2024 Planning Guide for Application Architecture, Integration and Platforms

Published 4 October 2023 - ID G00796446 - 41 min read

By Analyst(s): Matt Brasier, Richard Watson, Gary Olliffe, Brad Dayley, Steve Deng, Kevin Matheny, Eklove Mohan, Kyle Davis

Initiatives: Application Architecture and Integration for Technical Professionals; Software Development for Technical Professionals

In 2024, application technical professionals must confront a rapidly expanding set of technologies and approaches, such as GenAI and platform engineering, while supporting rising demand for automation and modernization. Here, we explore these trends and associated planning considerations.

Overview

Key Findings

- Generative AI (GenAI) technologies are receiving a lot of hype. Because of GenAI's far-reaching potential, it requires a more detailed capability analysis than other application technology trends from the last few years, such as metaverse and Web3.
- Digital transformation, modern application architectures and software development practices are all driving an increasing need for automation.
- Modernizing older applications is a business-as-usual task for application teams. However, technology-driven modernization approaches that fail to address business needs will make stakeholders less inclined to fund future modernization efforts.
- Platform engineering, along with a trend toward exploring new technology choices (versus defaulting to REST APIs and microservices), is altering the boundaries of application architecture, requiring application architects to work closely with platform architects.

Recommendations

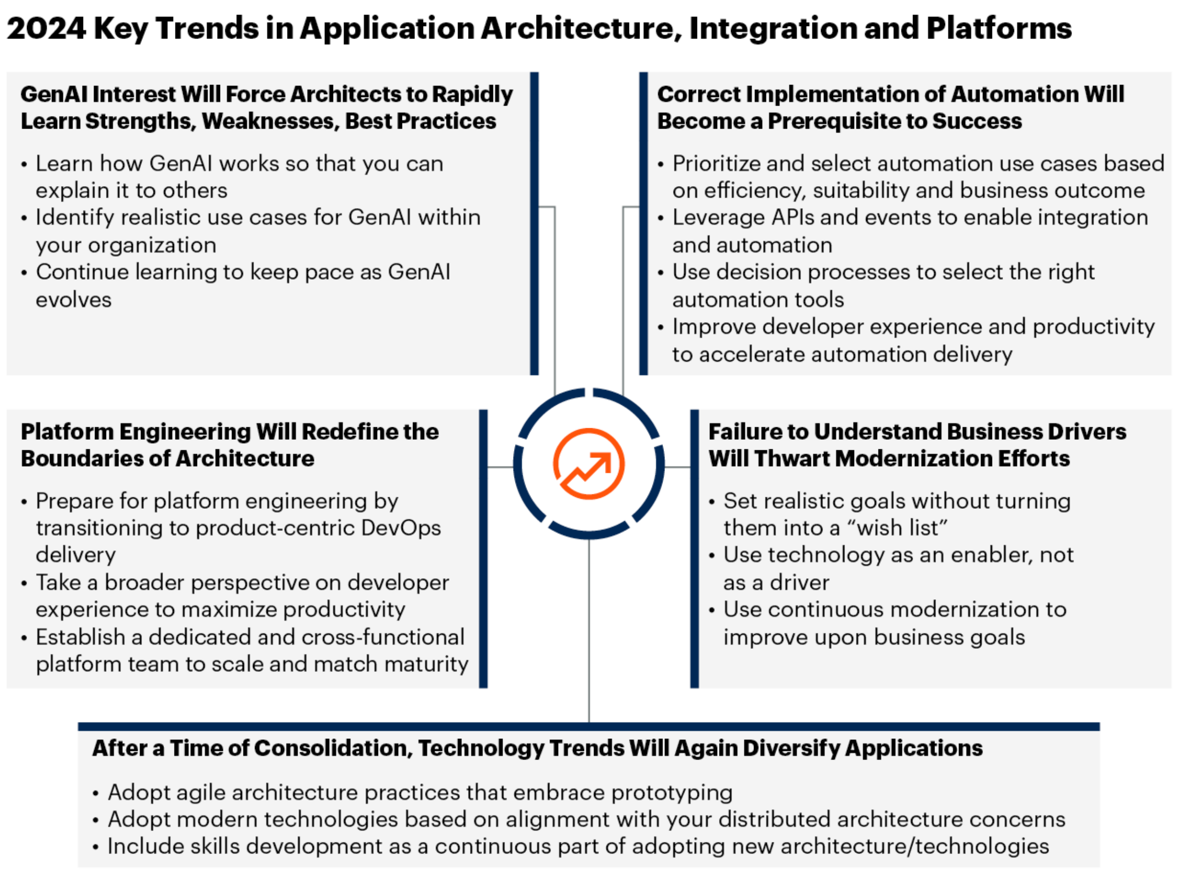
- Learn the basics of GenAI technology, so that you can differentiate between genuine areas of opportunity and hype-driven use cases that do not offer real-world benefits.
- Enable optimal selection of automation tools by implementing an automation strategy that supports multiple types of automation technology and provides guidance on when to use each approach.
- Select the optimal application technologies and tools for each architecture (not necessarily the newest technologies) by setting realistic goals for modernization based on business needs and outcomes.
- Accelerate development by adopting a platform engineering approach that treats application platforms as products delivered by agile teams.

Application Architecture, Integration and Platform Trends

In the previous iteration of this Planning Guide, we highlighted that the job of application architects is getting harder. This trend will continue into 2024. Not only must architects deal with a growing appetite for automation and modernization, but they must also learn and adapt to new trends, such as generative AI and platform engineering. Meanwhile, the recent era of technology consolidation — with teams preferring common technologies, such as REST APIs, microservices and Kubernetes containers — is starting to come to an end. Technology selections are becoming increasingly diverse among teams.

Gartner's application architecture and integration research for 2024 will have a strong focus on helping you establish the skills, tools and approaches needed to successfully navigate this period of rapid change. Figure 1 illustrates the key trends and associated planning considerations that application and integration architects will face in 2024.

Figure 1: 2024 Key Trends in Application Architecture, Integration and Platforms



2024 Key Trends in Application Architecture, Integration and Platforms

GenAI Interest Will Force Architects to Rapidly Learn Strengths, Weaknesses, Best Practices

- Learn how GenAI works so that you can explain it to others
- Identify realistic use cases for GenAI within your organization
- Continue learning to keep pace as GenAI evolves

Correct Implementation of Automation Will Become a Prerequisite to Success

- Prioritize and select automation use cases based on efficiency, suitability and business outcome
- Leverage APIs and events to enable integration and automation
- Use decision processes to select the right automation tools
- Improve developer experience and productivity to accelerate automation delivery

Platform Engineering Will Redefine the Boundaries of Architecture

- Prepare for platform engineering by transitioning to product-centric DevOps delivery
- Take a broader perspective on developer experience to maximize productivity
- Establish a dedicated and cross-functional platform team to scale and match maturity

Failure to Understand Business Drivers Will Thwart Modernization Efforts

- Set realistic goals without turning them into a "wish list"
- Use technology as an enabler, not as a driver
- Use continuous modernization to improve upon business goals

After a Time of Consolidation, Technology Trends Will Again Diversify Applications

- Adopt agile architecture practices that embrace prototyping
- Adopt modern technologies based on alignment with your distributed architecture concerns
- Include skills development as a continuous part of adopting new architecture/technologies

Source: Gartner
796446_C

The following technical planning trends are examined in this report (click links to jump to trends):

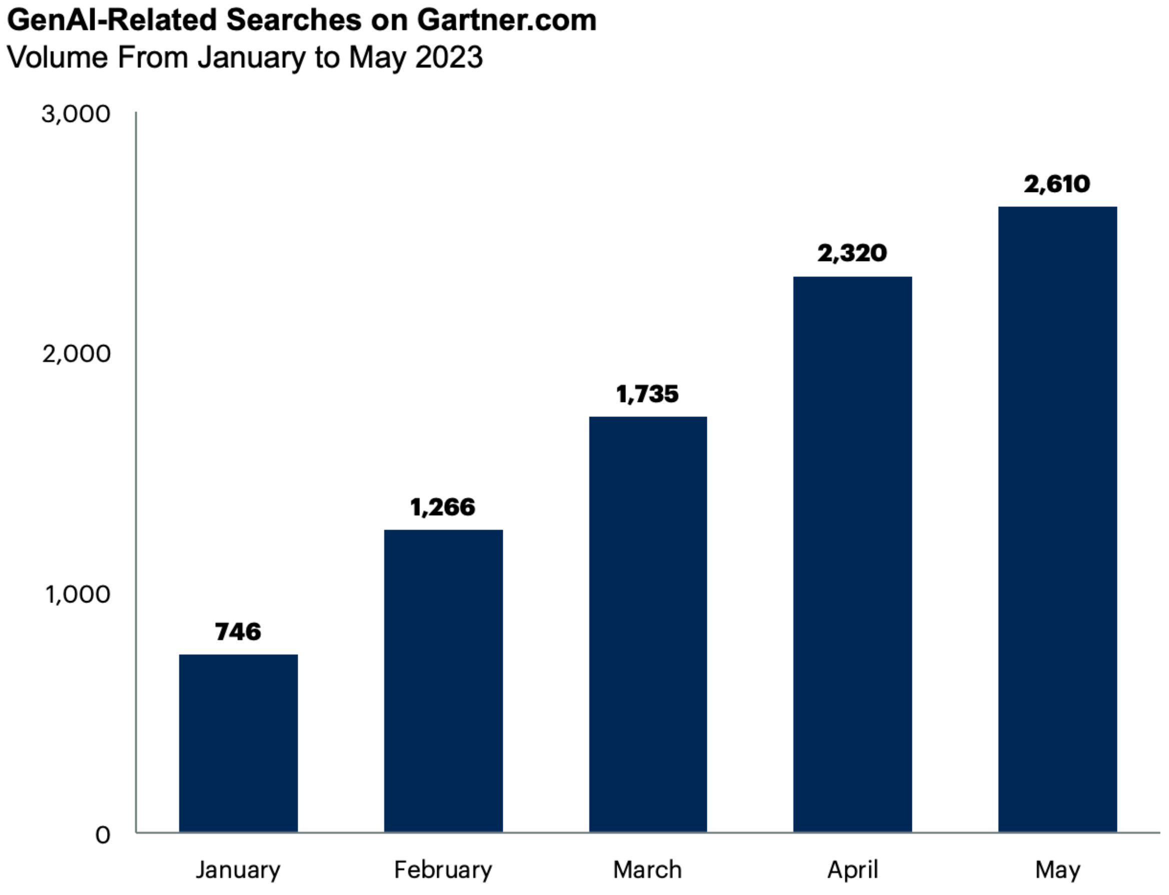
- Interest in generative AI will force architects to rapidly learn its strengths, weaknesses and best practices.
- Correct implementation of automation will become a prerequisite to success.
- Platform engineering will redefine the boundaries of architecture.
- Failure to understand business drivers will thwart modernization efforts.
- After a time of consolidation, technology trends will again diversify application architecture.

Interest in Generative AI Will Force Architects to Rapidly Learn Its Strengths, Weaknesses and Best Practices

Back to top

Like Web3 and non-fungible tokens (NFTs) in 2021, GenAI is everywhere, or at least, it seems to be. Unlike those prior technologies, GenAI offers some immediately available, easily accessed real-world benefits across a variety of skill levels and use cases. As shown in Figure 2, searches for generative AI on Gartner.com more than tripled from January 2023 to May 2023. GenAI in general and large language models (LLMs) in particular have become topics of interest for organizations of all sizes and across industries. Many organizations are seeking to understand how and where they can use GenAI in their business. Others are concerned about safeguarding their intellectual property or mitigating potential legal exposure from using GenAI.

Figure 2: GenAI-Related Searches on Gartner.com



Source: Gartner
794585

This increased interest has a direct near-term result: Architects and engineers are being tasked with investigating and implementing GenAI. To enable this investigation and implementation, technical professionals must quickly learn:

- How GenAI works
- What its strengths and weaknesses are
- What the costs, risks and pitfalls of using it are
- When its is best used
- Where it is best avoided (for now)

This task is complicated by the quickly evolving ecosystem of GenAI models, tools and techniques. It is made even harder by vendors that are incorporating GenAI into their offerings, creating new opportunities and new risks. The state of the art is constantly evolving, and technical professionals must keep pace.

To effectively evaluate and implement GenAI, architects and developers should follow these planning considerations:

- Learn how GenAI works so that you can explain it to others.
- Identify realistic use cases for GenAI within your organization.
- Continue learning to keep pace as GenAI evolves.

Planning Considerations

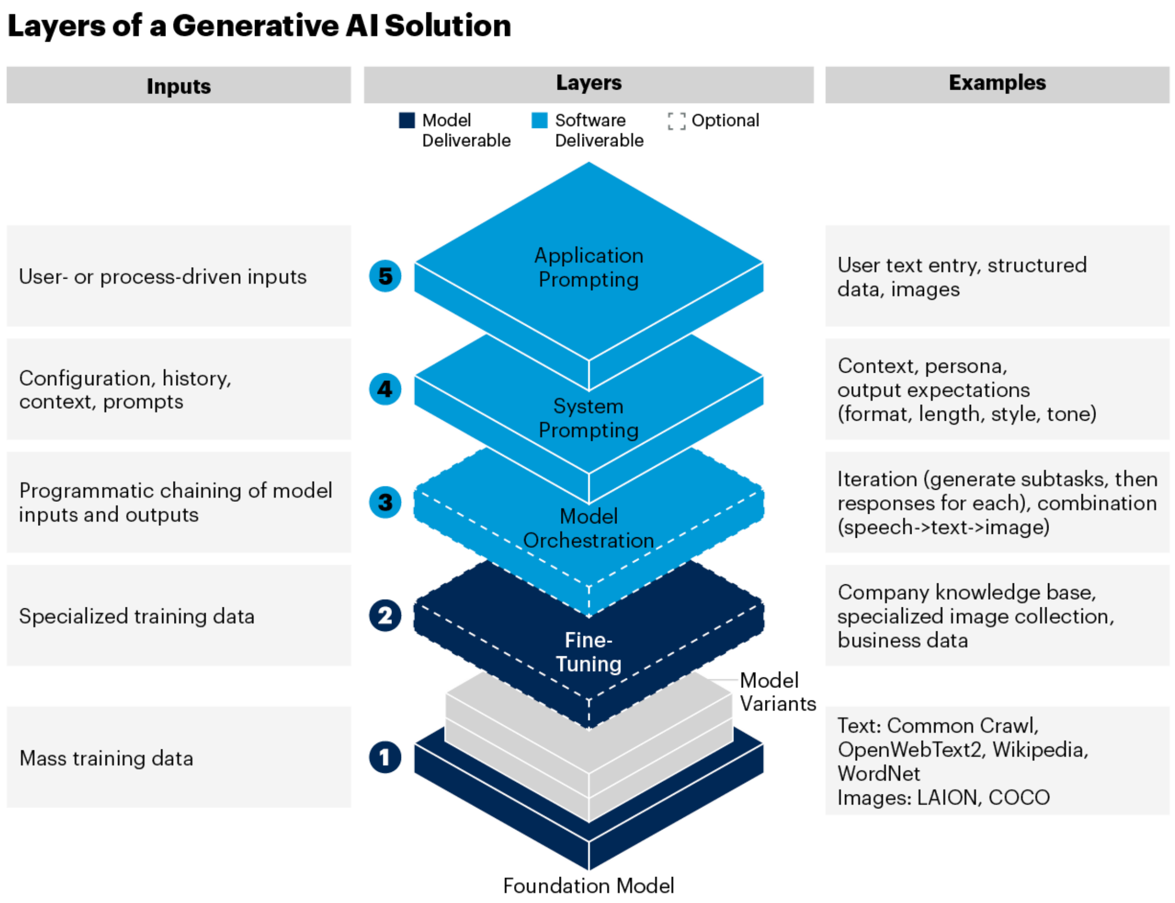
Learn How GenAI Works So That You Can Explain It to Others

Under normal circumstances, architects and software engineers play a central role in selecting software delivery technologies for their organizations. GenAI has created an unusual situation where senior leadership is directing the use of GenAI and often asking architects to find applicable use cases. Your knowledge and judgment are needed more than ever to provide guidance to leaders on when, where and how to apply GenAI toward solving business problems and creating advantages.

To ultimately evaluate and explain GenAI, start by learning how it works. Read articles and watch videos that are intended to teach the basics of GenAI to technologists. Prefer more recent content because of the fast pace of change in GenAI technologies and use cases. In addition, look for information from established sources with known technical skills to reduce the risk of learning things that are not true.

Gartner's own GenAI research is a valuable tool for this kind of understanding. For example, Getting Started With Generative AI in Your Application Architecture describes the logical structure of a GenAI solution, as shown in Figure 3. In addition, you can use the information in What Technical Professionals Need to Know About Large Language Models to stay current.

Figure 3: Layers of a Generative AI Solution



Layers of a Generative AI Solution

Inputs	Layers	Examples
User- or process-driven inputs	5 Application Prompting	User text entry, structured data, images
Configuration, history, context, prompts	4 System Prompting	Context, persona, output expectations (format, length, style, tone)
Programmatic chaining of model inputs and outputs	3 Model Orchestration	Iteration (generate subtasks, then responses for each), combination (speech->text->image)
Specialized training data	2 Fine-Tuning	Company knowledge base, specialized image collection, business data
Mass training data	1 Foundation Model	Text: Common Crawl, OpenWebText2, Wikipedia, WordNet Images: LAION, COCO

Source: Gartner
796446_C

Your next step is to discover the limitations of GenAI. Look at forums, communities (such as Stack Overflow and Reddit), and discussions to find the challenges and questions encountered by early adopters of GenAI. Excitement about GenAI is leading to many experiments, providing a rich source of information about what fails, what works and how well. While making mistakes is a valuable part of learning, learning from the failures of others is much less expensive than learning from your own. Answers to questions that your peers are asking, particularly for more established GenAI use cases, are likely to highlight the associated risks and benefits. Learning how to leverage the benefits and mitigate the risks will give you an advantage when determining the feasibility of using GenAI in a particular solution.

Once you have some ideas for use cases that are relevant to your organization, get hands-on experience with the technology. Create experiments aimed at discovering how your organization can use GenAI amid organizational constraints, tools and policies. Your overall goal is a level of understanding that lets you explain the technology and its constraints to both nontechnical leaders and fellow technologists.

Related Research

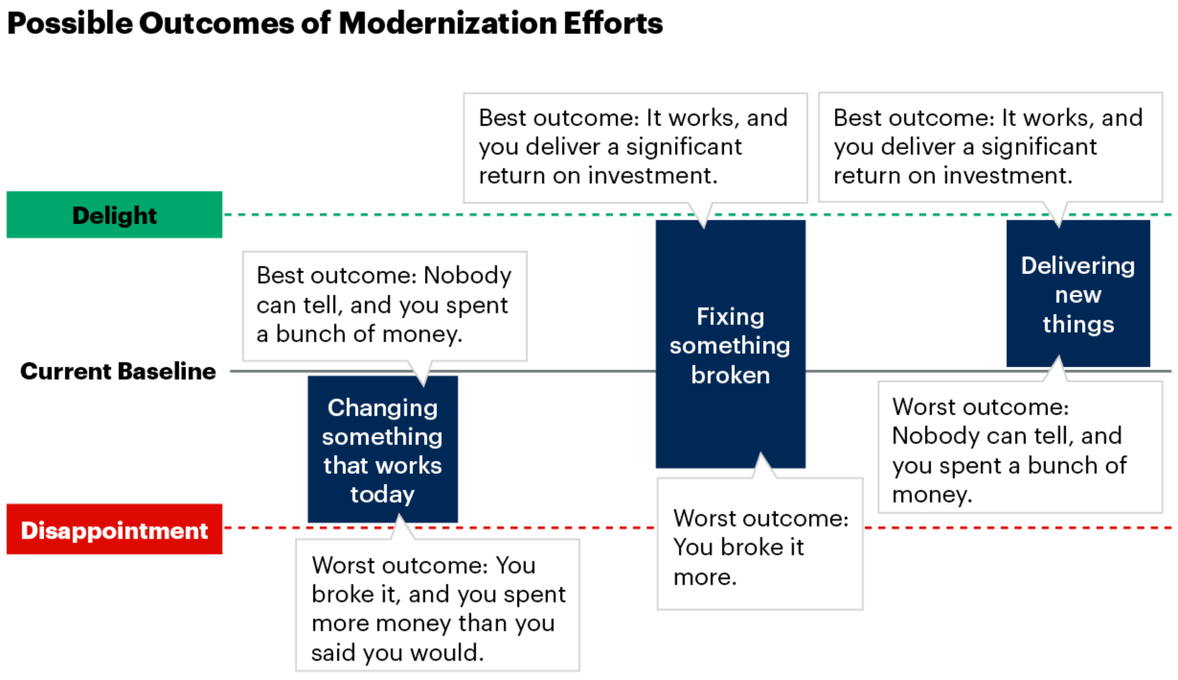
- Getting Started With Generative AI in Your Application Architecture
- Prompt Engineering With Enterprise Information for LLMs and GenAI
- What Technical Professionals Need to Know About Large Language Models

Identify Realistic Use Cases for GenAI Within Your Organization

Technology hype can result in teams picking a tool or approach, and *then* looking for reasons to use it — aka the “hammer looking for a nail” phenomenon. GenAI is far from the first technology to be overhyped, and it will not be the last. For recent examples of overhyped technologies that have not lived up to expectations, look no further than Web3 and NFTs, which we advised readers of the 2023 Planning Guide to approach with caution. Organizations that rushed to implement Web3 and NFT projects without first validating their ability to deliver business benefits have been disappointed with the outcomes of those projects.

Organizations can fall into the trap of thinking that a powerful, promising new technology is inherently superior, and that simply applying the new technology will be sufficient to generate improved outcomes. This is, of course, not true. Like any technology, GenAI has use cases where it is more and less valuable. In addition, the potential benefit gained from using new technologies varies based on the existing state of the system or process in question. Figure 4 illustrates the possible outcomes of a modernization effort. Making changes to existing working systems and processes has the least potential for good outcomes.

Figure 4: Possible Outcomes of Modernization Efforts


Possible Outcomes of Modernization Efforts
Delight
Current Baseline
Disappointment
Best outcome: Nobody can tell, and you spent a bunch of money.
Changing something that works today
Worst outcome: You broke it, and you spent more money than you said you would.
Best outcome: It works, and you deliver a significant return on investment.
Fixing something broken
Worst outcome: You broke it more.
Best outcome: It works, and you deliver a significant return on investment.
Delivering new things
Worst outcome: Nobody can tell, and you spent a bunch of money.

Source: Gartner
768465_C

If you are being directed to find ways to use GenAI to deliver value, remember that it is better to focus on fixing existing problems or delivering new capabilities than to disrupt working solutions. Review your existing problems and challenges — not just technical, but business — and evaluate whether GenAI would be a better solution than existing options. Note that "better" can be measured in several ways:

- Improved business outcomes, such as increased customer satisfaction, profitability or speed to market
- Easier implementation or lower cost to implement

- Lower cost to operate
- Greater flexibility to make changes in the future
- New capabilities that can solve previously unsolvable problems

In cases where GenAI offers a better option for achieving meaningful benefits, make sure that you have specific objectives and key results (OKRs) before beginning work.

Related Research

- Assessing How Generative AI Can Improve Developer Experience
- Quick Answer: Should Software Engineering Teams Use ChatGPT to Generate Code?

Continue Learning to Keep Pace as GenAI Evolves

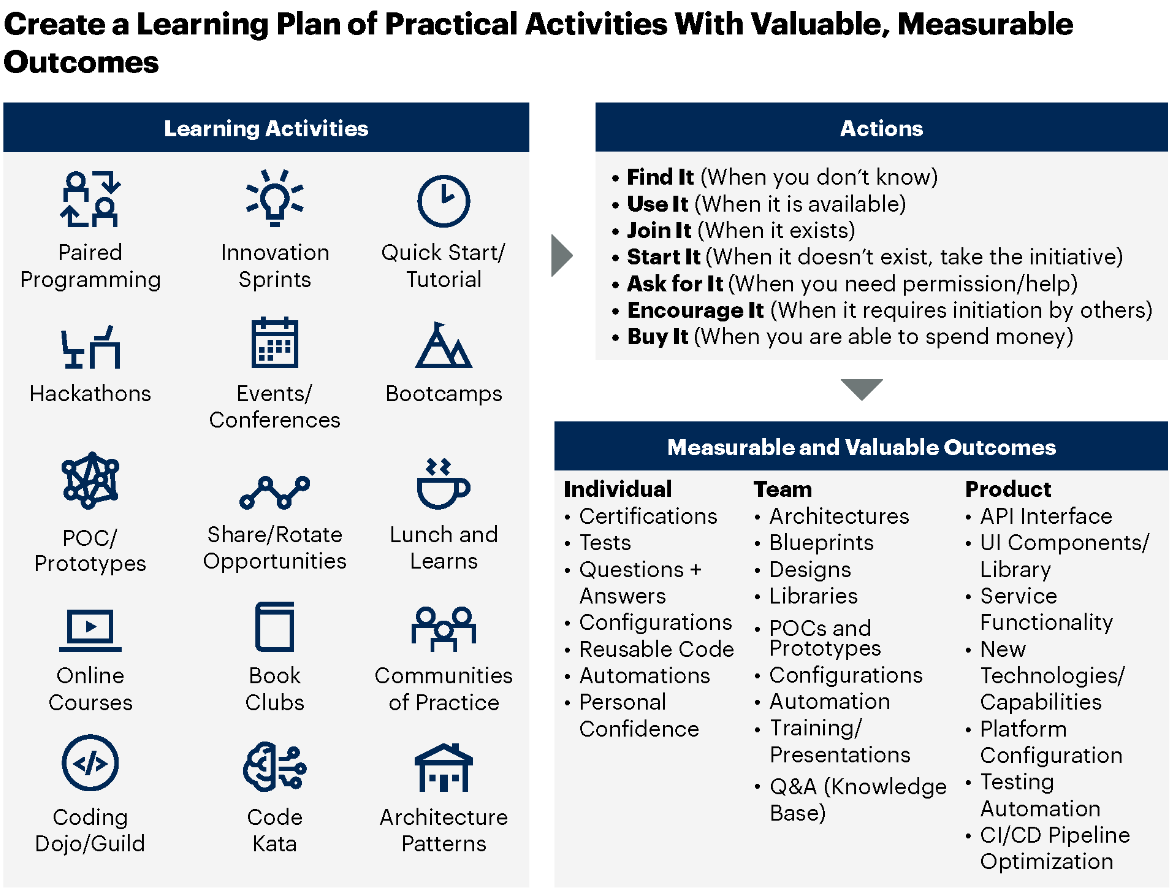
The GenAI ecosystem offerings are in flux, with significant changes occurring on a monthly or even a weekly basis. Examples include:

- Large language models, such as OpenAI's GPT-4, Google's Lambda and PaLM 2, and Meta's Llama 2
- Public offerings, such as Google's Bard, Microsoft's Bing Chat, Midjourney, and OpenAI's ChatGPT and DALL-E 2.

The ecosystem of tools and vendors is also expanding rapidly. In the first quarter of 2023, more than $1.7 billion in venture capital funding was allocated to startups, and a further $10 billion was announced but has not yet closed. [1] And, of course, the tools to support GenAI are evolving as well, as are the implementations of GenAI-powered functionality into applications and tools such as Google Docs and Grammarly.

This rapid evolution means you must dedicate time to finding and consuming information on GenAI. To start, use the sources and communities that you identified when learning how GenAI works. Then, identify other resources that cover emerging trends, and track new releases to provide current information. Create a plan for ongoing learning, and build time into your schedule to both maintain your understanding of the state of the art and sharpen your skills. Figure 5 shows Gartner's formula for creating a learning plan: Determine learning activities that map to actions leading to measurable, valuable outcomes.

Figure 5: Create a Learning Plan of Practical Activities With Valuable, Measurable Outcomes



Create a Learning Plan of Practical Activities With Valuable, Measurable Outcomes

Learning Activities

- Paired Programming
- Innovation Sprints
- Quick Start/ Tutorial
- Hackathons
- Events/ Conferences
- Bootcamps
- POC/ Prototypes
- Share/Rotate Opportunities
- Lunch and Learns
- Online Courses
- Book Clubs
- Communities of Practice
- Coding Dojo/Guild
- Code Kata
- Architecture Patterns

Actions

- **Find It** (When you don't know)
- **Use It** (When it is available)
- **Join It** (When it exists)
- **Start It** (When it doesn't exist, take the initiative)
- **Ask for It** (When you need permission/help)
- **Encourage It** (When it requires initiation by others)
- **Buy It** (When you are able to spend money)

Measurable and Valuable Outcomes

Individual

- Certifications
- Tests
- Questions + Answers
- Configurations
- Reusable Code
- Automations
- Personal Confidence

Team

- Architectures
- Blueprints
- Designs
- Libraries
- POCs and Prototypes
- Configurations
- Automation
- Training/ Presentations
- Q&A (Knowledge Base)

Product

- API Interface
- UI Components/ Library
- Service Functionality
- New Technologies/ Capabilities
- Platform Configuration
- Testing Automation
- CI/CD Pipeline Optimization

Source: Gartner
796446_C

Related Research

- Proven Practices to Enhance Application Architecture and Software Development Skills
- Develop Your Technical Skills Using Online Learning Platforms
- Community of Practice Essentials

Correct Implementation of Automation Will Become a Prerequisite to Success

Back to top

Hyperautomation is a business-driven, disciplined approach that organizations use to rapidly identify, vet and automate as many business and IT processes as possible. It involves the orchestrated use of multiple technologies, tools or platforms, including:

- AI/machine learning (ML)
- Robotic process automation (RPA)
- Business process management (BPM) and intelligent business process management suites (iBPMSs)
- Integration platform as a service (iPaaS)
- Low-code/no-code tools
- Packaged software
- Other types of decision, process and task automation tools

In recent years, organizations across diverse industries have been investing in automation to optimize operations, streamline repetitive processes, improve agility and responsiveness to demand, and deliver better customer experiences. However, organizations embarking on the automation journey often find themselves inundated with challenges that prevent them from getting the automation project off the ground, or from achieving the cost savings and benefits anticipated. These barriers to successful implementation of automation may include the following:

- **Heterogeneous vendor solutions with varying maturity levels** inundate organizations with complex and sometimes conflicting technology choices to meet their automation needs.
- **Siloed organizational functions** can make it difficult for automation initiatives to engage all relevant stakeholders, overcome cultural resistance, and balance automation priorities and desired business outcomes with cross-functional needs.
- **Failure to optimize existing processes** leads to overly complicated and costly implementations that try to automate inherently inefficient and flawed processes.

To overcome these barriers to implementing automation, architects and developers should follow these planning considerations:

- Prioritize and select automation use cases based on efficiency, suitability and business outcome.
- Leverage APIs and events to enable integration and automation.
- Use decision processes to select the right automation tools.
- Improve developer experience and productivity to accelerate automation delivery.

Planning Considerations

Prioritize and Select Automation Use Cases Based on Efficiency, Suitability and Business Outcome

This planning consideration is a critical prerequisite to the success of any automation initiative. Not all business processes are suitable for automation. Processes that are not well-documented or well-understood — or ones that involve manual intervention, insular knowledge, too many variants in rules or nondeterministic outcomes — are generally poor candidates for automation.

It's important to align the scope of your automation initiatives to your organization's hyperautomation maturity level. Focus on efficiency, suitability and business outcomes, but don't tackle more than you can handle (see Quick Answer: Use Gartner's Hyperautomation Maturity Model). For instance:

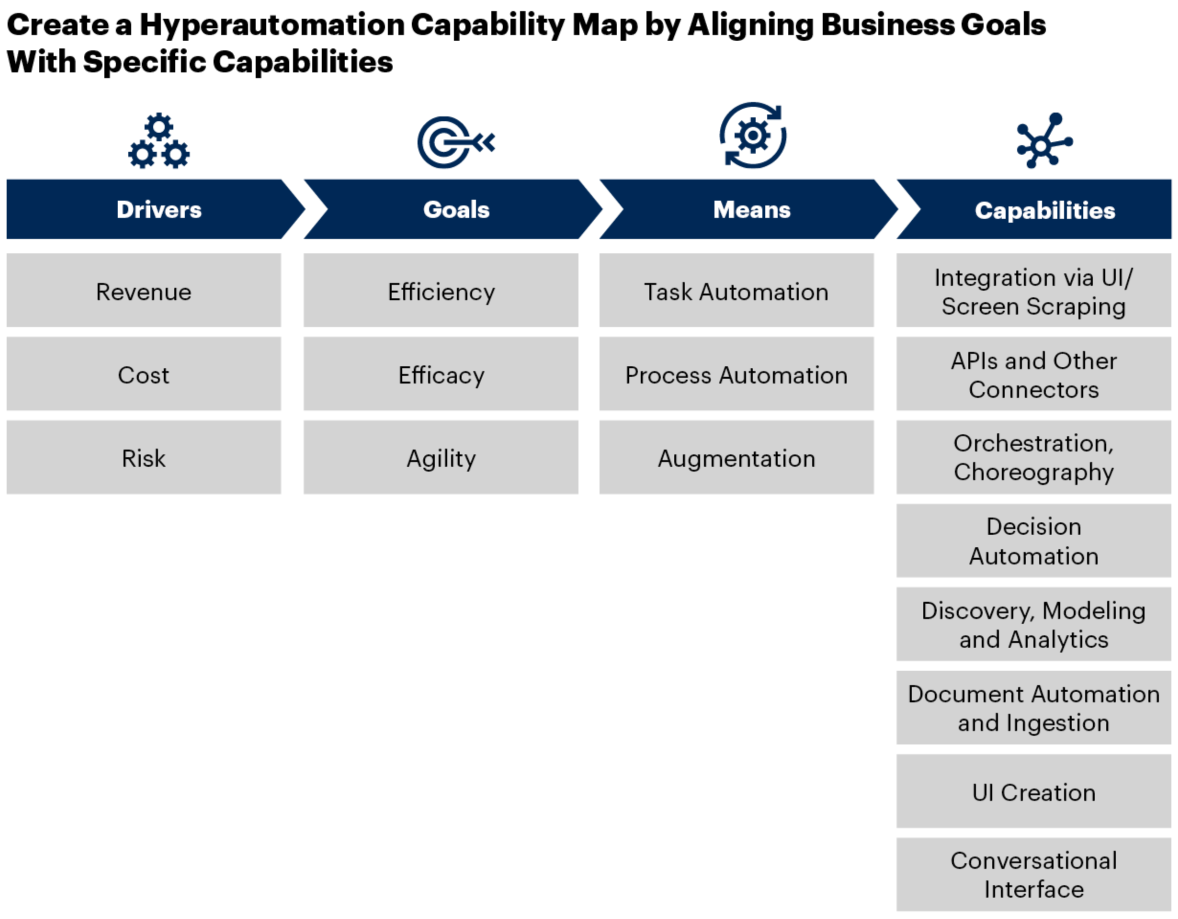
- At Maturity Level 1, your goal is to target quick time to value and quick wins.
- At Maturity Level 2, you are ready to establish a more scalable automation strategy.
- At Level 3, you can tackle front- and back-office operational efficiency as part of an automation transformation initiative.

As an architect or developer responsible for implementing automation, you will need to juggle a number of business and technical factors to select the right automation use cases (see Figure 6). Thus, you should:

- Explore automation opportunities and use cases, and analyze how they can help you achieve your target business outcome, such as revenue growth, cost optimization or risk mitigation.

- Prioritize automation use cases with metrics such as efficiency, efficacy and agility, which have a direct association with or contribution to business drivers. For example, can an automated process reduce human errors and ultimately improve customer experience and overall satisfaction?
- Determine and select the right approach to automation. Ask the following questions:
  - Should we use task automation to automate simple, repetitive or routine tasks?
  - Should we implement process automation to orchestrate and automate long-running processes?
  - Or is augmentation the right approach to deliver self-service capabilities to customers and situational, sensitive and timely suggestions to customer service agents?
- Identify the available technologies and tools that can deliver the capability needed by the automation use cases.

Figure 6: Create a Hyperautomation Capability Map by Aligning Business Goals With Specific Capabilities



Create a Hyperautomation Capability Map by Aligning Business Goals With Specific Capabilities

Drivers	Goals	Means	Capabilities
Revenue	Efficiency	Task Automation	Integration via UI/ Screen Scraping
Cost	Efficacy	Process Automation	APIs and Other Connectors
Risk	Agility	Augmentation	Orchestration, Choreography
			Decision Automation
			Discovery, Modeling and Analytics
			Document Automation and Ingestion
			UI Creation
			Conversational Interface

Source: Gartner

734894_C

Related Research

- Beyond RPA: Build Your Hyperautomation Technology Portfolio
- Communicate Hyperautomation Product and Service Value Using a Maturity Diagnostic Model
- Overcome Barriers to Hyperautomation to Improve Your Value Proposition

Leverage APIs and Events to Enable Integration and Automation

APIs and events have become increasingly valuable in modern integration and automation initiatives. APIs allow different services, applications and systems to communicate and exchange data. They are often used to implement automation tasks by integrating multiple services and applications. Events, on the other hand, enable automation by triggering activities based on specific notifications or changes in a system, data repository or application.

Together, APIs and events form the substrate upon which automation is composed, ranging from task automation and augmentation to process automation and orchestration. Organizations need to continuously manage, maintain and enhance their APIs and events to ensure the success of automation initiatives.

Related Research

- Maturity Model for Event Driven Architecture
- Choosing Event Brokers: The Foundation of Your Event-Driven Architecture
- Essential Patterns for Event-Driven and Streaming Architectures
- How to Design Great APIs
- How to Deliver Sustainable APIs
- Choosing an API Format: REST Using OpenAPI Specification, GraphQL, gRPC or AsyncAPI

Use Decision Processes to Select the Right Automation Tools

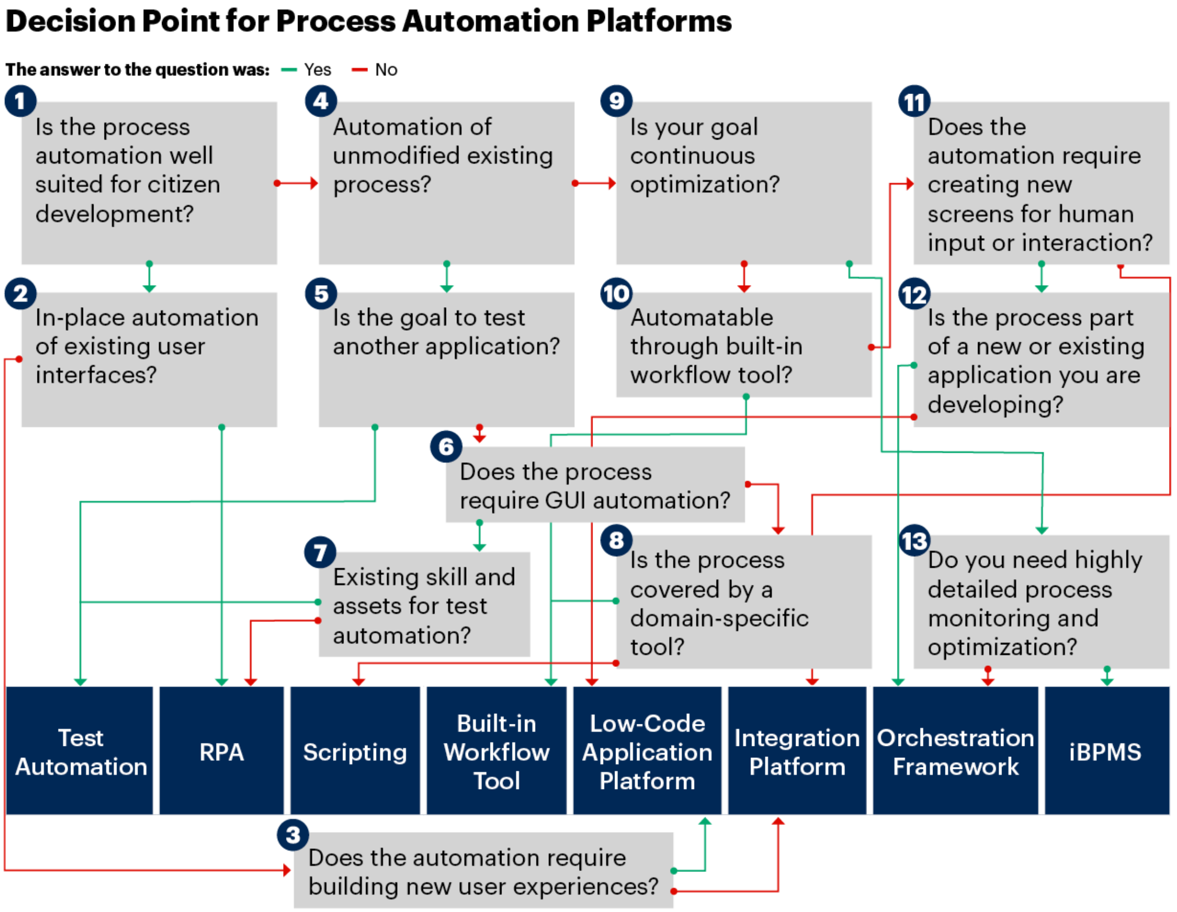
Automation tools that drive many automation initiatives are diverse and multifaceted. They span heterogenous technology markets and target a wide range of developer personas, from citizen developers to subject matter experts (SMEs) specializing in specific technology domains. Common automation tools may include:

- RPA
- Chatbots and conversational platforms
- Natural language understanding (NLU) and natural language processing (NLP) platforms
- Test automation
- Built-in workflow engines

- Low-code application platforms (LCAPs)
- Orchestration frameworks
- Integration platforms
- iBPMSs
- Scripting

Automation architects and developers should use a decision tree (see Figure 7) to select the right combination of tools based on their capabilities, applicable problem space, maturity, and alignment with use cases and developer personas. For instance, RPA, LCAPs and low-code-oriented iPaaS are suitable for less-experienced citizen developers and integrations. Orchestration frameworks are best for implementing long-running, end-to-end business processes, but often require skilled application developers.

Figure 7: Decision Point for Process Automation Platforms


Decision Point for Process Automation Platforms
The answer to the question was: — Yes — No
1 Is the process automation well suited for citizen development?
4 Automation of unmodified existing process?
9 Is your goal continuous optimization?
11 Does the automation require creating new screens for human input or interaction?
2 In-place automation of existing user interfaces?
5 Is the goal to test another application?
10 Automatable through built-in workflow tool?
12 Is the process part of a new or existing application you are developing?
6 Does the process require GUI automation?
7 Existing skill and assets for test automation?
8 Is the process covered by a domain-specific tool?
13 Do you need highly detailed process monitoring and optimization?
Test Automation
RPA
Scripting
Built-in Workflow Tool
Low-Code Application Platform
Integration Platform
Orchestration Framework
iBPMS
3 Does the automation require building new user experiences?

Source: Gartner

762474_C

Gartner

Related Research

- Decision Point for Process Automation Platforms
- Competitive Landscape: Business Process Automation
- Essential Patterns for Data-, Event- and Application-Centric Integration and Composition
- Essential Design Patterns for Enterprise Application Automation and Extension

Improve Developer Experience and Productivity to Accelerate Automation Delivery

Automation delivery is a complicated software engineering endeavor that often requires:

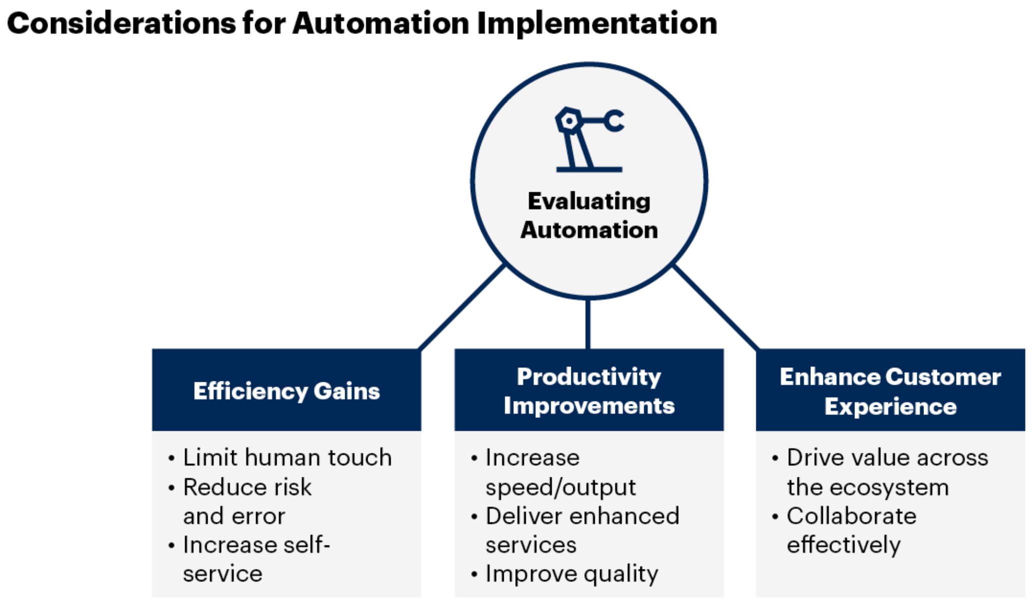
- Multidisciplinary teams

- Cross-organizational collaboration and governance
- A diverse set of technologies, tools, platforms and services managed and operated by different business units across hybrid and multicloud environments

Sometimes, these automation initiatives progress in parallel within an organization, demanding significant effort in coordination, collaboration and oversight. These challenges can place undue burdens on developers and members of the delivery team. Any added manual process or repetitive task in the DevOps cycle can seriously impede developer productivity.

Automation delivery is, in itself, a complex process that can benefit from automation, particularly in areas that boost DevOps process efficiency and improve developer experience and productivity (see Figure 8).

Figure 8: Considerations for Automation Implementation


Considerations for Automation Implementation

Evaluating Automation

Efficiency Gains
- Limit human touch
- Reduce risk and error
- Increase self-service

Productivity Improvements
- Increase speed/output
- Deliver enhanced services
- Improve quality

Enhance Customer Experience
- Drive value across the ecosystem
- Collaborate effectively

Source: Gartner
779820_C

The more you can streamline and improve your automation delivery process, the faster you will be able to implement and deliver automation to your organization at scale.

Related Research

- Benchmarks for Technologies That Enhance Developer Experience and Productivity

- Automate the Application Delivery Value Stream
- Keys to DevOps Success

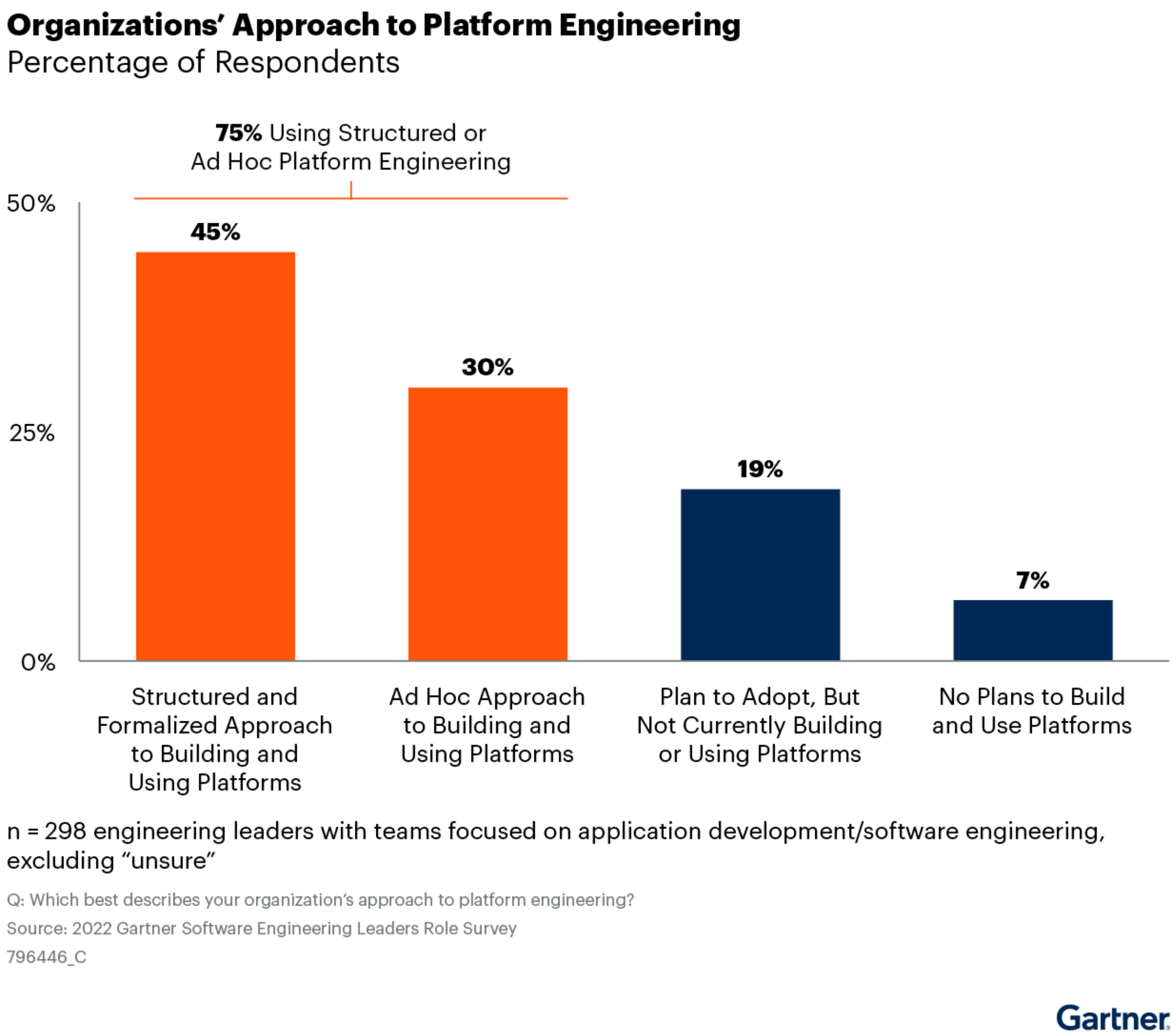
Platform Engineering Will Redefine the Boundaries of Architecture

Back to top

Platform engineering is the discipline of building and operating self-service internal developer platforms for software delivery and life cycle management. Platform engineering aims to optimize the developer experience and accelerate product teams' delivery of customer value.

The 2022 Gartner Software Engineering Leaders Role Survey found that 45% of respondents use a formalized approach to platform engineering, while 30% use an ad hoc approach (see Figure 9). [2] This level of adoption, with such a high percentage resorting to an ad hoc approach, means organizations need to prioritize formalizing their approach or risk underdelivering on this critical initiative.

Figure 9: Organizations' Approach to Platform Engineering



n = 298 engineering leaders with teams focused on application development/software engineering, excluding "unsure"

Q: Which best describes your organization's approach to platform engineering?

Source: 2022 Gartner Software Engineering Leaders Role Survey

796446_C

Platform engineering was also a trend in last year's Planning Guide, and adoption continues to accelerate. This acceleration is outstripping our previous expectations. Gartner chose platform engineering as one of its Top Strategic Technology Trends for 2023. This research, published in October 2022, assumed that "by 2026, 80% of software engineering organizations will establish platform teams as internal providers of reusable services, components and tools for application delivery." At the current rate of change, software engineering organizations will overshoot that percentage, and sooner.

Key benefits of adopting platform engineering include:

- Making it easier for developers to consistently meet security, compliance and architectural requirements by building policies and guidelines into the platform.

- Reducing the cognitive load on developers by providing a curated set of capabilities and tools that work together, accelerating customer value delivery. This approach improves developer experience by removing the toil of developer tool configuration and maintenance.
- Improving the developer experience by reducing or eliminating the duplicative, non-value-adding work required to create foundational components.
- Enabling delivery of the different application architectures required to meet ever-evolving business goals, without unduly increasing the complexity of operating environments.

To optimize platform engineering, application platform and integration architects should follow these planning considerations for 2024:

- Prepare for platform engineering by transitioning to product-centric DevOps delivery.
- Take a broader perspective on developer experience to maximize productivity.
- Establish a dedicated and cross-functional platform team to scale and match maturity.

These activities will redefine your architecture across multiple perspectives — organization, tooling and people. In the same way that agile, DevOps and value streams have transformed working dynamics, platform engineering will further empower teams and accelerate their ability to deliver value.

Planning Considerations

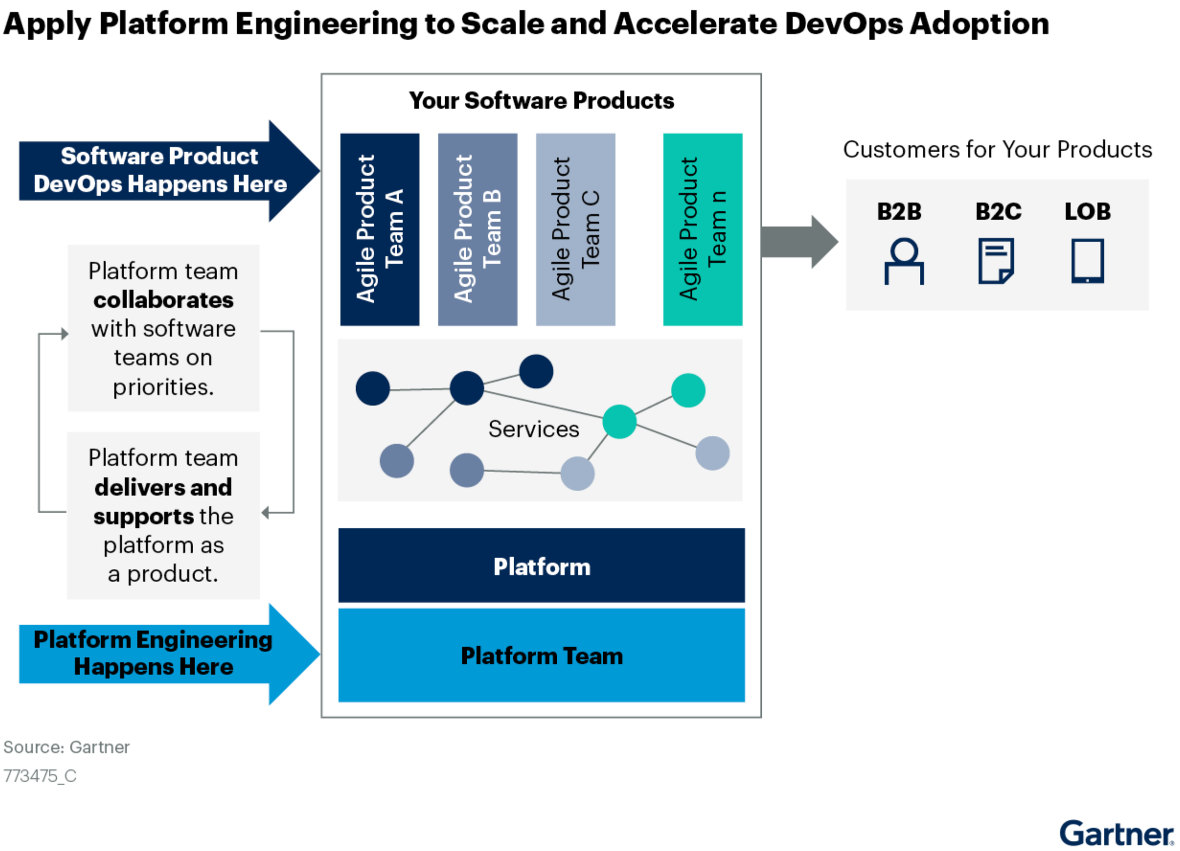
Prepare for Platform Engineering by Transitioning to Product-Centric DevOps Delivery

This prerequisite to successful platform engineering not only redefines the boundaries of architecture but also transforms the software delivery organization. This planning consideration is about what kind of organization is necessary for successful self-service consumption of platform technologies.

Traditional projects strive to deliver all their intended value at the end, after all the work has been done and the team has been disbanded. By contrast, each incremental delivery provides value using an agile product approach. A successful DevOps transformation will require organizations to switch from a traditional project delivery model by adopting organizational structures, processes and practices that support an agile product delivery approach.

Figure 10 illustrates the platform team and its place within the organization. At the heart of the organization, shown as columns in the figure, are the product teams. Also known as "value-stream-aligned teams," product teams perform work that is directly valuable to the business. The platform team interfaces with each product team, providing enabling technologies that allow each product team to produce more and better work. Thus, the platform team serves as a force multiplier.

Figure 10: Apply Platform Engineering to Scale and Accelerate DevOps Adoption


Apply Platform Engineering to Scale and Accelerate DevOps Adoption
Software Product DevOps Happens Here
Platform team **collaborates** with software teams on priorities.
Platform team **delivers and supports** the platform as a product.
Platform Engineering Happens Here
Your Software Products
Agile Product Team A
Agile Product Team B
Agile Product Team C
Agile Product Team n
Services
Platform
Platform Team
Customers for Your Products
B2B
B2C
LOB
Gartner

Source: Gartner
773475_C

According to Conway's Law, software architecture and organization structures are tightly coupled. "Don't ship the org chart" is Stephen Sinofsky's reformulation of Conway's Law. An organization intentionally changing its team structure to mirror products is referred to as an "inverse Conway maneuver." Performing this maneuver breaks silos that constrain the ability of individual teams to create value. Product teams supported by platform teams reduce dependencies and coordination efforts between teams.

Cost savings are unlikely. Platform engineering should improve productivity, cycle time and speed to market, among other important metrics. Expect a good return on investment, but not less investment overall.

Related Research

- Keys to DevOps Success
- Essential Skills for Agile Development
- Use Platform Engineering to Scale and Accelerate DevOps Adoption
- Organize for Agility With Team Topologies
- Product Owner Essentials

Take a Broader Perspective on Developer Experience to Maximize Productivity

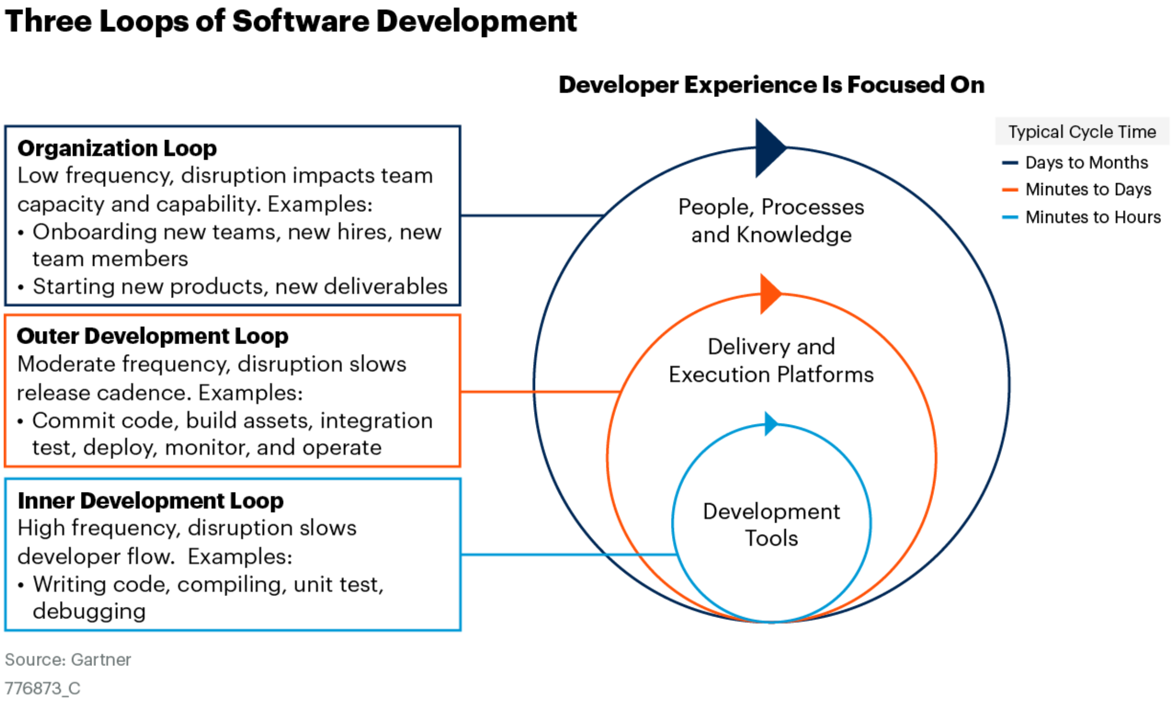
This planning consideration is about what kind of experience developers need from a platform engineering initiative to maximize the benefits of the investment and justify the enterprise disruption.

The Team Topologies approach introduced the concept of the "thinnest viable platform" (TVP) — aka "minimum viable platform" — to underline the importance of not overengineering the platform. (For more information, see Organize for Agility With Team Topologies.) The platform should be the minimum set of information, abstractions, tools and services required, to help reduce the cognitive load on stream-aligned product teams.

Applying the TVP principle means looking for friction in the developer experience and identifying platform capabilities that can remove or ease it. You should not start by designing "greenfield" shared platform technologies. Begin by assessing the level of duplicated technology or unmet need across product teams. Doing so will identify the extent of the challenge of consolidating these technologies into a smaller set of supported technologies in your shared platform.

When you are prioritizing tooling and platform enhancements, it is important to consider the impact of any improvement on the developer experience (and the value delivery of developers). In recent years, the idea of an inner and an outer loop of development has become a popular way of mapping the development flow. When you extend developer experience to *all* the interaction points a developer has, you must also introduce a third loop. We refer to this loop as the "organization loop," as shown in Figure 11. The organization loop incorporates lower-cadence but potentially high-impact activities, such as onboarding, information discovery, new product development initiation and cross-team collaboration.

Figure 11: Three Loops of Software Development


Three Loops of Software Development
Developer Experience Is Focused On
Organization Loop
Low frequency, disruption impacts team capacity and capability. Examples:
• Onboarding new teams, new hires, new team members
• Starting new products, new deliverables
Outer Development Loop
Moderate frequency, disruption slows release cadence. Examples:
• Commit code, build assets, integration test, deploy, monitor, and operate
Inner Development Loop
High frequency, disruption slows developer flow. Examples:
• Writing code, compiling, unit test, debugging
People, Processes and Knowledge
Delivery and Execution Platforms
Development Tools
Typical Cycle Time
Days to Months
Minutes to Days
Minutes to Hours

Source: Gartner
776873_C

Gartner clients often focus narrowly on the inner and outer development loops, missing the lower-frequency organization loop. Everything about your organization has a direct effect on the developer experience — from your people and culture, corporate processes, workload management, rewards, and recognition to your hardware strategy and work environment. Ensure that responsibility for the overall developer experience is clear, but do not assume that the platform team will be able to address all aspects of developer experience.

In addition, make sure that your platform capabilities match your organization's current (or intended) delivery model, rather than forcing product teams to adapt to bleeding-edge solutions. Gartner frequently speaks to organizations that have established new, containerized cloud-native platforms, but are struggling to get product teams to use them. The problem is not that the organization has chosen the wrong platform. It is that this choice was made without the participation of product teams.

Product teams deliver business value by getting features into production with low effort, risk and cycle times. Giving them self-service access to a Kubernetes cluster solves a small fraction of the problem. Changing platforms is a significant effort for product teams, and they will resist it unless that platform makes things easier for them. Your imagination about what developers want is a poor substitute for talking to them.

Related Research

- Adopt Platform Engineering to Improve the Developer Experience
- Automate the Application Delivery Value Stream
- Innovation Insight for Internal Developer Portals

Establish a Dedicated and Cross-Functional Platform Team to Scale and Match Maturity

Setting up dedicated platform teams is key to organizations' platform engineering efforts. The alternative is having platform consumers self-manage their platforms, with all the attendant undifferentiated toil.

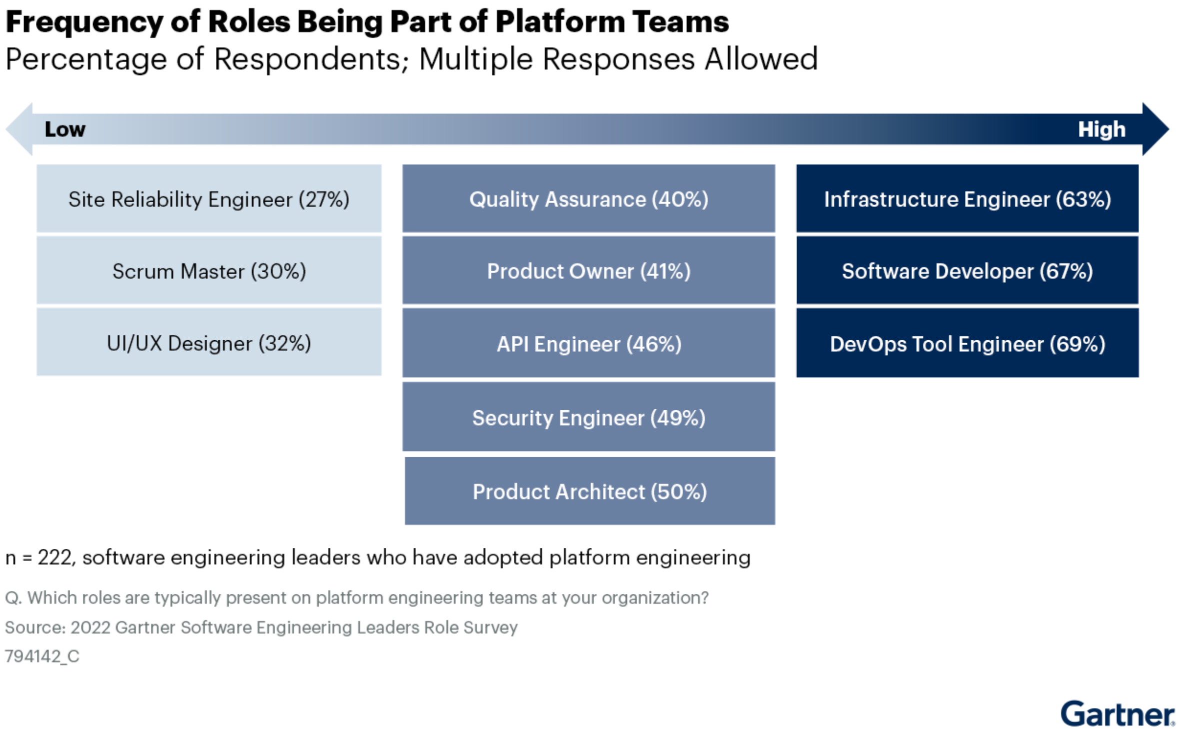
The platform team needs full-time commitment and accountability. This is not a part-time job for I&O or application development professionals to do in their spare time. Your internal customers — the product teams — will come to rely on the platform, which requires dedicated resources and long-term commitment.

You could start with one or two engineers and a product owner whose sole jobs are to deliver the platform. One of the key aspects of effective platform engineering is product management. The product owner's priority is to ensure effective platform product management by listening to and prioritizing demand from platform users.

In addition to the product manager or product owner, your organization will need platform reliability engineers with overall technical responsibility to deliver and operate the platform technologies. These engineers will also build the necessary integrations with the rest of your IT estate. They will work with your product teams to ensure that they use architectures and design patterns that suit the platform.

Respondents to the 2022 Gartner Software Engineering Leaders Role Survey who have platform teams indicate that the team roles vary by organization, but the most common are DevOps tools engineer, software developer and infrastructure engineer. More than 60% of the respondents stated that these three roles are typically part of their platform teams. Other roles frequently part of platform teams include product architect, security engineer and API engineer (see Figure 12). [2]

Figure 12: Frequency of Roles as Part of Platform Teams



Frequency of Roles Being Part of Platform Teams
Percentage of Respondents; Multiple Responses Allowed

Site Reliability Engineer (27%)	Quality Assurance (40%)	Infrastructure Engineer (63%)
Scrum Master (30%)	Product Owner (41%)	Software Developer (67%)
UI/UX Designer (32%)	API Engineer (46%)	DevOps Tool Engineer (69%)
	Security Engineer (49%)	
	Product Architect (50%)	

n = 222, software engineering leaders who have adopted platform engineering

Q. Which roles are typically present on platform engineering teams at your organization?
Source: 2022 Gartner Software Engineering Leaders Role Survey
794142_C

Platform engineering combines the strengths of I&O, security and software development teams into a partnership. I&O experts have operating system experience and know-how. Because they have run infrastructure (though not necessarily platforms), they may also possess specialized knowledge, such as a deep understanding of networking, storage, incident troubleshooting and SLA fulfillment. Software engineers understand the needs of product teams, have the skills to configure software-defined, API-driven infrastructure, and can take responsibility for researching, delivering and integrating platform technologies.

Cautions

However, you need to be aware of the following challenges platform engineering teams face:

- Skills shortages will hinder efforts. You will likely need to learn these skills on the job as the platform grows and evolves.
- You need to stay focused on the business goal, despite the complexity of delivering platform technology. In particular, for core platform teams, remember that you are an enabler, not a value stream. Customers don't pay you for creating this platform. They pay you for the products your product teams build with the platform.

Be ambitious but realistic. Platform engineering will redefine your organization, tooling and people.

Related Research

- Use Platform Engineering to Scale and Accelerate DevOps Adoption
- Benchmarking Data to Help Implement Platform Engineering
- Case Study: Infrastructure Platform Teams for Self-Service Delivery (Politiet)

Failure to Understand Business Drivers Will Thwart Modernization Efforts

Back to top

Application modernization addresses the migration of legacy applications to new architectures or platforms. It includes integrating new capabilities to provide the latest functionality to the business. Modernizing an application can involve several elements, such as:

- Utilizing cloud technology
- Implementing agile methodologies
- Upgrading software and hardware
- Incorporating automation
- Integrating data analytics

While these initiatives can significantly improve a company's capabilities, it is vital to have a thorough understanding of the underlying business drivers pushing for modernization. The conversations we have with clients show that many organizations find their efforts to modernize falling short of expectations. This shortfall is often due to a failure to identify and adequately prioritize the business drivers for modernization.

The Planning Guide's application modernization trend reappears after a year-long gap. While the need to modernize applications has become business as usual (leading to us removing it as a trend last year), we are still observing many organizations struggling to achieve their modernization goals. We also believe the push for GenAI capabilities in applications will lead to further challenges.

Organizations looking for long-term growth and innovation opportunities opt for application modernization as a key initiative. The business drivers need to be well-defined for each application undergoing the modernization cycle. Interview the key stakeholders to understand their needs and requirements. Then, conduct a gap analysis to understand the differences between the current application state and the desired application state. Afterward, develop the business case for modernization.

It's important to clearly understand what you hope to achieve to stay focused and motivated throughout the journey. Without clear business goals, the modernization initiative is bound to get sidetracked, and the journey will ultimately waste time and effort. Apply the following planning considerations as you prepare to modernize the application:

- Set realistic goals without turning them into a "wish list."
- Use technology as an enabler, not as a driver.
- Use continuous modernization to improve upon business goals.

Planning Considerations

Set Realistic Goals Without Turning Them Into a "Wish List"

When organizations are defining business goals for application modernization, they must be realistic to ensure those goals can be achieved within a reasonable time frame. Otherwise, the goals may become more of a wish list than an actionable plan. To define realistic goals, organizations need to consider current application issues, available resources, skill levels and potential obstacles.

By setting achievable goals, development teams can maintain motivation and make progress toward the desired outcomes. For instance, “refactor all the applications to use microservices architecture” is a wish-list item, not a goal. A realistic goal could be to “refactor the customer onboarding application to support weekly releases in the next two years.”

To achieve success, application architects, project managers and business stakeholders should set realistic and achievable goals. It is important to provide specific, feasible and time-bound objectives. Unclear or overly ambitious goals can be confusing. By contrast, too many conservative goals may prevent the organization from realizing the full benefits of modernization. In either case, the result is a failure.

The approach for defining the goals of application modernization should include the following steps:

- **Define goals with a quantifiable number and justification:** What business outcome do you want to achieve, and how can you measure it? For instance:
  - Increase performance by 20% to improve the conversion rate by 15%
  - Scale the application to handle twice the load to reduce the churn rate by 20%
- **Know your application (KYA):** Analyze the application to understand its strengths and weaknesses. This analysis will include the application architecture, the technology stack, the deployment model, the business areas to which this application caters, and the skill set of the development team managing this application. This information will help you set realistic goals for your application.
- **Collect baseline metrics:** Determine the baseline metrics of the application. This information will help measure and define the success criteria for the application once it is modernized. For instance:
  - How many concurrent users are currently supported by the application?
  - What is the service response time?
  - What is the page-rendering time (first load versus subsequent load)?

- **Define the milestones and timelines:** For the defined goals and the application in scope, specify the milestones for the complete timeline. Modernizing an application is a long-term activity requiring periodic checkpoints to measure the intermediate goals.

Related Research

- Align Projects, Products and Outcome Metrics to Business Goals
- Outcome-Driven Metrics for the Digital Era
- Building a Successful Business Case for an Application Modernization Program

Use Technology as an Enabler, Not as a Driver

One of the most common traps organizations fall into is adopting a purely technology-driven approach. This often entails prioritizing the latest technologies and trends without thoroughly evaluating how these changes will help achieve business objectives. Such an approach can be counterproductive and can lead to wasted resources and missed opportunities. As a technical professional, you must look past the excitement of getting to use a new technology. Focus on where and how the technology can deliver business impact that cannot be achieved in simpler ways.

When technology is used as a driver for modernization, a “one-size-fits-all” approach is applied. For instance, one utterly technology-centric approach an organization may choose is deploying all the applications on containers. While this is an effective technique for application modernization, not all applications can benefit from it without considerable changes to their existing architecture. The question to ask is whether this change adds any business value. Moving a monolithic application from a virtual machine to a container does not enable omnichannel support or improve customer experience, thereby failing to satisfy the organization’s business objectives.

The organization must ensure that business stakeholders, enterprise architects and application architects collaborate to succeed with application modernization. A systematic approach to analyzing each application based on technical, cost and business fit is highly recommended.

Related Research

- Using TIME for Application and Product Portfolio Triage: Data From the Field
- How to Assess the Fitness of Your Application Portfolio

- How to Choose the Right Approach for Application Modernization and Cloud Migration

Use Continuous Modernization to Improve Upon Business Goals

A common mistake is to execute the application modernization process as a one-time project. This approach leads to short-term success. The business goals keep on changing as the organization grows. Hence, it is important to ensure that the application modernization is not halted and reworked when required later.

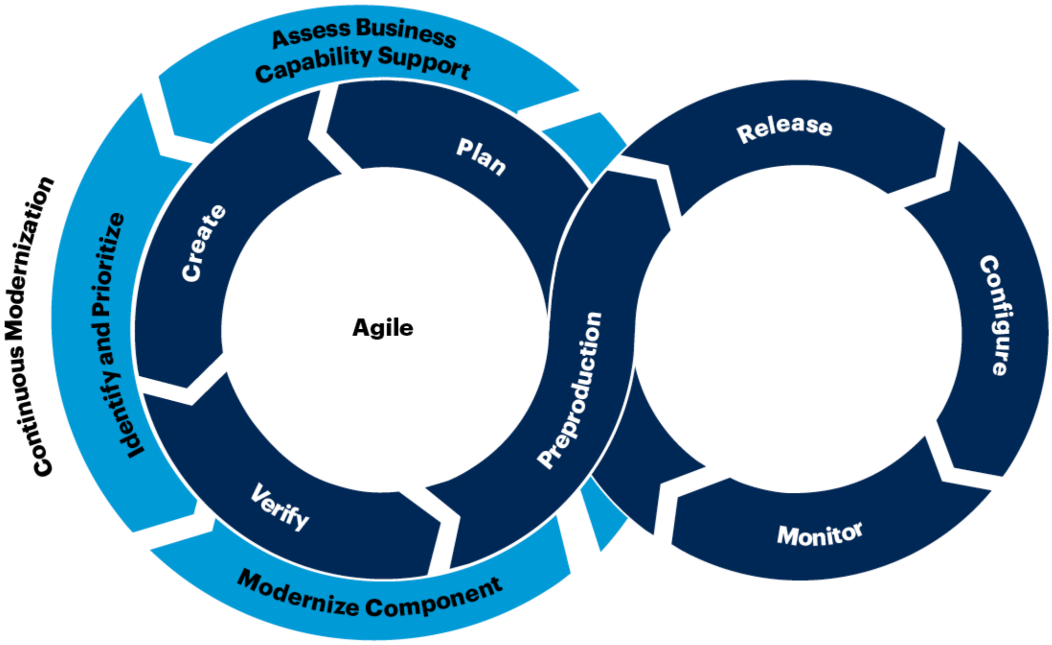
Application architects must be willing to continuously experiment and innovate to meet or exceed the expected benefits of modernization. Achieving business goals like performance improvement often requires multiple iterations. This iterative approach allows flexibility and adaptability to address any challenges that may arise during modernization. We recommend implementing a multistep process, split out across three key stages:

- **Assess and prioritize:** Assess the business capabilities within the application, and identify modernization drivers for each capability. Add these drivers to the backlog of modernization activities.
- **Transform:** Analyze the cause behind the problem, and select an approach to remediate the activity from the backlog. Possible approaches include encapsulating, replatforming, refactoring or replacing application components.
- **Refine and repeat:** Reevaluate the backlog to pick the next activity.

Figure 13 shows the iterative cycle of identifying, prioritizing and removing the obstacles that impede continuous value delivery.

Figure 13: Continuous Modernization Minimizes the Cost, Risk and Impact of Optimizing Legacy Applications

Continuous Modernization Minimizes the Cost, Risk and Impact of Optimizing Legacy Applications



Source: Gartner
772826_C

Gartner

Related Research

- How to Modernize Your Application to Use Cloud-Native Architecture
- Use Continuous Modernization to Optimize Legacy Applications

After a Time of Consolidation, Technology Trends Will Again Diversify Application Architecture

Back to top

The rapid adoption of microservices, containers, Kubernetes, service meshes, serverless technologies, event-driven architecture (EDA) and DevOps significantly increased the complexity of application architecture, forcing organizations to become more pragmatic. Organizations needed to consolidate their technologies, improve their skills, implement governance and simplify processes. Now, however, with better governance, processes and skills in place, organizations are once again beginning to take advantage of technology trends to diversify their architecture.

For example, some organizations are now becoming more sophisticated and effective at implementing cloud-native-application, microservices, event-driven and asynchronous-API architectures. Others are adopting modern front-end application architectures, such as micro front ends. In addition, organizations are better-utilizing various technologies, such as API management, hybrid/multicloud models, Kubernetes and service mesh. They know when to use and when not to use these technologies, and they're also becoming more effective at using them.

These shifts are an evolution of the trends we saw in 2022. The ability to diversify the application architecture allows organizations to make better architectural decisions and technology choices that align with specific application requirements. Ultimately, this type of flexibility allows application architects to better adapt to changing business requirements, improve the experience for developers and architect better applications overall.

Organizations that are most successful at diversifying their architecture tend to take a pragmatic approach to selecting new technologies and architectural paradigms. They make sure that the technologies and architectures they adopt align with their needs. They also ensure that they have the skills required to effectively maintain and support those technologies and architectures.

As such, application architects should apply the following planning considerations as they look to diversify their architecture:

- Adopt agile architecture practices that embrace prototyping.
- Adopt modern technologies based on alignment with distributed architecture concerns.
- Include skills development as a continuous part of adopting new architectures and technologies.

Planning Considerations

Adopt Agile Architecture Practices That Embrace Prototyping

Prototyping is an effective method to successfully adopt architectures and technologies. Prototyping enables you to explore technology capabilities, learn architecture approaches and establish best practices, without introducing risks to production systems and data. It allows you to become more confident in your selection of technologies and architectural approaches before you invest significant time and effort in a full implementation.

Agile concepts, such as development sprints, rapid feedback loops and iterative improvements, will help you be more effective when prototyping. Additionally, agile architecture practices, such as distributed/modular components and decoupled interfaces (e.g., APIs and events), will enable you to prototype specific technologies or parts of the application architecture independent of the entire application.

Application architects who are using prototyping to validate an application architecture approach or a technology selection should consider the following principles:

- **Define a valuable outcome:** Prototyping takes time. Identify and define the outcome that justifies the prototype. For example, the desired outcome may be to confidently select a technology, validate an approach, establish best practices or create a framework.
- **Apply an agile approach:** Use iterative sprints, with feedback loops, to ensure you stay on track to deliver the expected outcome from the prototype. Iterative sprints allow you to fail fast or adjust your prototype as you learn more about a technology or approach.
- **Employ the most complex use case:** Identify the most complex use case to ensure that you test the limits of the technology or establish confidence quickly.
- **Prototype only the basics for the use case:** Distill your (complex) use case down to its most basic requirements. Prototype only the pieces of the architecture, or the features of the technology, required to meet your desired outcome.
- **Consider all aspects of the prototype:** The prototype should not only include the functionality, but also consider things like application state, logical flow, data, communications and integrations.
- **Use productionlike data**: You should avoid connecting the prototype to production systems, but you should use data (copy of production or synthetic) that provides a similar level of complexity.
- **Stress-test the prototype:** Validate the prototype against the maximum stress identified by the application architecture.
- **Use existing platforms and technologies that support prototyping:** When possible, use platforms and technologies to simplify the implementation of the prototype. For example, create a prototype instance of an existing container used to validate production code. As another example, use API tooling to create mock production services for your prototype.

Related Research

- Essential Skills for Agile Development
- Scrum Essentials
- Kanban Adoption Guide

Adopt Modern Technologies Based on Alignment With Architecture Concerns

As organizations learn about new technologies and architecture approaches, they tend to focus on the cool new features or general benefits. They adopt the technologies based on general industry trends or, worse, hype, and then try to make their needs fit the capabilities of the technologies they have adopted.

Working from the technology up to define your application architecture is an anti-pattern that will cause you to compromise application quality, delivery and maintainability. By contrast, starting with the application architecture reduces the risk of introducing dependencies that create friction and slow down productivity.

Successful teams focus on defining the architectures that best solve their current and future needs. Only then do they adopt technologies that clearly align to and improve their ability to meet those objectives. To emulate these successful teams, you should:

- Decouple application architecture from technology selection to maximize flexibility.
- Identify the key architecture concerns to meet your objectives.
- Select technologies that best address your architectural concerns.

Decouple Application Architecture From Technology Selection to Maximize Flexibility

An important purpose of application architecture is to provide the structure, design principles and prescribed patterns that enable you to most effectively address the requirements of the application. Architecture simplifies, and applies consistency to, the implementation and operation of a system. Prior to selecting any type of technology, first identify the key requirements of the application, and use them to choose the appropriate architectural approach.

To select the appropriate architectural approach, you must understand key drivers that define and prioritize architecture objectives. These factors include:

- **Business/organization needs:** Modernization, delivery cadence, innovation, agility, composition, cost reduction, efficiency, training and skills, total cost of ownership (TCO), etc.

- **Application requirements:** User experience, device types, data/resources, functionality, performance, reliability, availability, etc.

- **Architectural priorities:** Agility, adaptability, scalability, modernization, integration, deployment, operations, automation, alignment to enterprise architecture vision and roadmap, etc.

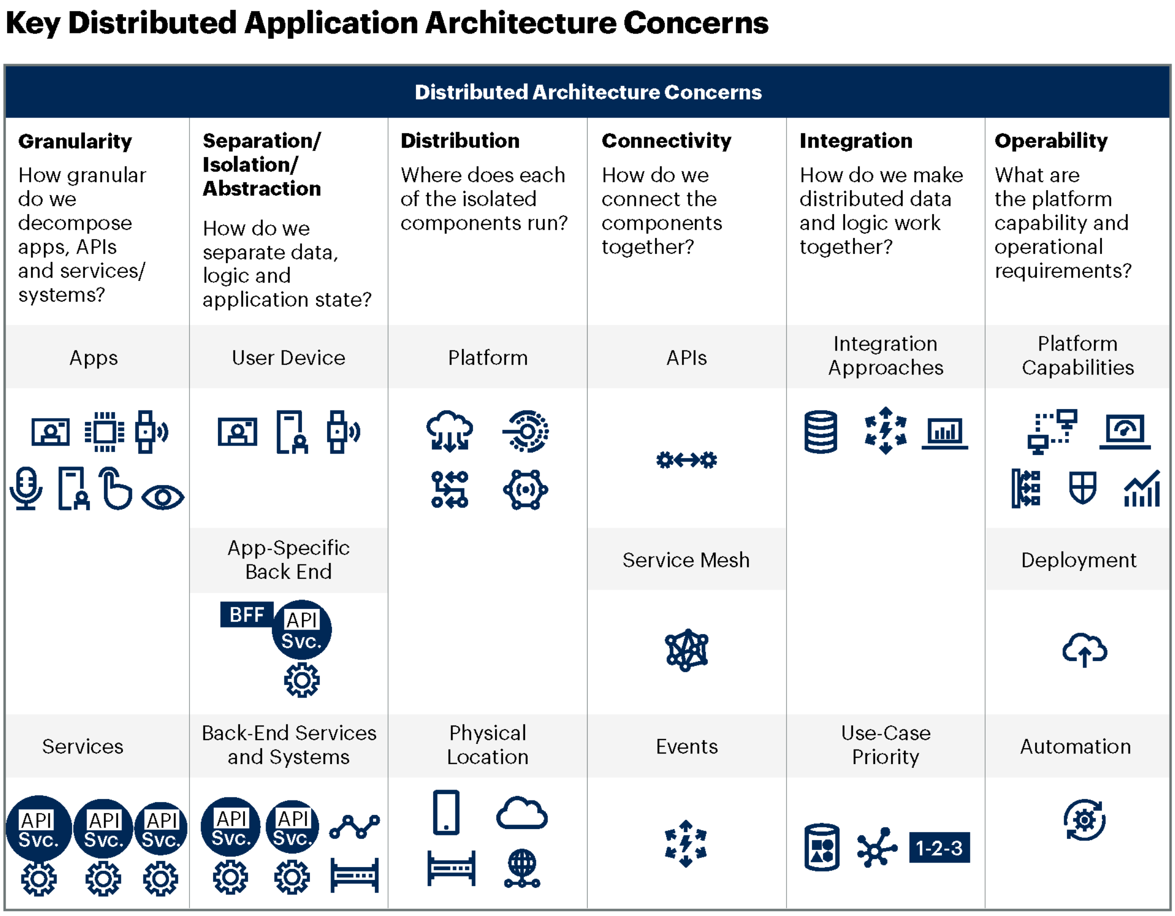
Identify the Key Architecture Concerns to Meet Your Objectives

Diversifying your application architecture enables teams to increase delivery agility, build cloud-native applications, adapt to evolving business needs, adopt new technology advancements and create the next generation of user experiences. A distributed application architecture also enables teams to adopt the optimal technologies to build composite applications from user-facing apps, systems and services that are connected together through API- and event-based communications.

However, diversifying application architecture significantly increases the complexity for everyone — from the architect to the security staff to the operations staff. These roles face the increased challenges of dealing with distributed communications, identity and access management, security, integration, API management (APIM), data design, and diverse runtime technologies. MASA: Create Agile Application Architecture With Apps, APIs and Services describes the challenges in more detail.

Therefore, it is extremely important to identify and define the key architecture concerns to meet your objectives. Doing so allows you to make better architecture decisions and then select the appropriate technologies to best solve your architecture challenges. Figure 14 describes the key architecture concerns that you need to consider when diversifying your application architecture.

Figure 14: Key Concerns in Distributed Application Architecture



Key Distributed Application Architecture Concerns

Distributed Architecture Concerns

Granularity	Separation/ Isolation/ Abstraction	Distribution	Connectivity	Integration	Operability
How granular do we decompose apps, APIs and services/ systems?	How do we separate data, logic and application state?	Where does each of the isolated components run?	How do we connect the components together?	How do we make distributed data and logic work together?	What are the platform capability and operational requirements?
Apps	User Device	Platform	APIs	Integration Approaches	Platform Capabilities
	App-Specific Back End		Service Mesh		Deployment
Services	Back-End Services and Systems	Physical Location	Events	Use-Case Priority	Automation

Source: Gartner

753875_C

Use the business, application and architectural drivers from the previous section to define granularity, separation, distribution, connectivity and integration concerns.

Select Technologies That Best Address Your Architectural Concerns

Once you have identified the key architectural requirements and concerns for your application, you can use those to evaluate which technologies will best address those challenges. Different technologies will support productivity to varying degrees at these architecture hot spots, and productivity is heavily based on utilizing existing skills. Thus, different people will find different technologies more or less productive.

To enable a diversified architecture, use distributed architecture practices to provide the same level of abstraction as an API. Each component in the architecture should not make assumptions or imply dependencies on the implementation details of the other components. When possible, ensure your technologies use standard protocols, specifications and interfaces to avoid proprietary dependencies.

Include Skills Development as a Continuous Part of Adopting New Architectures and Technologies

Application architects and software engineers should focus on foundational skills based on proven practices and technologies. Foundational skills also provide a significant positive impact on the individual, team and product.

Specifically, you should prioritize which skills to acquire or enhance based on each skill's ability to provide the following:

- **Architectural agility:** Does the skill enable delivery of components of the application (such as individual apps, APIs and services) at independent cadences and using the most appropriate technologies?
- **Productivity at scale:** Does the skill allow you to design, architect and implement the application in a more productive way, and does that productivity scale across the team and organization?
- **Innovation:** Does the skill allow you to be innovative with your architecture and integration designs, use the most appropriate technologies, and apply the most appropriate architectural models, patterns and practices?
- **Improved user/customer experience:** Does the skill improve the UI or make the application more capable and reliable in ways that will either help users be more productive or create unique/engaging digital experiences?

For more information about these types of skills, see:

- Essential Skills for Application Architecture
- Essential Skills for Application and Software Developers

Application architects and software engineers are most successful at gaining skills when they apply practical learning activities to develop both knowledge and experience. These types of activities provide multiple benefits:

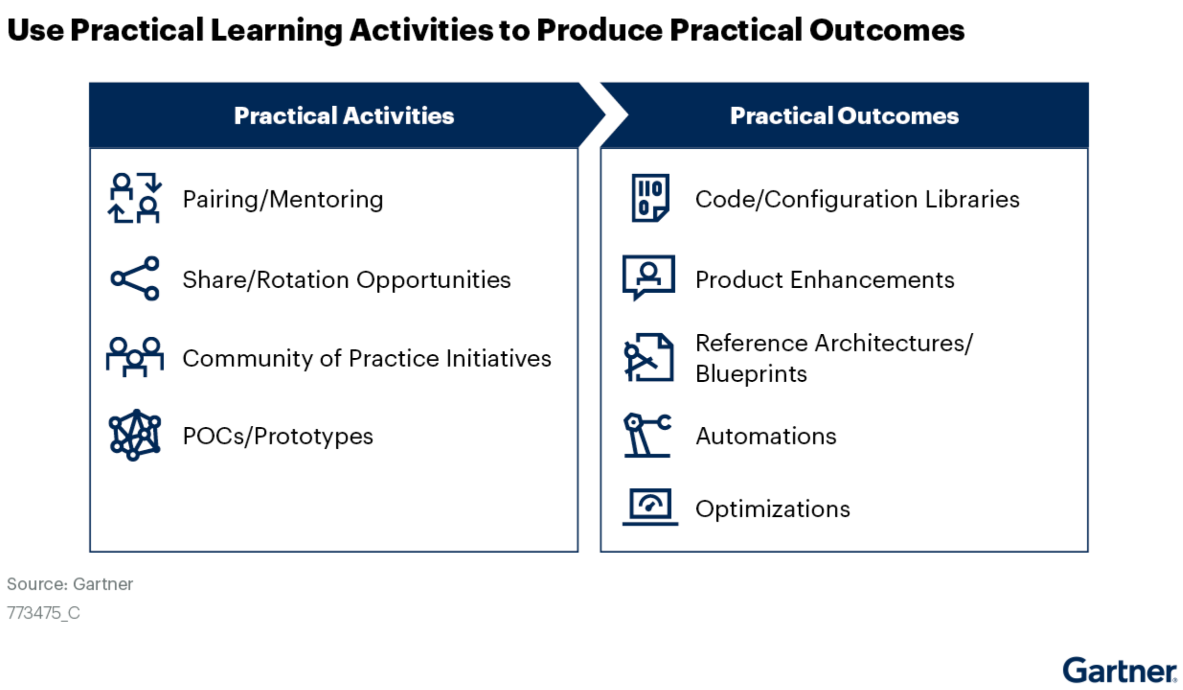
- They enable you to focus on real-world problems.
- They integrate into your normal workflow (so you don't need to set aside learning time).
- Their outcomes are often used in production applications.

Prioritize practical learning activities that enable you to gain both knowledge and experience.

As shown in Figure 15, the following are examples of learning activities that have practical outcomes:

- Pairing/mentoring (in programming, design and architecture)
- Team rotation or sharing
- Communities of practice
- Proofs of concept (POCs) or prototypes

Figure 15: Use Practical Learning Activities to Produce Practical Outcomes



Use Practical Learning Activities to Produce Practical Outcomes

Practical Activities

- Pairing/Mentoring
- Share/Rotation Opportunities
- Community of Practice Initiatives
- POCs/Prototypes

Practical Outcomes

- Code/Configuration Libraries
- Product Enhancements
- Reference Architectures/ Blueprints
- Automations
- Optimizations

Source: Gartner
773475_C

For more information about building application architecture skills, see Proven Practices to Enhance Application Architecture and Software Development Skills.

Architects must learn through multiple activities because the technology and architecture are composed of multiple layers, components, processes and pieces that all have to work together. Depending on the skill and your individual learning style, you can partake in various learning activities, such as tutorials, online courses, bootcamps, conferences, POC or prototype builds, book clubs, and lunch-and-learns. Look for those opportunities within your organization. If they don't exist, take the initiative and create an opportunity.

In most cases, people learn faster when they can work with someone who has greater knowledge and experience than they do. However, that is not always an option. In the absence of someone to learn *from*, you should find others to learn *with*, for example, by:

- Joining developer or architecture special interest groups
- Engaging in paired programming with another developer
- Participating in a community of practice

Evidence

[1] Generative AI Startups Jockey for VC Dollars, PitchBook.

[2] **2022 Gartner Software Engineering Leaders Role Survey:** This survey was conducted to understand how organizations attract, hire and retain software engineering talent; improve and modernize developer skills; improve developer productivity; establish platform engineering teams; create platform teams; and incorporate design into software engineering. The survey was conducted online from November through December 2022. In total, 300 respondents were interviewed from the U.S. Qualifying organizations operated in multiple industries (excluding IT software and public sector) and reported enterprisewide revenue for fiscal year 2021 of at least $250 million or equivalent, with 60% over $1 billion in revenue. Qualified participants were highly involved in managing software engineering/application development teams and the activities they perform. 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.

Document Revision History

2023 Planning Guide for Application Architecture, Integration and Platforms - 10 October 2022

2022 Planning Guide for Application Platforms, Architecture and Integration - 11 October 2021

2021 Planning Guide for Application Platforms, Architecture and Integration - 9 October 2020

2020 Planning Guide for Application Platforms, Architecture and Integration - 7 October 2019

2019 Planning Guide for Application Platforms and Architecture - 5 October 2018

Recommended by the Authors

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

Getting Started With Generative AI in Your Application Architecture

Prompt Engineering With Enterprise Information for LLMs and GenAI

Assessing How Generative AI Can Improve Developer Experience

Keys to DevOps Success

Essential Skills for Agile Development

How to Modernize Your Application to Use Cloud-Native Architecture

Adopt Platform Engineering to Improve the Developer Experience

Essential Skills for Application Architecture

Essential Skills for Application and Software Developers

Proven Practices to Enhance Application Architecture and Software Development Skills