faster-than-seniors-the-great-ai-pair-programming-shift-1f68fbacc6d7**
27. **https://www.ibm.com/think/insights/ai-improving-developer-experience**
28. **https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/**
29. **https://www.hackreactor.com/resources/report-finds-47-growth-in-entry-level-software-engineer-jobs/**
30. **https://codeconductor.ai/blog/future-of-junior-developers-ai/#:~:text=This%20slowdown%20has%20hit%20juniors,enable%20seniors%20to%20handle%20more**

31. **https://www.linkedin.com/pulse/software-developer-labor-demand-salary-trends-2025-julius-gromyko-o5vhf/**
32. **https://survey.stackoverflow.co/**
33. **https://www.index.dev/blog/how-to-become-a-senior-software-engineer**
34. **https://www.notonlycode.org/nobody-hires-juniors/**
35. **https://medium.com/career-programming/not-hiring-junior-developers-isnt-a-mistake-it-s-the-tragedy-of-the-commons-e053c9d5e6cb**
36. **https://hackernoon.com/software-engineer-qualification-levels-junior-middle-and-senior-f2229591df1c**
37. Ahmadzadeh, M., Elliman, D., Higgins, C., (2015). The Impact of Improving Debugging Skill on Programming Ability, *Innovation in teaching and Learning in Information and Computer Science*. (72-87). **https://doi.org/10.11120/ital.2007.06040072**.
38. **https://sidecar.ai/blog/the-pattern-matching-paradox-why-apples-ai-critique-misses-what-matters-for-associations**
39. **https://www.altexsoft.com/blog/software-engineer-qualification-levels-junior-middle-and-senior/**
40. **https://vettio.com/blog/rise-of-ai-engineer/**

Octopus Deploy

Level 4, 199 Grey St
South Brisbane, QLD 4101,
Australia

sales@octopus.com
+1 512-823-0256
octopus.com