

2024 Planning Guide for Software Development

Published 4 October 2023 - ID G00796376 - 31 min read

By Analyst(s): Peter Hyde, Matt Brasier, Tyler Bray, Danny Brian, James Coll, Brad Dayley, Bill Holz, Gary Olliffe, Sushant Singhal, Nick Wilcox

Initiatives: [Software Development for Technical Professionals](#); [Adopt Modern Architectures and Technologies](#); [Application Architecture and Integration for Technical Professionals](#); [Build a World-Class Software Engineering Organization](#); [Build and Deliver New Digital Products/Experiences to Drive Business Results](#)

Software development powers business innovation and growth with product differentiation fueled by build-over-buy decisions. Technical professionals must rethink front-end frameworks and adopt generative AI, platform engineering and continuous learning to achieve success amid rapid technology change.

Overview

Key Findings

- Generative AI (GenAI) presents a significant opportunity to automate routine aspects of software delivery, but improving developer effectiveness requires mitigating AI risks and targeting appropriate use cases.
- The increasing complexity and pace of product development creates a strong need for platform engineering to reduce team cognitive load, improve the developer experience and boost engineering transparency.
- The challenge of managing front-end framework dependencies makes it difficult for development teams to manage and enhance the functionality of rich web applications, especially where development teams neglect modern browser fundamentals.
- Upskilling and reskilling have become essential for software engineers struggling with high business demand, dynamic market conditions and accelerating technology change.

Recommendations

- Form or join a community of practice for AI-augmented development to share experiences and techniques for when and how to get the most out of GenAI.
- Collaborate with engineering management to establish compelling internal developer platforms, offering clear pathways for product and service teams to deliver exceptional business value.
- Minimize web front-end framework dependencies as much as possible. This requires that developers learn front-end development basics, including modern browser support for ES6 features and web components.
- Create a personal learning plan to identify development opportunities, embed learning in your work and resolve critical skill gaps to support your team and organization.

Software Development Trends

The role of a software engineer has never been more difficult. The explosion in complexity and innovation in underlying technologies, libraries and practices creates a demanding landscape for software engineers to traverse. Increased business expectations of resilient, secure and feature-rich customer experiences delivered on-demand compound the challenge and make continuous upskilling and regular reskilling a necessity.

“Complexity kills. It sucks the life out of developers, it makes products difficult to plan, build and test, it introduces security challenges, and it causes end-user and administrator frustration.”

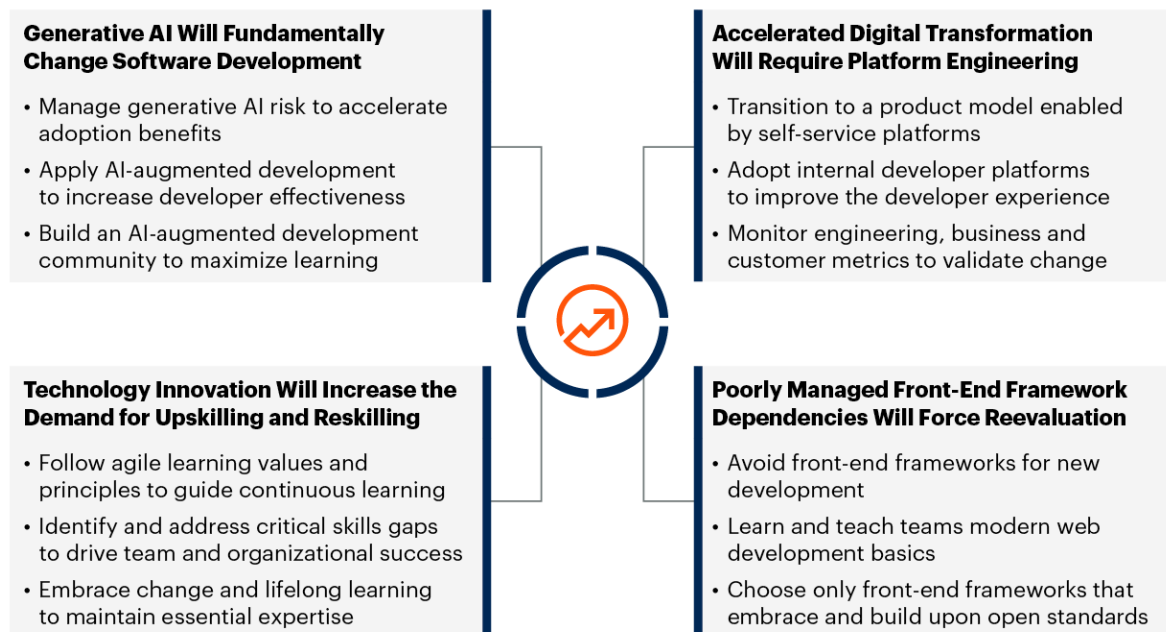
— Ray Ozzie, Microsoft Chief Technical Officer, 2005-2006¹

As a software engineer, you will need to evolve your technology skills and actively engage in change at your organization to manage complexity and increase value creation. Take advantage of the opportunities offered by AI-augmented development to improve the efficiency, quality and speed of application delivery. Adopt platform engineering to reduce the cognitive load caused by the increasing technical complexity of underlying systems. Minimize front-end framework dependencies and establish clear guidelines for their use to streamline the development process, ensure consistency across projects, improve code maintainability, and enhance the end-user experience.

Figure 1 shows Gartner's four key trends for software development in 2024, along with their corresponding planning considerations. These trends are based on our interactions with clients, their opportunities and challenges, and an analysis of the technology markets and ecosystems that support and influence them.

Figure 1: 2024 Key Trends in Software Development

2024 Key Trends in Software Development



Source: Gartner
796376_C

Gartner's four key trends for software development in 2024 are:

- [Generative AI Will Fundamentally Change Software Development](#)
- [Accelerated Digital Transformation Will Require Platform Engineering](#)

- Poorly Managed Front-End Dependencies Will Force a Reevaluation of Frameworks
- Technology Innovation Will Increase the Demand for Upskilling and Reskilling

Generative AI Will Fundamentally Change Software Development


















GenAI is the fastest-growing inquiry topic that Gartner has ever seen, and the opportunities to apply GenAI in software development are significant. However, this emerging technology has technical, financial, organizational and legal risks that must be managed to ensure successful adoption. Generated content must be carefully reviewed before use, as it can suffer from inaccuracies (often referred to as hallucinations) and can also result in artifacts (including code) that infringe on other organizations' or individuals' intellectual property.

Development teams are keen to adopt new technologies that can improve their workflow and minimize some of the less enjoyable aspects of their work. However, they are nervous about doing so under unrealistic expectations of order-of-magnitude productivity improvements or expected team size reductions.

Successfully incorporating AI-augmented development tools into your software development workflow requires identifying where the tools can reduce effort, while mitigating the associated risks (see Figure 2).

Figure 2: Balancing Generative AI Benefits and Risks

Balancing Generative AI Benefits and Risks

Identify Areas Where Generative AI Can Help	Mitigate and Avoid the Risks and Pitfalls Associated With Using Generative AI	Consider Alternative Approaches
 Generating Code	 Managing Intellectual Property	 Low-Code/No-Code Application Platforms
 Generating Documentation	 Moving Rather Than Saving Effort	 Development Outsourcing
 AI-Augmented Testing	 Doubling Down on Cognitive Bias	 Pair Programming
 Debugging Code	 Loss of Skills	 Modern IDEs and Development Tools
 Generating Build and Deploy Artifacts	 Believing The Model Knows What It Is Doing	
 Learning New Languages and Frameworks	 Speed of Change	
 Integration		

Source: Gartner
793995_C

However, it is important to recognize that the AI-augmented software development tool market is experiencing a dynamic and rapid expansion, accompanied by significant technology changes. While certain risks, like hallucinations, are inherent to the technology and cannot be fully eliminated, other risks, such as IP theft, are being targeted and reduced.

Over the next few years, the initial hype surrounding GenAI will subside and developers will better understand how and when to leverage assistance from these tools, resulting in notable improvements in developer effectiveness.

AI-augmented development tools are here to stay and in two years, using them will be as natural as using any other code-generation technology.

Software engineers should follow these planning considerations as they adopt GenAI tools:

- [Manage generative AI risk to accelerate adoption benefits](#)
- [Apply AI-augmented development to increase developer effectiveness](#)
- [Build an AI-augmented development community to maximize learning](#)

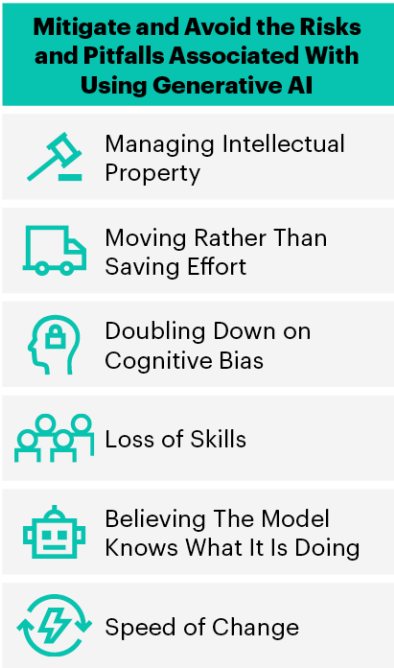
Planning Considerations

Manage Generative AI Risk to Accelerate Adoption Benefits

Using GenAI in your development process presents multiple risks and pitfalls that must be mitigated or avoided (see Figure 3). The risks and pitfalls are not all technical in nature, and resolving them will require a coordinated organizational effort involving software engineers, software engineering leadership, security and legal representatives.

Figure 3: Risks and Pitfalls of Generative AI In Development

Risks and Pitfalls of Generative AI In Development



Source: Gartner
793995_C

The key risks and pitfalls of GenAI in development are:

- **Managing intellectual property** — GenAI tools can both leak your intellectual property and generate artifacts that infringe the intellectual property rights of others.
- **Moving rather than saving effort** — Obtaining good results requires carefully crafted prompts and detailed examination of the output. Sometimes this takes longer than just creating the artifact yourself.
- **Doubling down on cognitive bias** — Having a tool assist you in getting your approach to work can reinforce your bias that your approach is correct. AI-augmented development tools are often compared to pair programmers, but a good pair programmer will challenge you if they think your approach is wrong.
- **Loss of skills** — If simple tasks are delegated to AI tools, it removes them as learning and practice opportunities, resulting in skills never being learned or atrophying through lack of use.
- **Believing the model knows what it's doing** — Automation bias is the tendency for humans to trust recommendations from automated systems even in the face of contradictory information from other sources. GenAI tools have high error rates (including creating vulnerable code) and automation bias makes identifying and correcting those errors harder.
- **Speed of change** — While the overall technology is moving quickly, many of the underlying models used were trained several years ago and cannot work with the latest language and framework versions.

Software engineers must develop an understanding of the benefits and limitations of GenAI to manage these risks and pitfalls. This skill will help them to assess when an AI-based tool saves time or if creating prompts and validating output will take longer than doing the task manually.

Read [Assessing How Generative AI Can Improve Developer Experience](#) for more guidance on managing these risks.

Apply AI-Augmented Development to Increase Developer Effectiveness

The question that everyone wants to know the answer to is: “How much of a productivity gain can I expect from GenAI tools?” The truth is that it will take a while before we have enough data to understand the impact. Many articles claim productivity benefits based on the reduction in time taken to write code to solve common interview-level tasks. However, writing code is only part of what developers do, and the complexity of the code they write will vary on a day-to-day basis.

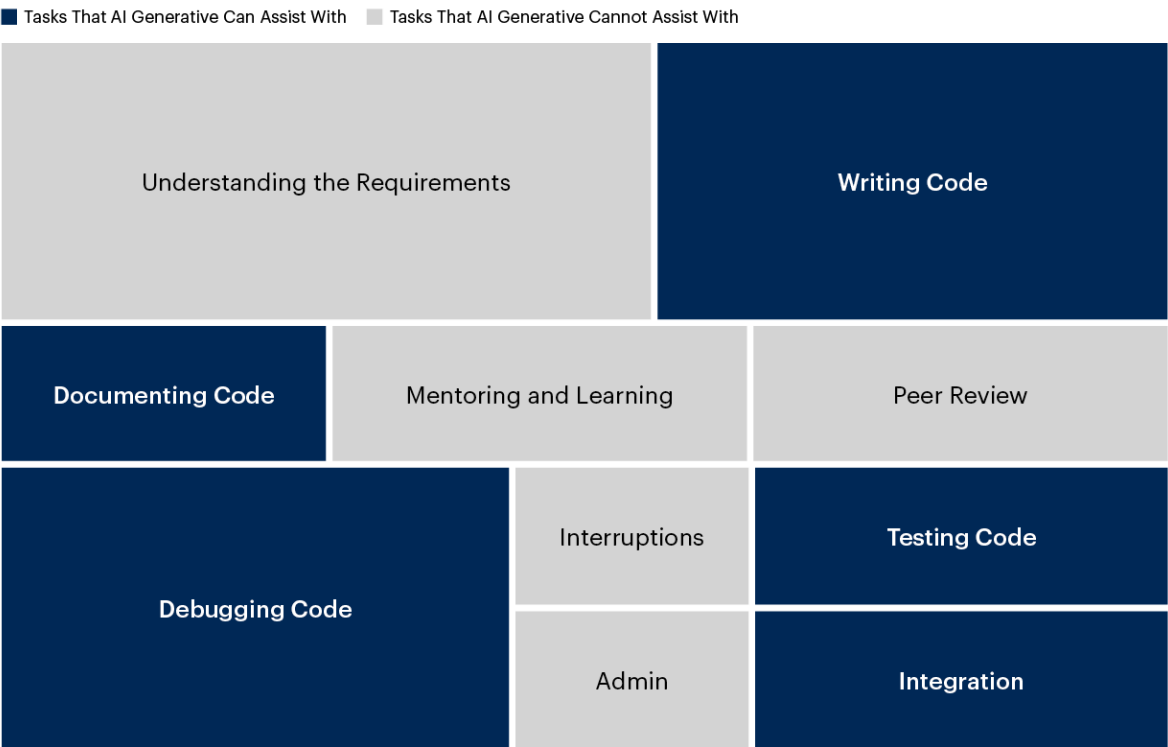
There are many problems with implementing GenAI with a defined productivity goal in mind, not least that reductions in the time taken to write code will not necessarily translate to an improvement in delivery time. The time a developer spends on a task is often influenced by completion estimates or deadlines, and product teams can decide to incur technical debt to achieve development targets.

Suppose a developer saves 20 minutes on a coding task. They may spend that extra time writing tests, refining requirements, reducing technical debt or any number of other tasks that developers must perform. The result is an improvement in quality, not a reduction in delivery time.

Figure 4 illustrates the types of tasks developers perform as part of their role, with the sizes indicating approximate proportions.

Figure 4: What Tasks Do Developers Perform?

What Tasks Do Developers Perform?
Illustrative



Source: Gartner (based on data from multiple Generative AI tool vendors)
Note: Sizes are very approximately proportional to the amount of time before using Generative AI.
796376_C

Reducing technical debt is unlikely to show up as an immediate boost in “productivity,” yet it can still result in a higher quality product (and therefore less time spent on rework) many months in the future.

Several aspects are already becoming clear:

- GenAI tools can provide a suitable environment for learning unfamiliar frameworks or languages, but when used in this way, the resulting code should not be considered production quality.
- GenAI tools trained on good quality source data can produce production quality code, but doing so requires careful prompting from an experienced developer who knows what “production quality” requires in a given framework or language.

- AI-augmented development tools are better at some tasks (such as generating boilerplate code, test harnesses, documentation or implementing standard algorithms) than others (such as creating proprietary algorithms or implementing unique business logic).
- Experienced developers are better able to judge when a GenAI tool will save them time in the short term, or when understanding and validating the GenAI output and managing any technical debt will cost time in the longer term.
- Tools that allow developers to spend more time on tasks they enjoy (such as solving complex business problems) and less time on tasks they do not enjoy (writing documentation, testing etc.) can improve the developer experience.

Rather than focusing specifically on development productivity, treat AI-augmented development tools as a way of improving developer experience and effectiveness. If applied well, these tools will increase the effectiveness of your software development resulting in improved productivity, quality or cost depending on how and when the tools are applied.

Read [Prompt Engineering With Enterprise Information for LLMs and GenAI](#) for more information on the effective use of GenAI.

Build an AI-Augmented Development Community to Maximize Learning

There are three significant challenges that developers face when building the skills that they need to effectively use AI-augmented development tools:

- The technology and user experience are changing rapidly, making it hard to select the best product for your needs.
- The scenarios where AI-augmented development tools are useful can vary significantly between organizations.
- Widespread adoption of these tools is in progress, but they are still in their early stages of development and not yet mature. Best practices are not yet established for their use.

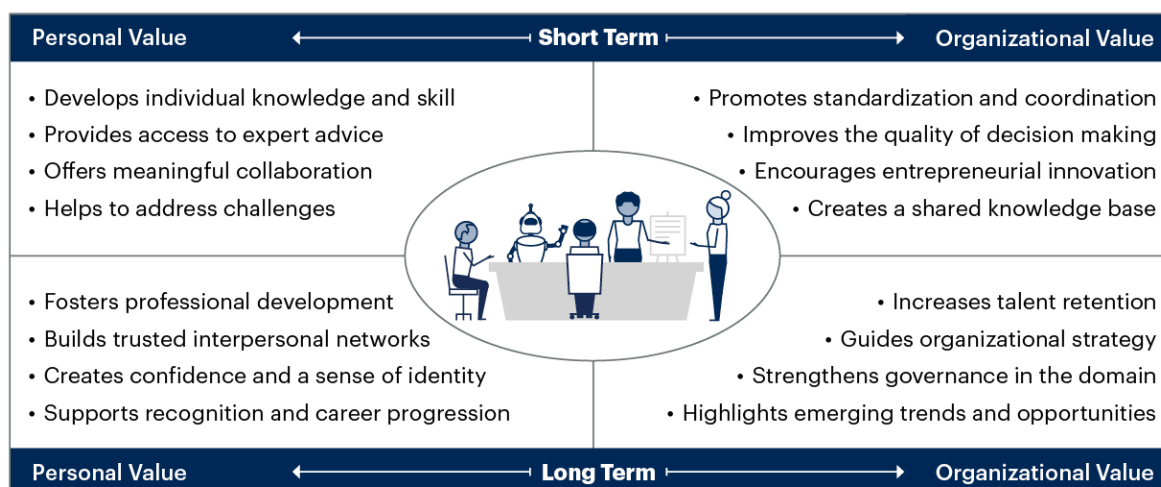
While there are some universal best practices on how to use GenAI tools, many aspects vary from organization to organization. For example, knowing the best keywords to use to get a model to generate secure or high-performance code depends on the model, language and framework versions you are using.

The three factors above mean that learning by sharing experiences with peers within your organization is currently the best way to build AI-augmented development skills. An AI-augmented development community of practice is an excellent way to share hands-on experiences and define best practices. Communities of practice focus on social sharing of information that is of value to practitioners, rather than top-down or more academic approaches to learning. Create an AI-augmented development community of practice (see Figure 5) that focuses on identifying and sharing information on the following topics:

- How to choose the right tools to match the specific development task.
- How to leverage AI-augmented development to create the most benefit and the least disruption.
- Which interaction styles (such as chat and prompt) work best for which tasks?
- What is the best sequence of prompts and keywords for specific desired outcomes (such as security, thread safety, performance) in code?
- How to address issues encountered when using AI-augmented development.
- What benefits have you observed, and how did they compare to those you expected?
- When does prompting a model and validating its output take longer than just creating the code, and what metrics can be used to identify similar tasks?

Figure 5: Benefits of a Community of Practice

Benefits of a Community of Practice



Source: Gartner
796376_C

Read [Community of Practice Essentials](#) to create compelling, engaging and thriving communities that support AI-augmented development.

Accelerated Digital Transformation Will Require Platform Engineering

[Back to Top](#)

Organizations are accelerating their digital transformation initiatives due to increased business demand, dynamic market conditions and disruptive technology change. Platform engineering supports software engineers by providing self-service internal developer platforms for software delivery and life cycle management. Platform engineering aims to optimize the developer experience and accelerate the creation of customer value.

Business agility requires a customer-centric approach with empowered and autonomous teams focused on delivering high-quality products and services. Common platform services help product and service teams to reduce their team cognitive load and increase the delivery of business and customer value.

Platforms must start as simple as possible (thinnest viable platforms) to offer a compelling set of valuable services to address the needs of product and service teams. Platform engineering helps improve the developer experience by reducing the frustration caused by the increased complexity of distributed, continuously deployed and cloud-native applications.

Software engineers should follow these planning considerations as they implement platform engineering:

- [Transition to a Product Model Enabled by Self-Service Platforms](#)
- [Adopt Internal Developer Platforms to Improve the Developer Experience](#)
- [Monitor Engineering, Business and Customer Metrics to Validate Change](#)

Planning Considerations

Transition to a Product Model Enabled by Self-Service Platforms

Collaborate with your engineering leadership to form stream-aligned teams accountable for a single valuable product or service. Reduce the cognitive load for these product and service teams by extracting commonly used supporting services into self-service platforms. Accelerate the flow of business change by holding workshops to identify capability gaps and forming enabling teams to address them.

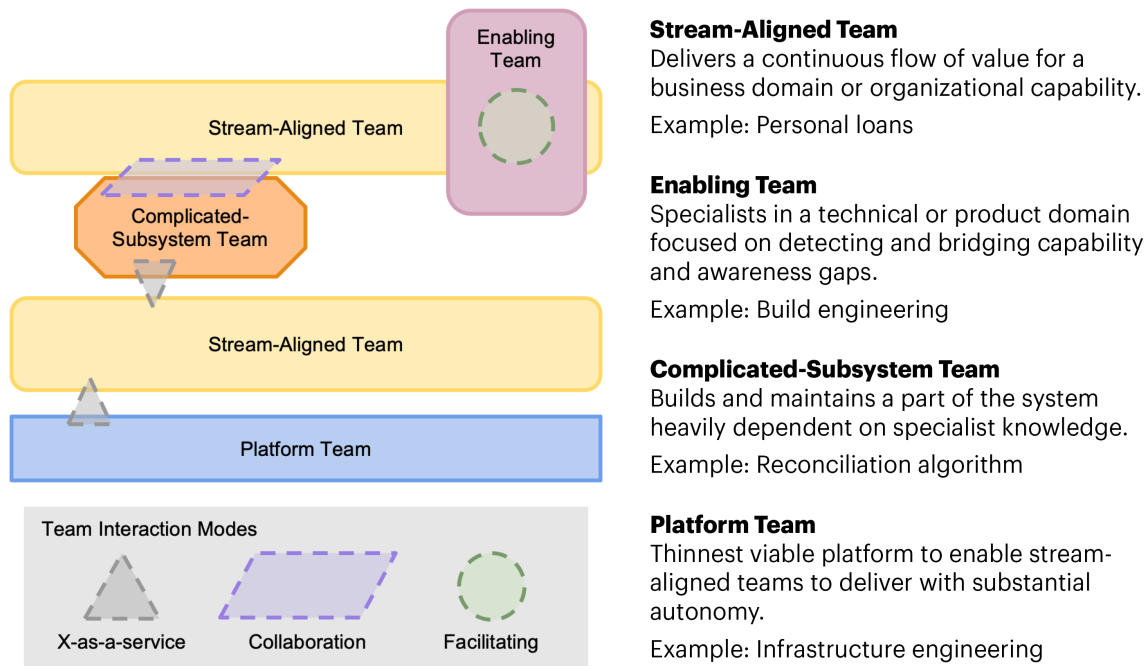
“Great platform teams can tell a story about what they have built, what they are building, and why these products make the overall engineering team more effective.”

— Camille Fournier, Head of Engineering and Architecture CIB, JPMorgan Chase.²

The Team Topologies approach provides software delivery groups with a framework for developing an organizational blueprint that enables business growth and improves the delivery of business value (see Figure 6). Use the shared language of Team Topologies to restructure teams, realign communication paths and optimize the effectiveness and efficiency of products and service delivery.

Figure 6: Organizational Design With Team Topologies

Organizational Design With Team Topologies



Source: Adapted From Team Topologies by Matthew Skelton and Manuel Pais. Used with permission.

772777_C

Read [Organize for Agility With Team Topologies](#) for more information on organizational design for software engineering and [Use Platform Engineering to Scale and Accelerate DevOps Adoption](#) to treat platforms as compelling products with dedicated platform owners.

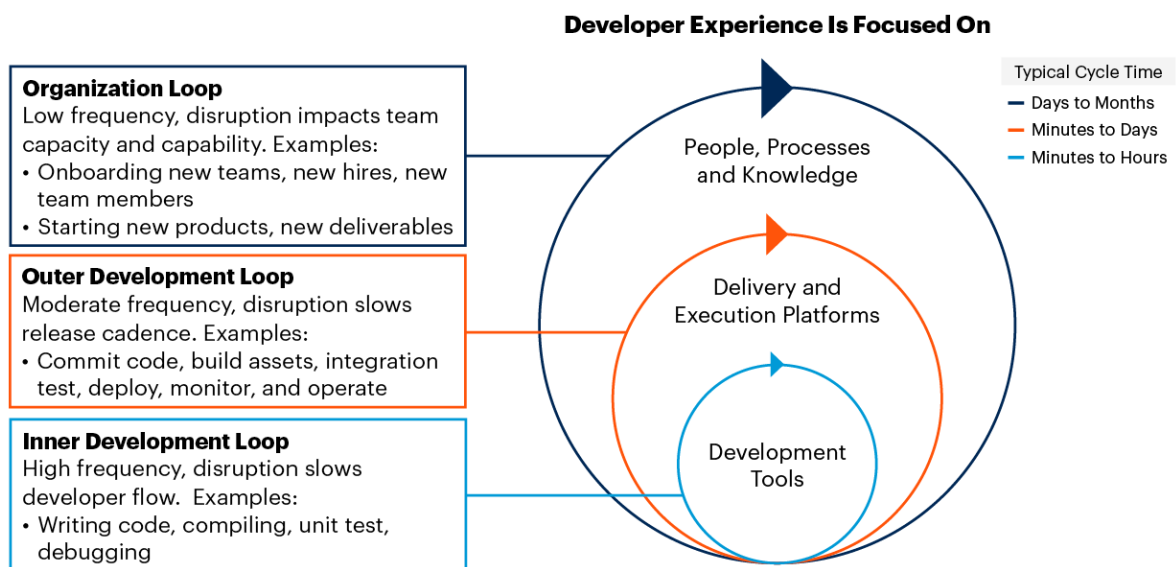
Adopt Internal Developer Platforms to Improve the Developer Experience

Internal developer platforms provide developers with a coherent, integrated and abstracted set of tools and services that streamline and simplify the work required to deliver working software with quality and security. The capabilities of an internal developer platform should be identified and prioritized by working with developers to understand where they experience the most friction, frustration or cognitive load impacting their delivery of value (i.e., working software).

The developer experience can be viewed as three loops that capture software development activities at different cadences, as shown in Figure 7. The inner and outer development loops in the software industry are widely used to describe the actions of a software engineer and how they interact with specific tools (whether self-, team- or platform-sourced). The organization loop captures elements of the developer experience that fall outside the inner and outer loops.

Figure 7: Three Loops of Software Development

Three Loops of Software Development



Source: Gartner
776873_C

In more depth, the three loops of software development are:

- **The inner loop** represents the activities of developers designing, writing, unit testing, debugging and committing code to version control, and typically happens on a cadence measured in minutes or hours. The developer experience for the inner loop is strongly influenced by:
 - Integrated development environment (IDE) or a code editor
 - Programming language, runtime and frameworks for development and testing
 - Build tools and debugging tools
 - Dependency management tools
 - Developer resource access (e.g., local container management or development infrastructure environments)
- **The outer loop** represents the activities that follow completion of a coding task, to get changes deployed and operational. The cadence for the outer loop can be anything from minutes to days. Note that some aspects will vary depending on the pipelines and environments used. The developer experience for the outer loops is impacted by capabilities such as:
 - Software version control processes
 - Build and test automation pipelines — including any integrated validation, scanning or governance
 - Deployment and release automation, including packaging, infrastructure-as-code processes, policy-as-code enforcement and feature flag support
 - Monitoring and observability tools to support validation of successful deployment and diagnostics of any issues

- **The organization loop** represents activities necessary at the organizational level (e.g., team or group) to enable software delivery. The cadence of this loop is typically measured in days to months. The developer experience for the organization loop includes:
 - Establishing new teams and the resources required, including provisioning accounts and environments with the developer platform
 - Onboarding new developers (whether new-hire or transfers from other teams), including documentation and training materials to help them be productive as quickly as possible
 - Initiating the creation of new software products or platforms, or new components within an existing product from templates, or changing processes and using automation to ensure consistency and alignment with organizational requirements and guidelines

Naturally, the fast cadence of the inner loop means that small gains in efficiency or improved flow are amplified — saving a few seconds of time hundreds or thousands of times a day across a team adds significant benefit. Conversely, the relatively slow cadence of the organization loop means that any capability you invest in there must result in substantial improvements each time it is used. Systems thinking is required to ensure broader process inefficiencies do not negate inner loop improvements.

After engaging with development teams to understand their challenges and needs, you can use the three loops to assist with planning and prioritizing your internal developer platform capabilities. Each capability should be mapped to these loops, helping to:

- Identify which capabilities are closely related to each other in the developer's use of the platform. For example, when two capabilities are used in the same time window, or when the output of one capability is used as the input to another. In these instances it is important to have a low-friction user experience.
- Determine which capabilities will be used most frequently and which will have the most impact each time they are used.
- Identify the criticality of the capability to the software's delivery and operation. For example, users are directly affected if the production environment is unavailable. But, if deployment pipelines are unavailable, changes cannot be delivered, indirectly affecting users.

Improving your internal development platform can be valuable at any scale. At the lowest scale, it may simply encourage developers to create helpful documentation of critical processes or develop and share tools that help automate repetitive tasks. At scale, if your development organization has multiple teams with similar challenges, you should evaluate whether to apply platform engineering principles to deliver a valuable and sustainable platform. See [Adopt Platform Engineering to Improve the Developer Experience](#) for more details.

Monitor Engineering, Business and Customer Metrics to Validate Change

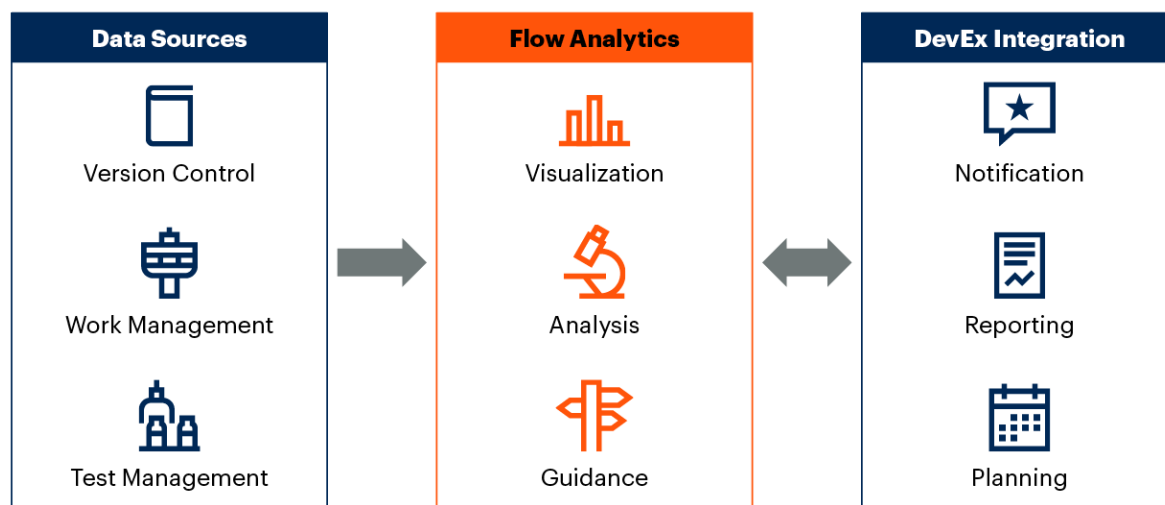
Validating change by monitoring key engineering, business and customer metrics is essential for developing successful delivery processes. Businesses that develop and leverage this capability have a distinct competitive advantage. Active monitoring of metrics enables data-driven and evidence-based decisions to continuously improve the development and delivery of business and customer value.

Track engineering metrics to evaluate system performance, identify process issues, support root cause analyses and improve the developer experience. Perform business metric tracking to provide valuable insights into regulatory compliance, risk mitigation and the impact of change. Integrate customer metrics to identify pain points, measure adoption and usage, track product satisfaction and maintain a customer-centric focus.

Establish and use a software engineering intelligence platform to provide software engineering leaders and their teams with insight into the software development process. These platforms collect and analyze large quantities of data from engineering systems to identify patterns and provide guidance. Interactive dashboards give software engineering leaders and their teams insights into developer productivity and value delivery (see Figure 8).

Figure 8: Connecting Flow Analytics Into the Value Stream

Connecting Flow Analytics Into the Value Stream



Source: Gartner
DevEx = Developer Experience
796376_C

Analyze value stream metrics to understand the flow of work through your process. Flow analytics benefit end-to-end visibility, data-driven decisions, constraint-focused improvements and collaborative ownership. Follow this guidance to implement flow analytics:

- Define the business and engineering goals to guide tool adoption and usage.
- Determine role-based objectives to identify corresponding metrics.
- Identify which software development tool connections are required to deliver the data.
- Incorporate regular flow metrics usage and reviews into your process.

Implement an efficient monitoring system to measure and share metrics. Build a data-driven culture to socialize and make decisions based on the metrics collected.

Read [Essential Metrics for Agile and DevOps Teams](#) for more information on metrics and [Analyze Value Stream Metrics to Optimize DevOps Delivery](#) for flow analytics.

Poorly Managed Front-End Dependencies Will Force a Reevaluation of Frameworks

[Back to Top](#)

Over the past decade, heightened user expectations and more powerful devices and browsers have led to a proliferation of front-end frameworks such as Angular, ReactJS and Vue.js. These have introduced many open-source dependencies that require frequent maintenance as the frameworks and libraries constantly innovate through rapid release cycles.

Meanwhile, applications developed on these frameworks — especially the first generation of “single-page applications” running user interface (UI) logic client-side — suffer from a legacy architecture. With early generations of these rich front ends, it was difficult to manage dependencies, maintain code, and support interoperability with other frameworks and emerging architectures. Best practices were not always clear, and changes between the major releases of these frameworks often altered those best practices. The pace of innovation in the front-end framework landscape is unlikely to slow, although features for most frameworks have stabilized over the past year.

While most organizations will continue to work with front-end frameworks, it is imperative that foundational understanding of how to implement quality front-end applications without using frameworks drives your 2024 planning efforts. Without core and basic knowledge of your coding language (often JavaScript) and modern browser features, framework overuse will add to ongoing technical debt.

Software engineers should follow these planning considerations as they implement front-end applications and evaluate their front-end frameworks:

- [Avoid Front-End Frameworks for New Development](#)
- [Learn and Teach Teams Modern Web Development Basics](#)
- [Choose Only Front-End Frameworks that Embrace and Build Upon Open Standards](#)

Planning Considerations

Avoid Front-End Frameworks for New Development

Early generations of web developers were often constrained in how they built and deployed rich front-end code. Frameworks did not integrate well, each having its own mechanisms for the core features of such a framework. These features included, but were not limited to:

- **Templating and data binding** — How variable data, typically from a server data source, gets displayed.
- **Client-side routing** — How client-side state gets managed with respect to URLs.
- **Code structure** — How developers separate concerns and organize code.
- **Build tools** — How code, including dependencies, gets bundled and deployed.
- **Modular architecture** — How developers define granularity or components and interoperation of code.
- **Deployment and loading** — How code gets to production and is loaded into running applications.

Only the first two of these features strictly require a front-end framework today, as the modern web browser itself does not provide what may be sufficiently complex mechanisms for some use cases. However even here, as with templating and data binding, a framework-free solution can often be preferable where the need for dynamic data is limited. For the rest of the feature list above, recent innovations in browser capabilities and the JavaScript language mean that developers can reduce their dependency on frameworks. This, in turn, results in better code portability, improved maintainability and the potential for future-proof UI development.

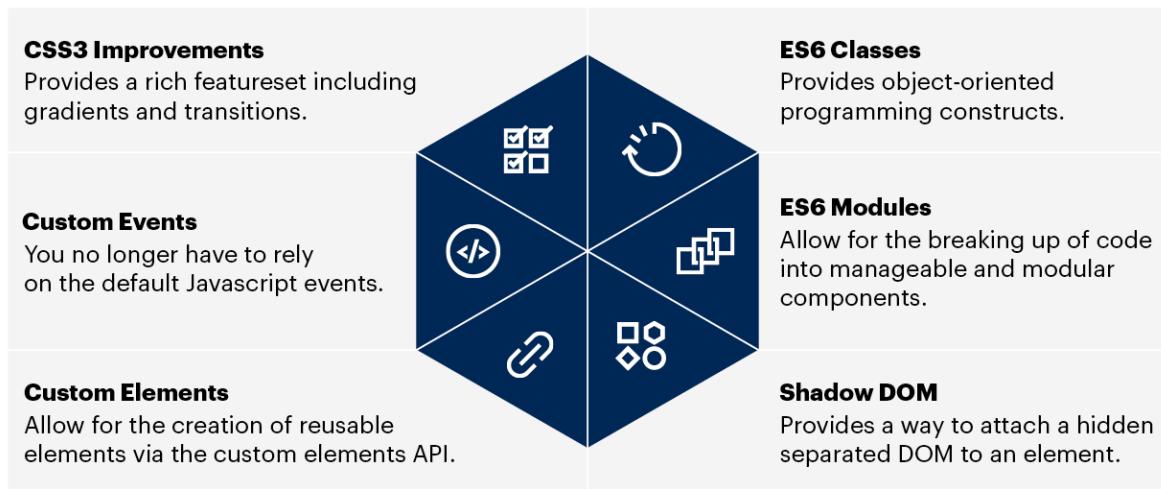
Browser features for front-end development without using a framework (see Figure 9) include:

- **ES6 classes** make it easier to build object-oriented and more reusable code without a framework. These can extend most well-proven built-in browser classes, such as `HTMLElement`.
- **ES6 modules** with import and export syntax provide a dynamic and interoperable way of loading code.
- **Custom elements** allow for a standards-based way of building and reusing individual components.

- **Shadow Document Object Model (DOM)** allows developers to deploy components in a way that “hides” their inner functionality from the global browser scope.
- **Custom events** give developers a standards-based way to communicate between components.
- **CSS3 improvements** make it easier to provide rich interfaces.

Figure 9: Tools for Front-End Development Without a Framework

Tools for Frontend Development Without a Framework



Source: Gartner
796376_C

These six front-end development features are supported in all modern, evergreen browsers (Chrome, Edge, Safari, Firefox, etc.) and are usable by all front-end developers without a dependency on external frameworks or libraries.

Learn and Teach Teams Modern Web Development Basics

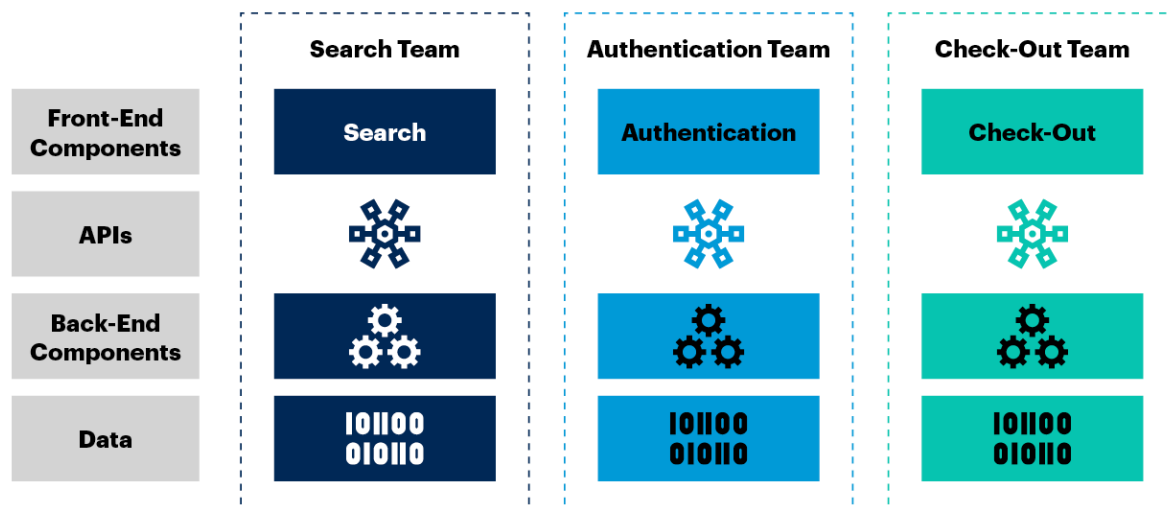
Today’s web browser is the closest thing modern developers have to a “future-proof” UI runtime, and yet too few developers have expertise with its features. What sounds like a trite observation to return to basics is more important than ever, because the basics aren’t at all what they used to be. To illustrate, [this CodePen link](#) shows a simple self-contained counter. It is implemented in pure JavaScript, using custom elements, and dynamically loadable into any web application regardless of the modern browser used, or other frameworks or libraries also running within the application. This example uses many of the browser features shown in Figure 9, which previously required libraries or frameworks to develop and execute.

The problem of framework overdependence and lack of developer familiarity with these features extends beyond legacy applications. Net new development by teams using frameworks such as Angular and ReactJS also tends to be bound entirely to those frameworks, when they may be unnecessary. This is largely a byproduct of such teams having learned front-end web development through the “lens” of a particular framework, or because they have built up ready-to-use dependencies to accelerate development using these frameworks.

The growing popularity of micro-frontend (MFE) architecture presents additional challenges to teams unfamiliar with the tools shown in Figure 9. MFEs allow teams to deploy functionality to production independently and frequently while ensuring that micro-frontends can interoperate within the same application. In many cases, an MFE architecture must support multiple frameworks, especially where third parties must contribute, and different teams may prefer a different framework. In all cases, the aforementioned browser enhancements significantly reduce a development team’s dependency on a framework if teams take the time to learn these core basics. Figure 10 shows an MFE application architecture, where different teams may deploy features separately, own the complete vertical “stack” of their functionality, and even use other frameworks.

Figure 10: Micro-Frontend Architecture

Micro-Frontend Architecture



Source: Gartner
773697_C

MFE architecture should not be considered a best practice or a required architecture, as the challenges of governance and quality of user experience are substantial. Regardless of whether an MFE architecture is necessary, solid modularity and general framework independence should be considered prerequisites to any such architecture.

The core challenge of framework dependency and unmaintainable code using JavaScript frameworks has a straightforward solution. Teams must first learn and adopt the standards-based mechanisms to solve the fundamental needs of code organization and distribution, and only then choose a framework to solve the problems not solved by the browser itself. This, in turn, requires a shift from “framework-first” development to “framework-last,” where significant dependencies are only introduced where absolutely necessary.

Read [When to Use ReactJS for Building Innovative Web Applications](#) to learn more on the popular JavaScript library.

Choose Only Front-End Frameworks that Embrace and Build Upon Open Standards

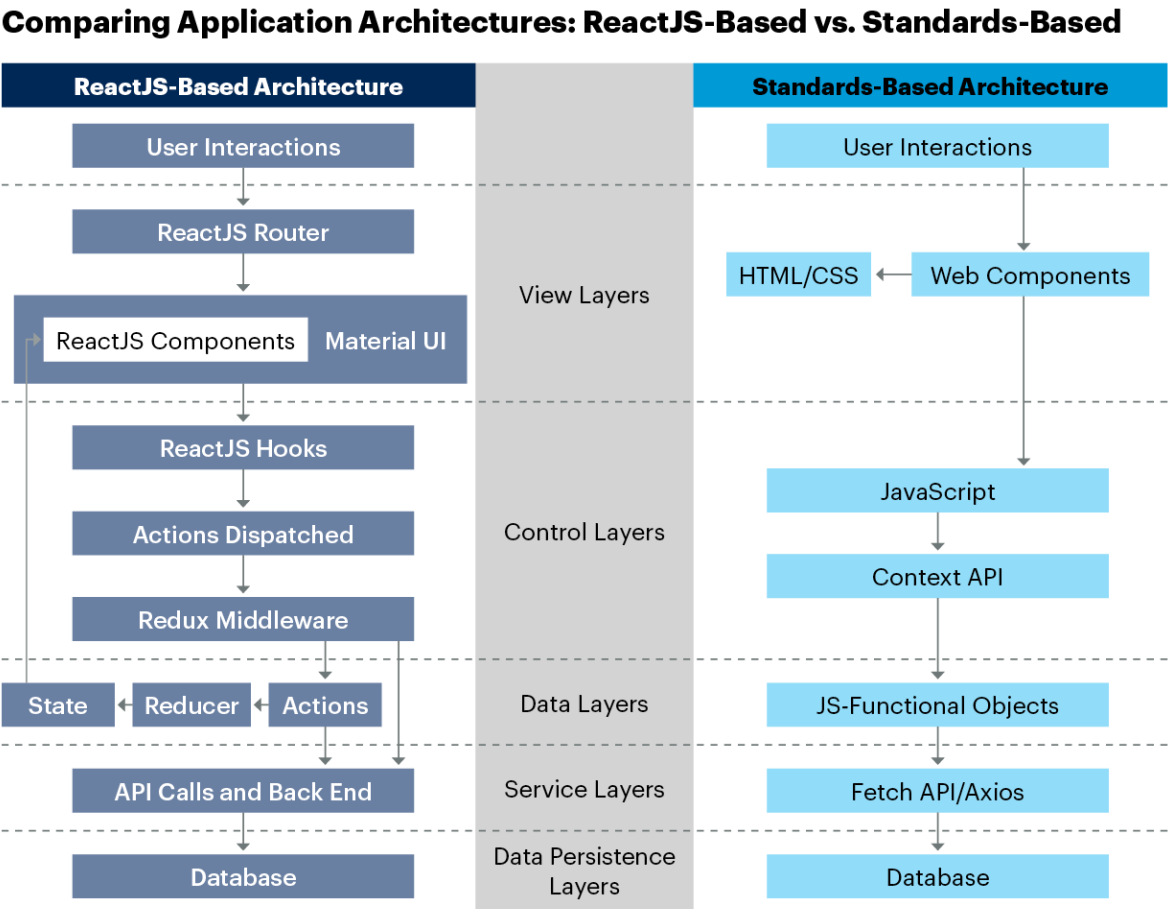
It is possible today to build a complex application front end without a framework. Doing so may sound counterintuitive to any modern developer coming from big frameworks such as traditional ASP.NET, large Angular or ReactJS deployments, or cross-compilers that promise truly universal code portability. This strategy treats web browsers and standards as the centerpiece of client-side application development and won't be the best fit for all teams or use cases. A large application may need Angular as its “shell” to maintain its overall state while loading many smaller components, including those using Angular or vanilla JavaScript.

Figure 11 compares ReactJS-based and standards-based architectures, emphasizing their typical characteristics and general flow. This depiction is not of universally accepted best practices, but instead aims to showcase common elements of and differences between the two approaches.

- ReactJS-based architecture emphasizes integration and cohesion, streamlining development by using consistent tools and patterns. This provides a unified development experience at the cost of dependencies on specific technologies; limiting flexibility and increasing the complexity of updates and maintenance. Other frameworks such as Angular and Vue.js encourage similar approaches.

- Standards-based architecture leans on open web technologies, inherently facilitating integration with a broader range of tools and platforms. While this design is less anchored to a specific framework or library, making it potentially more resilient to technological shifts, it demands a strong familiarity with the foundational web technologies. Moreover, without careful oversight, it will lack the tight-knit integration often found in framework-focused architectures such as ReactJS.

Figure 11: Comparing Application Architectures: ReactJS-Based vs. Standards-Based



Source: Gartner
796376_C

What developers require from frameworks has changed. Recent enhancements to JavaScript frameworks have — sometimes slowly — incorporated the features described in Figure 9. This hasn't been easy for the major JavaScript frameworks since much of their original architecture aimed to solve these same challenges in more proprietary ways. For example, Angular has now introduced Angular elements (portable custom elements) and stand-alone modules (which use ES6 modules as their deployment mechanism). ReactJS's rendering model, which takes ownership of the UI changes, has always been largely incompatible with custom elements. This makes it difficult to use frameworks in combination with ReactJS or deploy built components without developers having to understand its dependencies.

The requirements of a scalable and maintainable modern front-end architecture require more than what any single framework provides. The web browser will remain an indispensable aspect of all modern front-end development, even for teams focused primarily on native mobile app development, where a WebView is often used to embed web content. Being standards-based and supporting many runtimes, it is the closest thing developers have to a “future-proof” front-end runtime. As such, and given the relative ease of learning its core technologies, all front-end developers and architects should gain a strong understanding of its features to make the best choices for their own organizations regarding user experiences, frameworks and dependencies.

Technology Innovation Will Increase the Demand for Upskilling and Reskilling

[Back to Top](#)

The acceleration in digital transformation, with agile, DevOps, platform engineering and AI, in addition to constantly evolving technologies, requires a skilled workforce who continue to grow their knowledge and develop their expertise. Software engineers who embrace continuous learning can maintain the pace of change necessitated by increased business demand, faster delivery cadence and discerning users.

Prioritize your own learning by dedicating time in your calendar to participate in learning activities that advance your knowledge, support your team and focus on developing yourself. Create a personal learning plan to identify skills to focus on, activities to perform and to create motivating short-term goals. Dedicate study time in your calendar to prioritize this activity, as it will help your career, team and organization.

Software engineers should follow these planning considerations as they upskill and reskill:

- Follow Agile Learning Values and Principles to Guide Continuous Learning
- Identify and Address Critical Skills Gaps to Drive Team and Organizational Success
- Embrace Change and Lifelong Learning to Maintain Essential Expertise

Planning Considerations

Follow Agile Learning Values and Principles to Guide Continuous Learning

Seek opportunities to embed learning in everyday activities to increase skill development opportunities, shorten the learning-to-earning cycle and significantly improve learning outcomes. Follow the values and principles defined in the Agile Learning Manifesto (see Figure 12) to develop your continuous learning strategy and make personal development central to your role, not dedicated time away from your regular work.

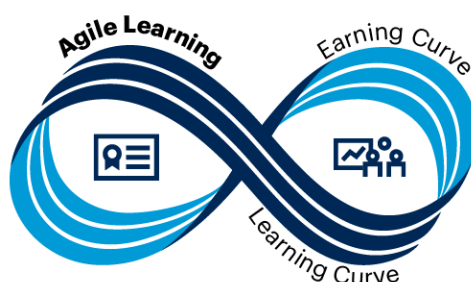
Figure 12: The Agile Learning Manifesto

The Agile Learning Manifesto

Values

- ★ **Business outcomes**
over knowledge gained
- ★ **Growth mindset**
over current skill set
- ★ **Real-time embedded**
over training time offline
- ★ **Community compounding**
over individual practicing

Source: Gartner
796376_C



Principles

- 1 Learning to earning
- 2 Motivation multiplier
- 3 Just-in-time microbursts
- 4 Dynamic pathways
- 5 Progressive layering
- 6 Flow of value delivery
- 7 Data-driven, AI-enabled
- 8 Socially amplified

Digital trends are shortening the half-life of technical skills and even specific roles. Adopt a growth mindset to develop new skills and hone existing ones through dedicated effort and practice. Continuous disruption requires you to continuously upskill and periodically reskill to adapt to change and benefit from dynamic technology shifts. Being a fast learner makes you a highly valuable employee. Take control of your learning, do not wait until your manager identifies a business requirement to learn new skills — you are the author of your own career.

Read [Proven Practices to Enhance Application Architecture and Software Development Skills](#) to create a personal learning plan and embed learning in your daily activities. Identify role-based skills to learn by reading [Essential Skills for Application and Software Developers](#), [Essential Skills for Agile Development](#) and [Essential Skills for QA Engineers](#).

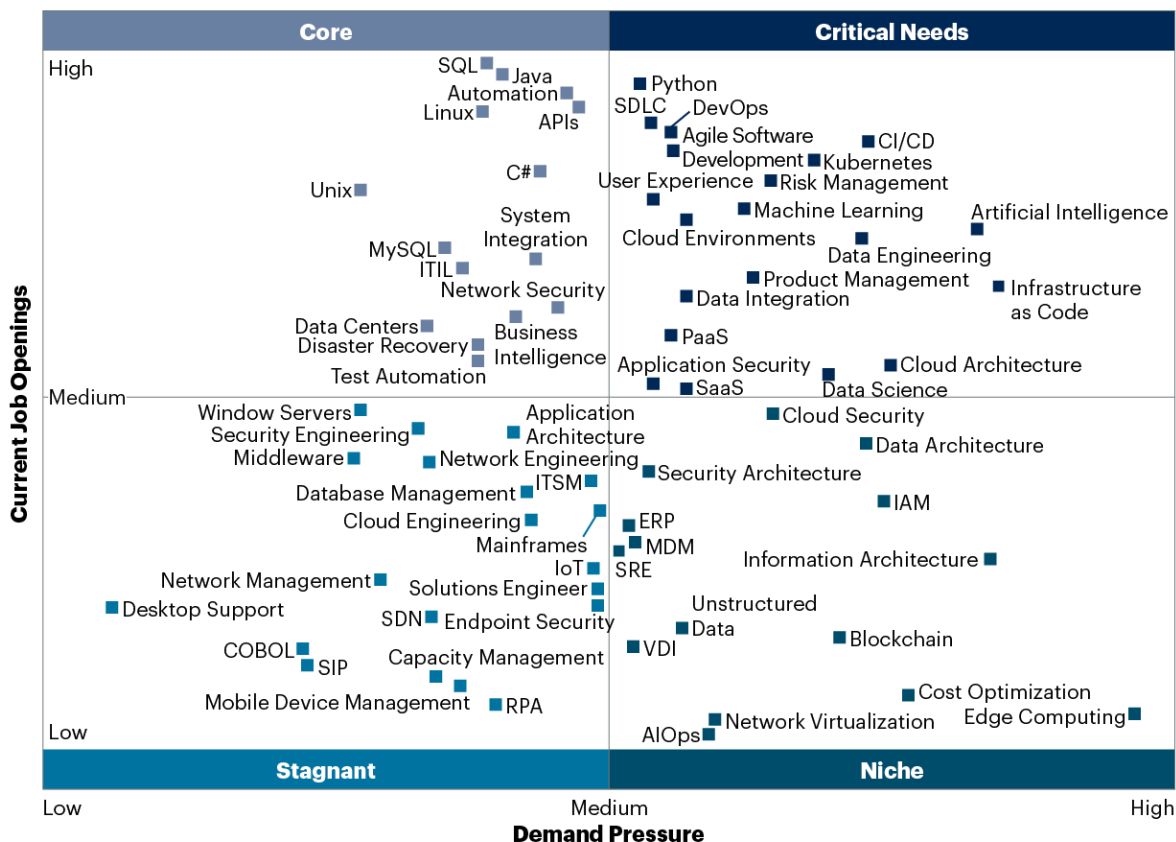
Identify and Address Critical Skills Gaps to Drive Team and Organizational Success

Software engineers must possess a broad spectrum of technical skills including: architecture, design, testing, automation, DevOps and user experience (UX) expertise; and technology skills such as: development languages, frameworks and tooling. Expand your skills and the value you offer your team and organization by becoming a generalizing specialist with a deep core specialty and strong knowledge in several other disciplines. Generalizing specialists are resilient and adaptive, able to benefit from knowledge intersections and rapidly increase their skill level in adjacent subjects.

Assess your organization's capability by speaking with your engineering and product management, engaging with your architecture and design groups, and joining related communities of practice. Identify stagnant skills to avoid, core skills to maintain, critical needs to develop and niche skills to explore (see Figure 13).

Figure 13: IT Skills Matrix

IT Skills Matrix



Source: 2022 Gartner IT Skills Roadmap
796376_C

Effective generalizing specialists are not hired — they develop themselves by learning from experience, by working in situations that provide new learning opportunities and by embracing an agile, continuous learning mindset. Broaden your scope by seeking opportunities to work with others and explore new roles in adjacent areas. Use the knowledge you gain to help others upskill and reskill. Support your team, advance your career and future-proof your knowledge through continuous learning and addressing the critical skill gaps that drive your team and organization's success.

Broaden your knowledge of agile, DevOps and adjacent areas by reading:

- [Agile Superhero Training Manual](#) and [Keys to DevOps Success](#)
- [Essential Skills for Cloud-Native Application Architects](#)
- [Essential Skills for Modern Integration Architecture](#)

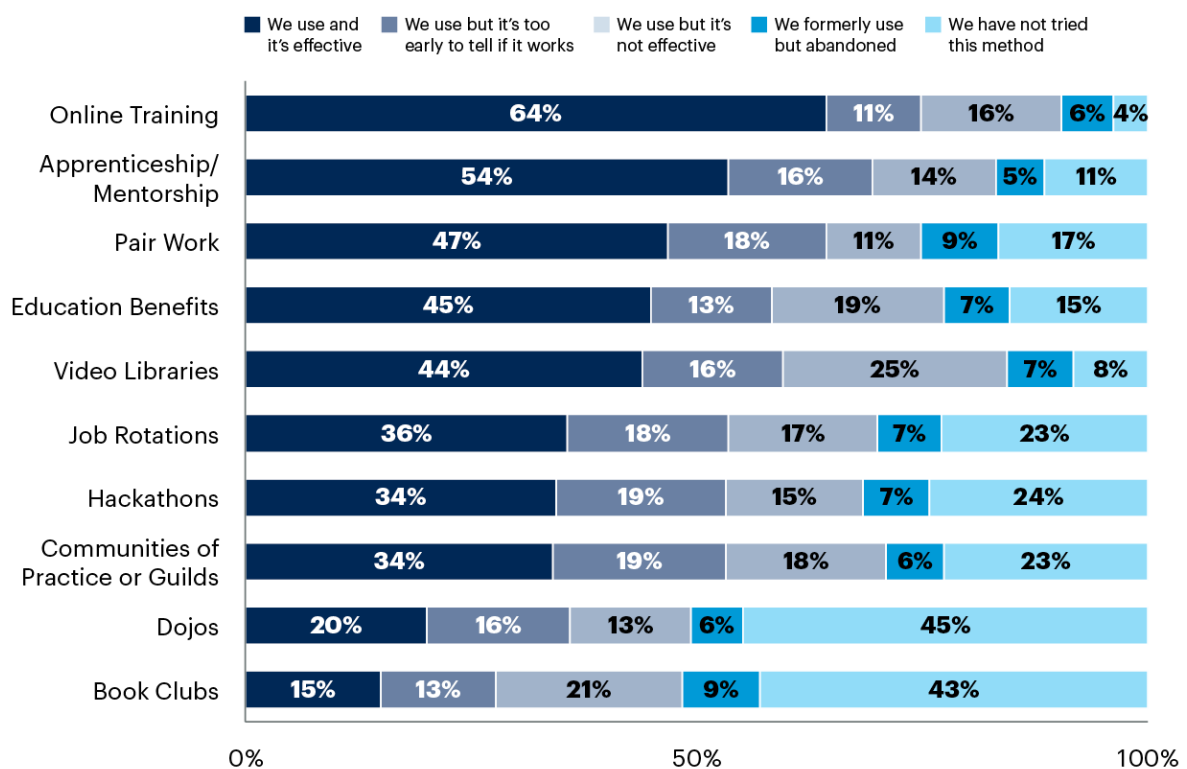
- Essential Skills for Infrastructure Automation Engineers
- Essential Skills for Security in Modern Application Development.

Embrace Change and Lifelong Learning to Maintain Essential Expertise

Speak with your learning and development group and line manager to determine the learning channels that your organization provides. Take advantage of offered opportunities within your workplace and seek others beyond. A recent Gartner survey of software engineering leaders shows the learning channels they found to be the most effective (Figure 14).

Figure 14: Knowledge and Skill Development Effectiveness

Knowledge and Skill Development Effectiveness



n = 314; software engineering leaders

Q: How effective or ineffective are the following methods in helping staff develop knowledge and skills?

Source: 2021 Gartner Software Engineering Leaders Survey

796376_C

Hold a quarterly personal learning retrospective with your professional development manager to refine your learning technique and verify that you are investing your time learning the right skills and applying them effectively. Blend individual learning (reading, online training, etc.) with social learning (communities of practice, hackathons, etc.) and practice-based experiential learning (pairing, job rotations, etc.). Recognize that technical skills are only one aspect of your learning journey; you must also incorporate communication, collaboration, writing, ways of working, presentation and leadership skills.

Read [Community of Practice Essentials](#) and [Develop Your Technical Skills Using Online Learning Platforms](#) for more information on learning options.

Evidence

Gartner Software Engineering Leaders Survey. The 2021 Gartner Software Engineering Leaders Survey was conducted to understand the challenges and responsibilities of software engineering leaders. The research was conducted online from April through June 2021 among 314 respondents from North America (n = 155), Western Europe (n = 103) and Asia/Pacific (n = 56). Respondents were screened as responsible for at least one team of software engineers at organizations with over \$20 million in worldwide revenue across organizations from all industries, except construction, natural resources, energy, some manufacturing sub-industries, local or regional government, and wholesale. The survey was developed collaboratively by a team of Gartner analysts and Gartner's Research Data, Analytics and Tools team. Disclaimer: Results of this survey do not represent global findings or the market as a whole, but reflect the sentiments of the respondents and companies surveyed.

¹ [Ozzie Memo: "Internet Services Disruption,"](#) CNET.

² [Product for Internal Platforms,](#) Medium.

Document Revision History

[2023 Planning Guide for Application Development - 10 October 2022](#)

[2022 Planning Guide for Application Development - 11 October 2021](#)

[2021 Planning Guide for Agile Software Development and DevOps - 9 October 2020](#)

[2020 Planning Guide for Software Development and UX - 7 October 2019](#)

[2019 Planning Guide for Software Development and UX - 5 October 2018](#)

Recommended by the Authors

Some documents may not be available as part of your current Gartner subscription.

[2024 Planning Guide for Application Architecture, Integration and Platforms](#)

[Create, Optimize, Integrate: The 3 Pillars of Building Effective Automated Tests](#)

[Decision Point for Choosing a Mobile App Architecture](#)

[Essential Skills for Application Architecture](#)

[How to Create an Effective Test Plan](#)

[How to Effectively Test Mobile Apps](#)

[Organize for Agility With Team Topologies](#)

[Shift Left With Test-Driven Development](#)

[Solution Path for Agile Transformation](#)

[Solution Path for Continuous Delivery With DevOps](#)

© 2023 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner is a registered trademark of Gartner, Inc. and its affiliates. This publication may not be reproduced or distributed in any form without Gartner's prior written permission. It consists of the opinions of Gartner's research organization, which should not be construed as statements of fact. While the information contained in this publication has been obtained from sources believed to be reliable, Gartner disclaims all warranties as to the accuracy, completeness or adequacy of such information. Although Gartner research may address legal and financial issues, Gartner does not provide legal or investment advice and its research should not be construed or used as such. Your access and use of this publication are governed by [Gartner's Usage Policy](#). Gartner prides itself on its reputation for independence and objectivity. Its research is produced independently by its research organization without input or influence from any third party. For further information, see "[Guiding Principles on Independence and Objectivity](#)." Gartner research may not be used as input into or for the training or development of generative artificial intelligence, machine learning, algorithms, software, or related technologies.