Gartner DevSecOps Maturity Guide

Software security is a top pain point for software engineering leaders who must balance developer experience and business goals. This research provides a five-dimension maturity model framework for securing software development and enables plotting a path toward secure by design.

Overview

Key Findings

- Without an initial state and desired target, software engineering leaders and their teams can feel lost while trying to improve software security maturity.
- Cybersecurity and software engineering teams often have competing priorities, impeding efforts for cross-organization collaboration to transition to DevSecOps.
- Instilling a security-first mindset in software engineering teams is challenging because software engineers require coaching and resources to improve their own security capabilities.

Recommendations

- Identify the initial maturity state for DevSecOps by evaluating the characteristics and scenarios in this maturity model as a comparative framework.
- Foster collaboration between engineering and security teams by establishing a community of practice (CoP) that continuously aligns shared objectives toward improving DevSecOps maturity.
- Empower software teams to achieve higher levels of DevSecOps maturity by establishing enabling teams to provide resources, coaching and mentoring.

Introduction

Software engineering leaders are experiencing more pressure than anytime previously to improve the security of applications without disrupting flow or hindering innovation. The reality of improving security in software engineering is a multifaceted, multiyear program that requires cooperation from all sections of software engineering, cybersecurity, operations and architecture teams.

Security in DevOps (DevSecOps) is a top issue for software engineering leaders. According to the Gartner Software Engineering Survey for 2024, lack of application security skills is considered a pain point by close to two-thirds of software engineering leaders.[1] Mixed advice and an oversaturated security market containing overlapping capabilities ultimately lead to more confusion than clarity.

How can software engineering leadership provide consistently secure software while balancing stakeholder concerns and business goals? This DevSecOps Maturity Model will help guide software engineering leaders to plot a roadmap and what strategies must be employed to guarantee the success of adoption. This maturity model is broken down into five dimensions, each addressing a separate domain for DevSecOps:

1. Security skills and knowledge
2. Developer enablement
3. Secure design and threat assessment
4. Automated security practices
5. Software supply chain security

Analysis

Identify the Initial Maturity State and Desired Targets for DevSecOps

This maturity model is broken into five dimensions, each targeting a particular activity software engineering leadership must improve upon to enhance for DevSecOps. In addition, each dimension has four maturity levels that build on the previous level. Each step forward in maturity must evolve software security practices in each dimension in a tangible, fundamental way. See Table 1 for a summary breakdown.

DevSecOps Maturity Model

	Level 1 (Initial)	Level 2 (Developing)
Security skills and knowledge	Security training program is available but is informal and reactive.	Engineers receive security training regularly and a security knowledge base is created.

Developer enablement	Cybersecurity and Infrastructure and Operations (I&O) perform its tasks manually through work requests.	Cybersecurity testing is included in the SDLC and I&O is standardizing application environments.
Secure design and threat assessment	Secure design review is completed ad hoc and threat modeling is informal.	Threat modeling is completed by cybersecurity teams, outputting security requirements.
Automated security practices	Application security Testing (AST) is used for security testing, but completed manually after application build.	AST is used for automated security testing and is integrated into developer tools, but findings are delivered outside of SDLC tooling.
Software supply chain security	Source control management (SCM) is in use with branch protections.	Software composition analysis (SCA) and secrets management are introduced to reduce risks.

Source: Gartner (August 2024)

Score the Organization’s Initial DevSecOps Maturity

The following sections go into the details of each dimension and what every maturity level entails. DevSecOps maturity is not wholly “Level 1” or “Level 2,” but is a mix that consists of different maturity levels in each dimension.

- Level 1 (Initial)
- Level 2 (Developing)
- Level 3 (Managing)

- Level 4 (Optimizing)

For every dimension, refer to the characteristics table and example scenarios as a comparison framework. Based on these characteristics, compare where the organization currently resides for scoring initial maturity for each dimension. Keep in mind that this model is not prescribing a rigid methodology but a framework to inspire introspection.

Security Skills and Knowledge

Improving security skills and knowledge for software engineering improves security outcomes.[2] Not all secure training is created equally, and there are strategies to pursue that makes learning more engaging and entails higher levels of knowledge retention (see Table 2).

Security Skills and Knowledge Maturity Characteristics

Level 1 (Initial)	Level 2 (Developing)	Level 3 (Managing)
• Security training is available. • Learning is informal and initiated reactively.	• Security training is available. • Secure software guidance is offered through a knowledge base. • Security training is assigned on a regular basis and monitored for completion.	• Security trai • Secure softw knowledge b • A formal secu established t within engine • Assigned sec based trainin
•		

Source: Gartner (August 2024)

Security Skills and Knowledge Example Scenarios

- Level 1 (Initial): When not given an official answer, software engineers who are lost or confused will search the internet for guidance on specific problems. That reduces their productive time and could lead to copy-and-pasted code. It is important for organizations to offer security training for software engineers. At level one, learning is available but not directly assigned and training completion is unmonitored. Learning is also informal and initiated reactively to security incidents.

- Level 2 (Developing): To have oversight, software engineers begin receiving assigned security training on a regular basis (annually or biannually). This grants engineering a standard baseline for training. However, video- or presentation-based security training is not contextual to their unique problems and is easily forgotten after months. Therefore, a software security knowledge base is created for reference that is contextual to the organization's unique concerns and requirements.
- Level 3 (Managing): Instead of video or presentation training, engineering is assigned secure coding training in a learning platform that provides hands-on labs with gamified learning elements to improve engagement and knowledge retention. To extend further security expertise, a formal security champion program is implemented for those individuals who make additional effort on training content. It is recommended to read more on these platforms in Develop Your Technical Skills Using Online Learning Platforms.
- Level 4 (Optimizing): Training is now integrated into the SDLC, providing just-in-time, contextual knowledge when software engineers require it. Engineering receives guidance for remediating security issues automatically as part of their workflow. This naturally leads to seeking additional training as needed, including working closely with security champions.

Developer Enablement

Core to developer enablement is reducing dependencies on external teams. Gartner survey data shows that when software engineering staff are more directly involved in activities traditionally owned by cybersecurity, there are better security outcomes.[2] Being directly involved does not necessarily mean that software engineering owns all security concerns. Cross-collaboration with cybersecurity and I&O teams helps prop up software engineers to conduct work on their value stream with little direct dependencies acting as a constraint.

Table 3 lists the developer enablement maturity characteristics at each level (initial, developing, managing and optimizing).

Developer Enablement Maturity Characteristics

Level 1 (Initial)	Level 2 (Developing)	Level 3 (Manag

• The cybersecurity team owns security testing and triage. • I&O teams manually configure application environments.	• The cybersecurity team owns security testing and triage. • Security tooling is integrated into the SDLC. • I&O teams have standardized application environments.	• Security as a ser • Security orchestr • Enginee changes

For information on IDPs, see Market Guide for Internal Developer Portals. All security tooling must offer

Source: Gartner (August 2024)

Developer Enablement Example Scenarios

- Level 1 (Initial): At the initial maturity stage for developer enablement, developers are either highly dependent on cybersecurity and I&O teams for approvals to move software products forward, or security issues are backlogged and insecure apps are deployed. Both scenarios are not ideal and lead to delays, frustrations and increased security risk.
- Level 2 (Developing): Developers now have access to security tooling integrations as part of the SDLC, and issue triage is driven through work management tools, such as Atlassian Jira or Microsoft Azure DevOps. However, developers still depend on cybersecurity for initiating security testing and issue triage. Additionally, I&O defines the infrastructure and identity and access management (IAM) policy manually through work requests.
- Level 3 (Managing): Engineering now relies on security testing as a service for their software products without waiting for cybersecurity teams. IaC is used to automate application environments creation to remove dependency constraints on I&O. Cybersecurity primarily provides guidance and support to software engineering.
- Level 4 (Optimizing): The now-independent developer uses an IDP for their software products through secure-default, pipeline templates. Creating application environments and security controls are fully automated by these templates to reduce cognitive load and dependency constraints. For this step to be highly

mature, it is prudent to strongly consider using platform engineering for software production. For guidance on platform engineering, see How to Start and Scale Your Platform Engineering Team.

Secure Design and Threat Assessment

Software engineering is based on the concept of patterns. However software is sliced, there is a pattern solving problems and delivering value. Hence, there also exists antipatterns in software. These antipatterns can result in innocuous bugs or devastating security vulnerabilities. “Secure by Design” principles are more important than ever, which are now encompassed by the Cybersecurity and Infrastructure Security Agency (CISA).[3] Secure-by-design examples include:

- Establish internal security controls.
- Publish high-level threat models.
- Publish detailed secure SDLC self-attestations.
- Embrace vulnerability transparency.

Use and publicize well-known design patterns for secure coding with 5 Essential Secure Coding Practices for Software Engineers.

Table 4 shows secure design and threat assessment maturity characteristics.

Secure Design and Threat Assessment Maturity Characteristics

Level 1 (Initial)	Level 2 (Developing)	Level 3
• Assessments for secure software design are being conducted. • Secure design assessments are completed ad hoc. • Threat modeling is not a standard activity.	• Assessments for secure software design are being conducted. • Security requirements are created. • Threat modeling is employed during software product inception. • Cybersecurity teams are mostly responsible for design review and threat modeling.	• • • •

•

An Introduction to Threat Modeling Best Practices is recommended to understand more about the threa

Source: Gartner (August 2024)

Secure Design and Threat Assessment Example Scenarios

- Level 1 (Initial): Design review and threat modeling is a vital first step for software security. However, recent developer traits, such as “move fast and break things” run counter to these ideals. This is fantastic for engineers who want the freedom to innovate, but no application design oversight can lead to particularly nasty vulnerabilities. Cybersecurity teams may be enforcing some secure design policies or ad hoc threat modeling, but they are generally left out of the conversation during software product inception.
- Level 2 (Developing): Engineers now must adhere to threat modeling activities, per software product, by the cybersecurity team. Cybersecurity teams are mostly accountable and responsible for this process and they deliver security requirements as an output. Design reviews may simply be manual questionnaires with design documentation provided, but the “break it” developer is impatient, so they may just ignore these checks and move forward without them.
- Level 3 (Managing): Secure-by-design principles are available in the knowledge base for design review and threat modeling, which is now a shared responsibility between cybersecurity and security architects or security champions. Security requirement output is included in the DoD for user stories. For establishing a DoD, see How to Create and Apply a Definition of Done.
- Level 4 (Optimizing): The “break it” developer can begin to “move fast” again as design review and threat modeling are applied uniformly based on risk, using internal security champions and automated tooling with generative AI (GenAI). Threat modeling is even more powerful with threat intelligence feeds provided by cybersecurity teams. Secure requirements included in the DoD are automatically tested during preproduction and release stages.

Automated Security Practices

Gartner survey results for security in software engineering show a 15% improvement in security outcomes when more security activities are automated.[2] Using AST tools can scan applications for known insecure patterns. For more information on vendors in the AST space, refer to the Magic Quadrant for Application Security Testing. Additionally, reference Trends to Guide Security Automation Decisions in Software Engineering for benchmarking data to guide decisions on security activities to automate.

Table 5 lists the maturity characteristics for automated security practices by level (initial, developing, managing and optimizing).

Automated Security Practices Maturity Characteristics

Level 1 (Initial)	Level 2 (Developing)	Level 3 (Mar
• Automated security testing is used. • Security testing is manually completed just before deployment.	• Applications are analyzed with AST tooling. • Security automation is integrated into SDLC tooling. • Automated security test results are delivered outside of SDLC tooling.	• Appli tooli • Initia autor mana • Auto deliv
Initial rollout of automated security scanning must be carefully implemented and monitored. AST toolin is able to improve accuracy. For posture management tooling, see Innovation Insight for Application Sec		

Source: Gartner (August 2024)

Automated Security Practices Example Scenarios

- Level 1 (Initial): Automated security practices at initial maturity include using AST tools, but little else has been done to mature their capabilities beyond manual scans just before software deployment. These tools deliver a baseline of security findings, but will cause frustration for developers who are burdened by its output.

- Level 2 (Developing): At minimum, static application security testing (SAST) is used. Scanning occurs as a result of actions in the change management process (e.g., new pull or merge requests and new build processes). Yet, these reported issues do not make their way directly back to the developer, forcing engineers to use unfamiliar interfaces outside of their desired tooling. This leads to continued frustration for developers who are assigned security findings with little assistance or oversight.
- Level 3 (Managing): Reported issues from automated security testing are now delivered directly to software engineers in their desired tooling. This reduces developer frustration, since they can use their familiar toolsets to receive AST findings, and the cybersecurity team continuously manages and optimizes these tools for accuracy.
- Level 4 (Optimizing): AST tooling is fully optimized and integrated into the developer's day-to-day tasks, and engineers benefit by customizing the scanning rules directly. All AST tools report into a single source of truth to correlate security risk data and orchestrate vulnerability management. Engineers now receive all automated security testing automatically configured by default in their desired pipelines.

Software Supply Chain Security

When broken apart, the majority of code in software originates from third parties, primarily open source. Attacks against the software supply chain are driving increasingly higher annual costs, according to research in the Leader's Guide to Software Supply Chain Security. More attention is required for security in our SDLC processes and tools (see Table 6).

Software Supply Chain Security Maturity Characteristics

Level 1 (Initial)	Level 2 (Developing)	Level 3 (Managing)
• Using source control management (SCM). • Branch protection is present in source control.	• Using SCM. • Source control requires manual code approvals before code merge.	• Using SCM v • Granular rol enforced.

• Applications are manually built and deployed.	• SCA is present. • Secrets scanning is employed. • Dependencies are managed in a trusted artifact repository. • Software builds are partially automated with manual deployment.	• Multifactor a for all SCM, • Prevention c • Requiremen • SBOMs are g • Build and de automated a initialization

During this transition, the build and release process may encounter lengthier timelines or increased bur
schedules carefully while this transition is still in progress. Use the NIST Secure Software Development
Supply-Chain Levels for Software Artifacts (SLSA) framework as a baseline guide for artifact attestation

Source: Gartner (August 2024)

Software Supply Chain Security Example Scenarios

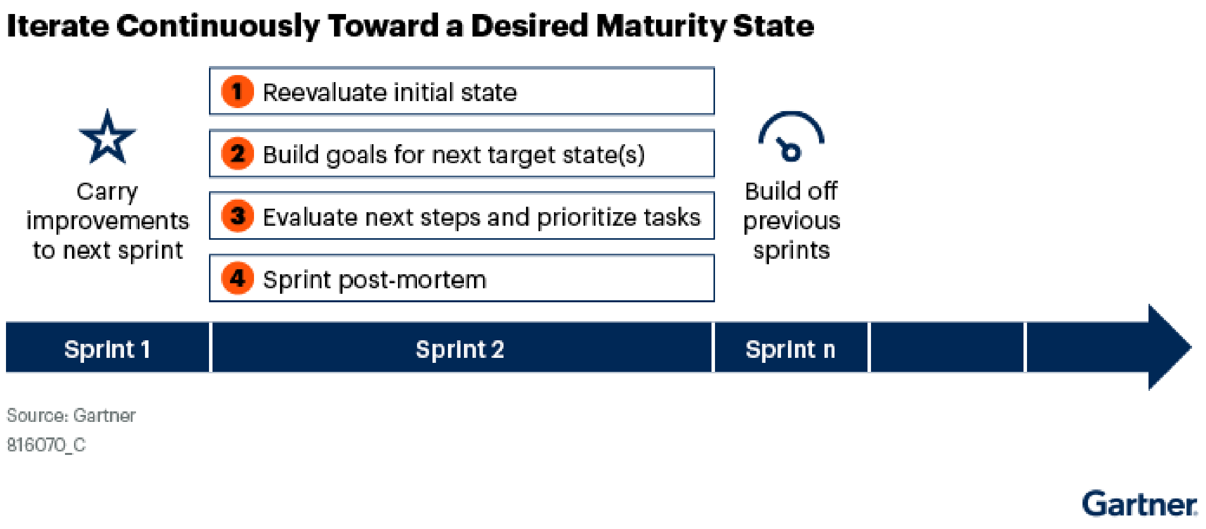
- Level 1 (Initial): Software engineers are expected to use third-party and open-source code to help build features. It makes little sense to reinvent functionality when it can just be imported into the codebase. The developer trusts that the external code will help complete their work. Unfortunately, the trusting developer may be inheriting more than they expect.
- Level 2 (Developing): Because of third-party components that contain known security issues, SCA is deployed as part of the SDLC.[6] Engineers continue to use third-party components and SCA will check for known security issues. Additionally, application secrets are being handled via a secrets management system. This process, however, is still mostly manual, including reviews on code changes before they're allowed to merge. Post merge, the process to build and deploy the application is a largely manual process that is initiated by nondevelopment staff, usually operations or a build-and-release team. Live production access has been removed for software engineers.

- Level 3 (Managing): Strict role-based access controls are applied to all software production tooling (e.g., issue tracking, SCM and build-and-release tooling), including centralized IAM, MFA and requirements for signed commits and artifacts. SBOMs files are required for all third-party components and are being generated for all first-party software.
- Level 4 (Optimizing): Engineers now “trust but verify” their third-party components. Provenance data is created in first-party applications to ensure integrity and traceability throughout the software supply chain. All third-party packages and open-source code are continuously evaluated for operational health and security vulnerabilities using SCA via SBOM processing. All first-party code commits, pipelines and containers are cryptographically signed utilizing a public-key infrastructure (PKI) system.

Continuously Iterate Toward the Desired State

While the goal of getting to the optimizing stage for all dimensions is noble, it is not realistic to invest that much time and resources. Figure 1 shows how to define a continuous, iterative process toward desired maturity states. These iterations should realistically address what the organization can accomplish during a normal sprint time frame (e.g., two to three weeks). Also, see Note 1 for examples of outcome-driven metrics to track progress.

Figure 1: Iterate Continuously Toward a Desired Maturity State


Iterate Continuously Toward a Desired Maturity State
1 Reevaluate initial state
2 Build goals for next target state(s)
3 Evaluate next steps and prioritize tasks
4 Sprint post-mortem
Carry improvements to next sprint
Build off previous sprints
Sprint 1
Sprint 2
Sprint n
Source: Gartner
816070_C
Gartner

Establish a DevSecOps Community of Practice

While software engineering leaders can pursue many of the maturity improvements in this model alone, reaching a desired maturity state requires collaboration with the rest of the technology organization. A CoP can cooperatively drive an implementation strategy between departments.

According to Gartner survey data, there is a 27% improvement to security outcomes when there is a high-level of collaboration between developers and security. However, only 29% of those surveyed say the two groups strongly agree with each other.[2]

The DevSecOps CoP must include software engineering leader sponsorship to launch the program. The sponsor will:

- Define the CoP's objectives, charter and audience.
- Establish the backlog of topics and determine collaboration methods.
- Encourage ongoing participation and engagement with the CoP.
- Monitor the CoP's health.

— S. Pereira and A. Davis, "Flow Engineering," 2024

"There's nothing more fatal to an organization's ability to get things done than a team that can't focus on its goal."

The domains that should be included in creating the DevSecOps CoP are:

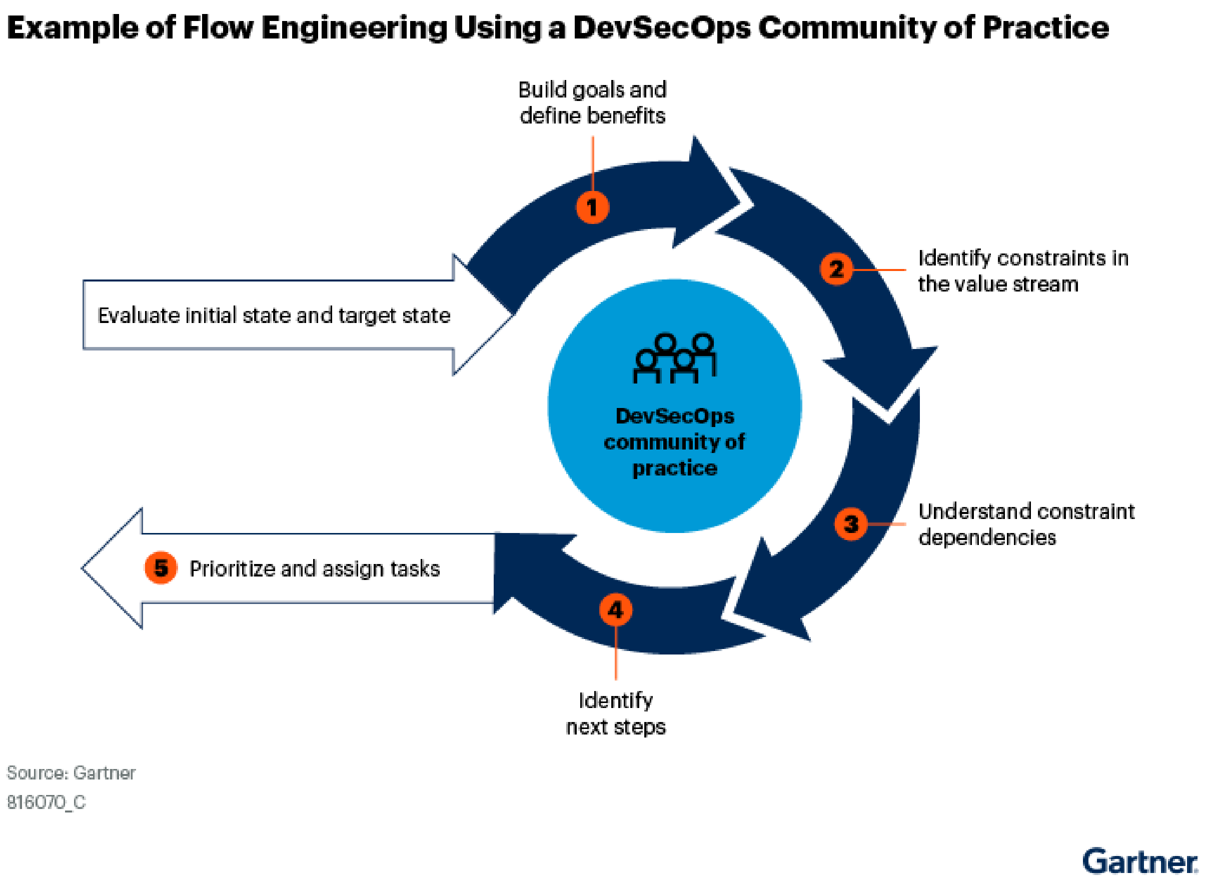
1. Cybersecurity: A representative from cybersecurity leadership should be present, including the chief information security officer (CISO) or a team leader, for application security. Depending on the scale of the cybersecurity team, this could include multiple leaders. They will contribute their knowledge to improve and manage automated security guardrails and enable security as a service.
2. Software engineering: Software engineering leaders who oversee the engineering domain. These leaders work with the business unit, product owners and project managers.
3. Infrastructure and operations: Those who define the infrastructure for building, deploying and operating application workloads must be included. I&O professionals contribute their knowledge of application workload architecture and how to make improvements to DevSecOps by simplifying infrastructure definition and reducing gaps between developers and their application workloads.
4. Platform engineering: Not all organizations will have an engineering platform in place, but those that do must include these roles. Platform engineering is quickly

becoming a standard for high-performing software engineering. DevSecOps will become a key tenet of platform engineering (see 2024 Planning Guide for Security).

5. Business unit(s): Improving DevSecOps maturity will require buy-in and investment from the business unit. Plan to work with a sponsor from the business to share goals and benefits from improving DevSecOps maturity. Use Six Steps to Manage Cybersecurity Risk Appetite Through Protection-Level Agreements to build buy-in for investments.

As shown in Figure 2, the DevSecOps CoP should continuously meet on a regular basis (typically at the end of a sprint) and work iteratively toward improving DevSecOps maturity using "Flow Engineering." [7]

Figure 2: Example of Flow Engineering using a DevSecOps Community of Practice


Example of Flow Engineering Using a DevSecOps Community of Practice
Build goals and define benefits
1
Identify constraints in the value stream
2
Evaluate initial state and target state
DevSecOps community of practice
Understand constraint dependencies
3
5
Prioritize and assign tasks
4
Identify next steps
Source: Gartner
816070_C
Gartner

During each session, use the following steps to structure the discussion:

1. Working backward from the desired target maturity state, identify the goals and benefits for DevSecOps improvements.

2. Discover any obstacles or constraints in the value stream that obstruct reaching the desired state.
3. Understand what external dependencies are contributing to constraints.
4. Identify next steps for improvement activities to reach the desired target state.
5. Prioritize and assign ownership of improvement tasks.

There are several other factors to be aware of and addressed when operating a CoP, including costs, risks and red flags. See Ignition Guide to Creating Communities of Practice in Software Engineering.

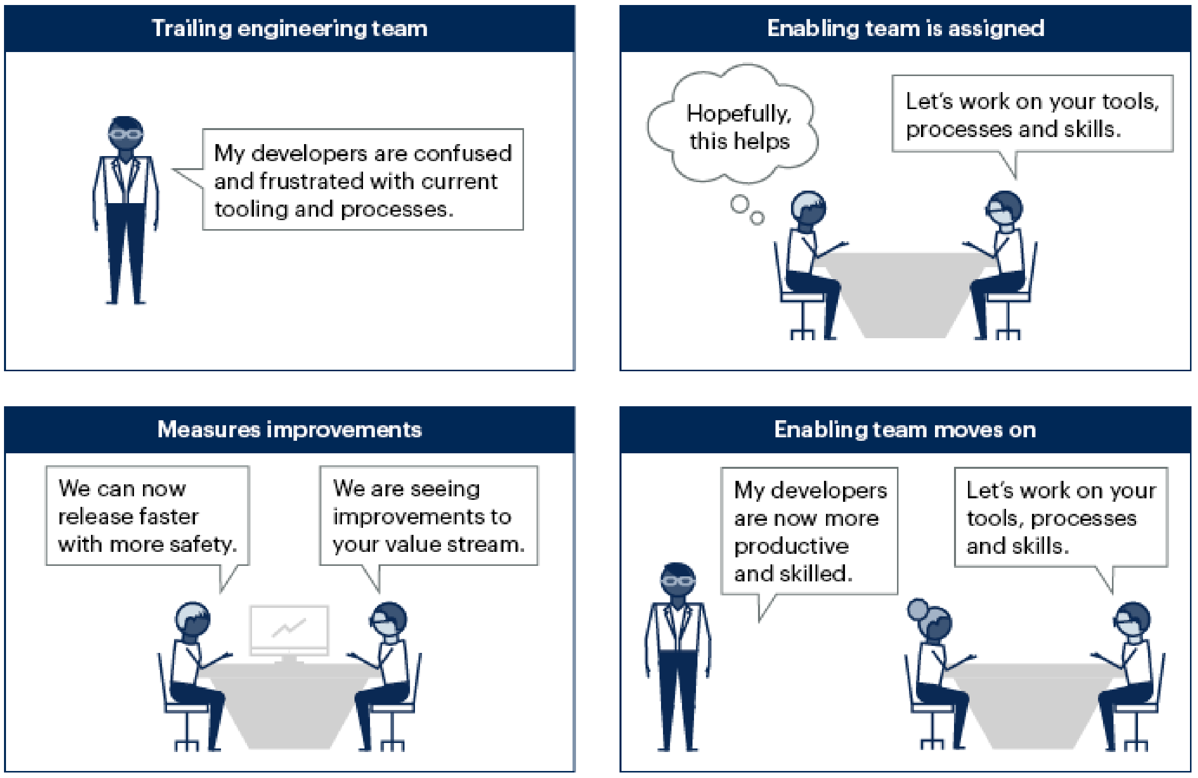
Create DevSecOps Enabling Teams

The concept of an enabling team is a recent trend in engineering organizations. An enabling team's purpose is to help trailing teams upskill and onboard to new tools and knowledge. These teams are made up of senior and staff practitioners across the engineering organization that have demonstrated the highest standards of security in their respective domains.

Enabling teams will partner with one or two other trailing software teams at a time to actively elevate their security maturity. As seen in Figure 3, the enabling team will evaluate, coach and mentor trailing teams to bring them up to speed with newly defined DevSecOps improvements. Once completed with one engagement, the enabling team will move onto another. They can alternatively be thought of as a technical consulting team.[8] Most enabling teams plan for their own extinction, so the teams they help do not become dependent on them.

Figure 3: Utilize Enabling Teams to provide training, coaching and mentoring for DevSecOps

Utilize Enabling Teams to Provide Training, Coaching and Mentoring for DevSecOps


Trailing engineering team
My developers are confused and frustrated with current tooling and processes.
Enabling team is assigned
Hopefully, this helps
Let's work on your tools, processes and skills.
Measures improvements
We can now release faster with more safety.
We are seeing improvements to your value stream.
Enabling team moves on
My developers are now more productive and skilled.
Let's work on your tools, processes and skills.

Source: Gartner
816070_C

Gartner

Enabling teams do not own their own systems or software products, but are purposely independent to help other teams bridge their knowledge gaps and help in their DevSecOps journey. Enabling teams must also be able to demonstrate their impact, defining desired DevSecOps outcomes early in their engagements and measuring once certain levels of maturity are reached. See Four Steps to Develop Outcome-Driven Metrics for Cybersecurity to develop these outcomes.

The CoP meets regularly to map the overarching strategy of security maturity and investments required across multiple engineering domains, and the enabling teams focus on tactical experimentation and implementing DevSecOps improvements. See Organize for Agility With Team Topologies for even greater detail about enabling teams and Team Topologies.

Evidence

Note 1: Example Outcome-Driven Metrics for DevSecOps

1. Lead time for changes: Average time from request to delivery.
2. Release candidate failure rate (security): Percentage of release candidates that are sent back from test to development for security fixes.
3. Production release failure rate (security): Percentage of production releases that require urgent security fixes.
4. Automated security testing coverage: Percentage of code that is covered by automated security testing.
5. Security incident response time: Average time from incident reported to incident closed.
6. Number of vulnerabilities detected versus remediated: The difference between the number of vulnerabilities detected and vulnerabilities remediated, lower being better.