Agile and DevSecOps Overview

Tim Chick

tchick@sei.cmu.edu

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM22-0640

Working Definition of Agile

Agile

An *iterative* and *incremental* (evolutionary) approach to software development which is performed in a *highly collaborative manner* by *self-organizing teams* within an *effective governance framework* with *"just enough" ceremony* that produces *high quality software* in a *cost effective and timely* manner which *meets the changing needs of its stakeholders*. [Ambler 2013]

[Ambler 2013] Ambler, Scott. *Disciplined Agile Software Development: Definition*. http://www.agilemodeling.com/essays/agileSoftwareDevelopment.htm

Agile Manifesto

Manifesto for Agile Software Development

February 2001

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan
That is, while there is value in the items on the right,
we value the items on the left more.

The Twelve Agile Principles$_1$

1. Our highest priority is to **satisfy the customer through early and continuous delivery of valuable software**.
2. **Welcome changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. **Business people and developers must work together daily throughout the project.**
5. **Build projects around motivated individuals.** Give them the environment and support they need, **and trust them to get the job done**.
6. The most efficient and effective method of **conveying information** to and within a development team is **face-to-face conversation**.

The Twelve Agile Principles$_2$

7. **Working software is the primary measure of progress.**
8. Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to **maintain a constant pace indefinitely**.
9. **Continuous attention to technical excellence and good design enhances agility.**
10. Simplicity—**the art of maximizing the amount of work not done**—is essential.
11. **The best architectures, requirements, and designs emerge from self-organizing teams**.
12. **At regular intervals, the team reflects on how to become more effective**, then tunes and adjusts its behavior accordingly.

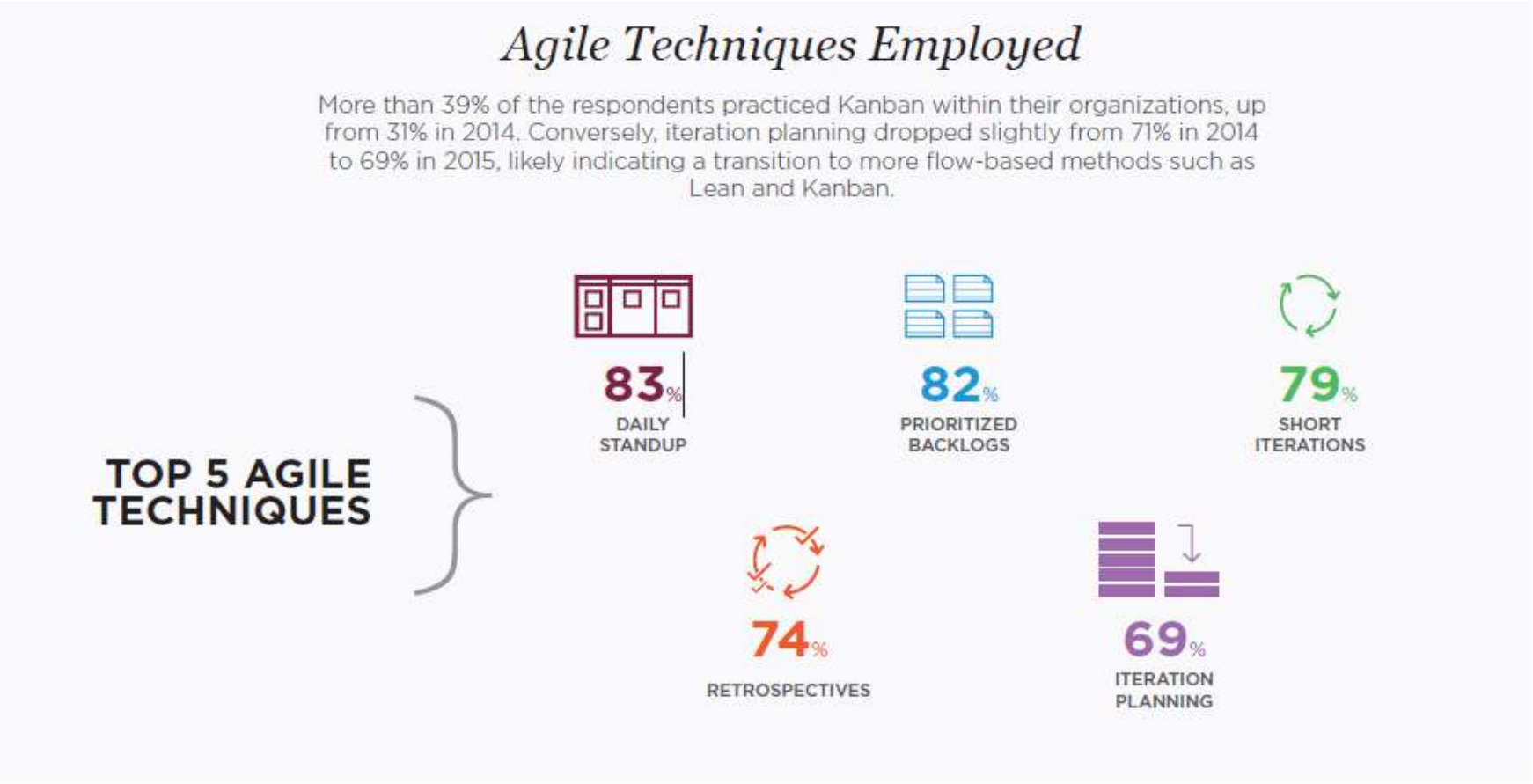
Traditional and Agile Perspectives on Software Development

	Traditional View	Agile Perspective
Design Process	Deliberate and formal, linear sequence of steps, separate formulation and implementation, rule-driven	Emergent, iterative and exploratory, knowing and action inseparable, beyond formal rules
Goal	Optimization	Adaptation, flexibility, responsiveness
Problem-solving Process	Selection of the best means to accomplish a given end through well-planned, formalized activities	Learning through experimentation and introspection, constantly reframing the problem and its solution
View of the Environment	Stable, predictable	Turbulent, difficult to predict
Type of Learning	Single-loop/adaptive	Double-loop/generative
Key Characteristics	Control and direction Avoid conflict Formalizes innovation Manager is controller Design precedes implementation	Collaboration and communication; integrates different worldviews Embraces conflict and dialectics Encourages exploration and creativity; opportunistic Manager is facilitator Design and implementation are inseparable and evolve iteratively
Rationality	Technical/functional	Substantial
Theoretical and/or Philosophical Roots	Logical positivism, scientific method	Action learning, John Dewey's pragmatism, phenomenology

Dyba, Tore; Torgeir Dingsoyr, *What Do We Know about Agile Software Development*, Published by the IEEE Computer Society, 0740-7459/09, 2009.

Methods Generally Termed "Agile"

Top 5 Agile Techniques Used in Industry

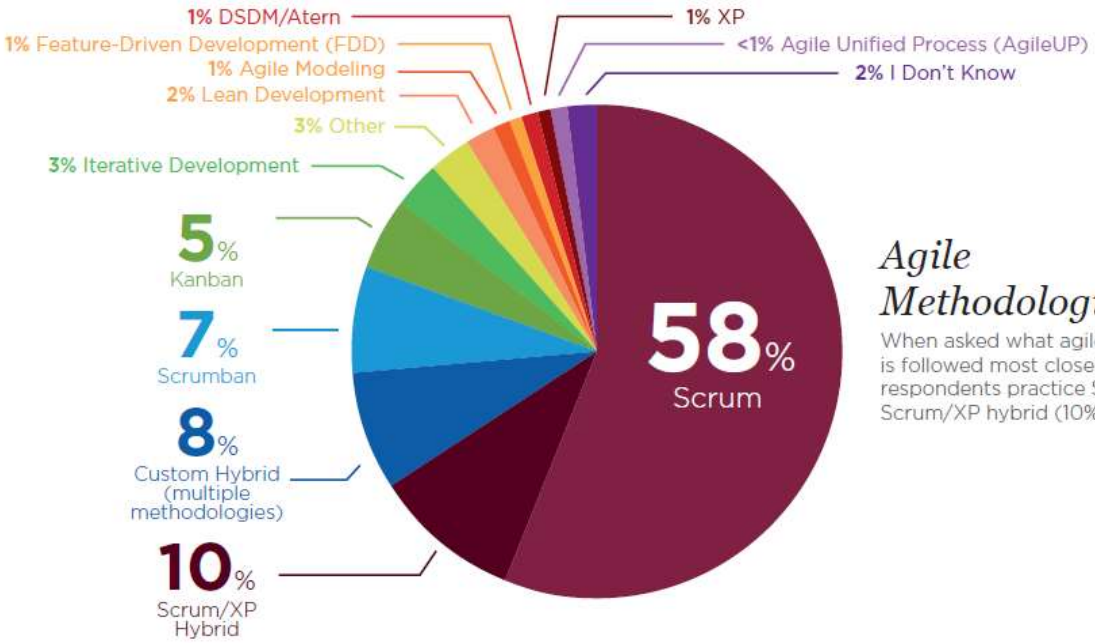


Source: 10th Annual Survey on State of Agile, Version One

Kanban - One Way to Implement the Top 5 Agile Techniques

10th Annual STATE OF AGILE™ REPORT

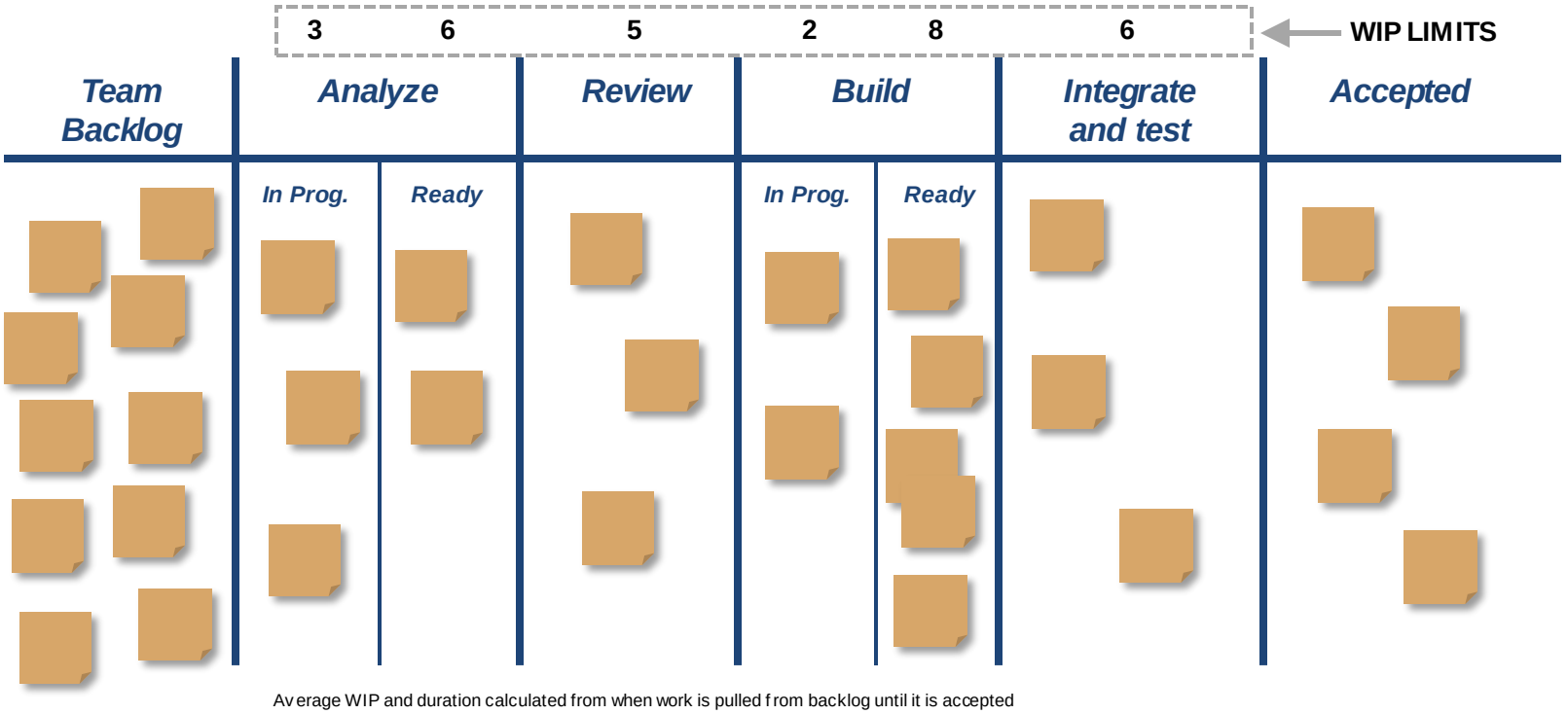
AGILE METHODS AND PRACTICES



Agile Methodologies Used

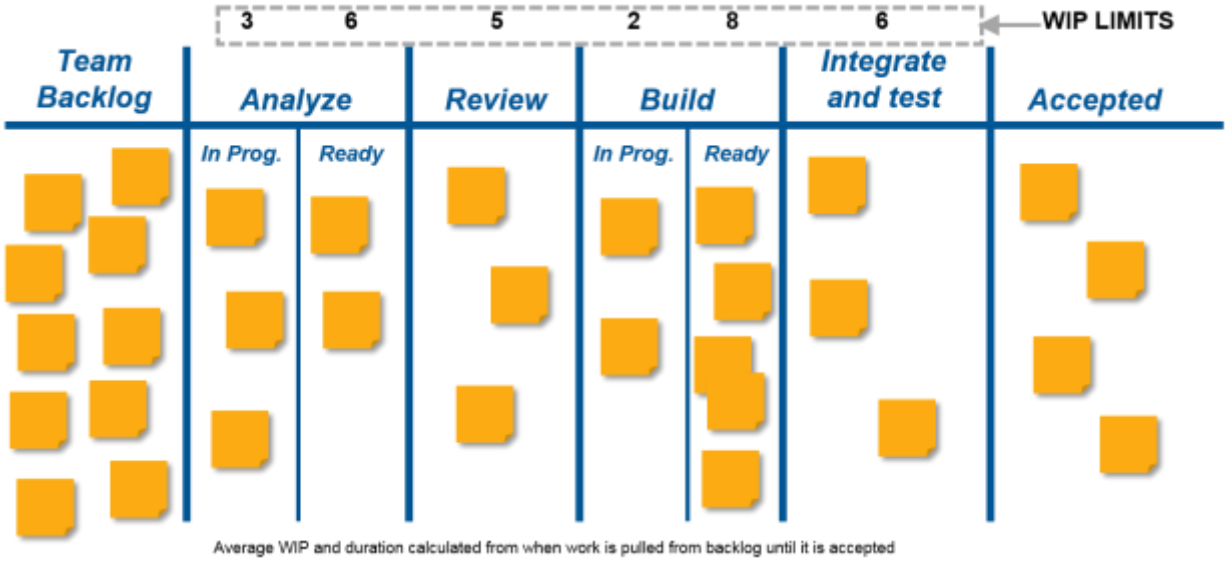
When asked what agile methodology is followed most closely, nearly 70% of respondents practice Scrum (58%) or Scrum/XP hybrid (10%).

Visualize and Limit WIP, Reduce Batch Sizes, and Manage Queue Lengths



Average WIP and duration calculated from when work is pulled from backlog until it is accepted

Key Concepts-Useful for Tasks that Don't Easily Conform to a Time Box



Defined States the Work Progresses Through

Rules about Limiting Work in Process for Each State

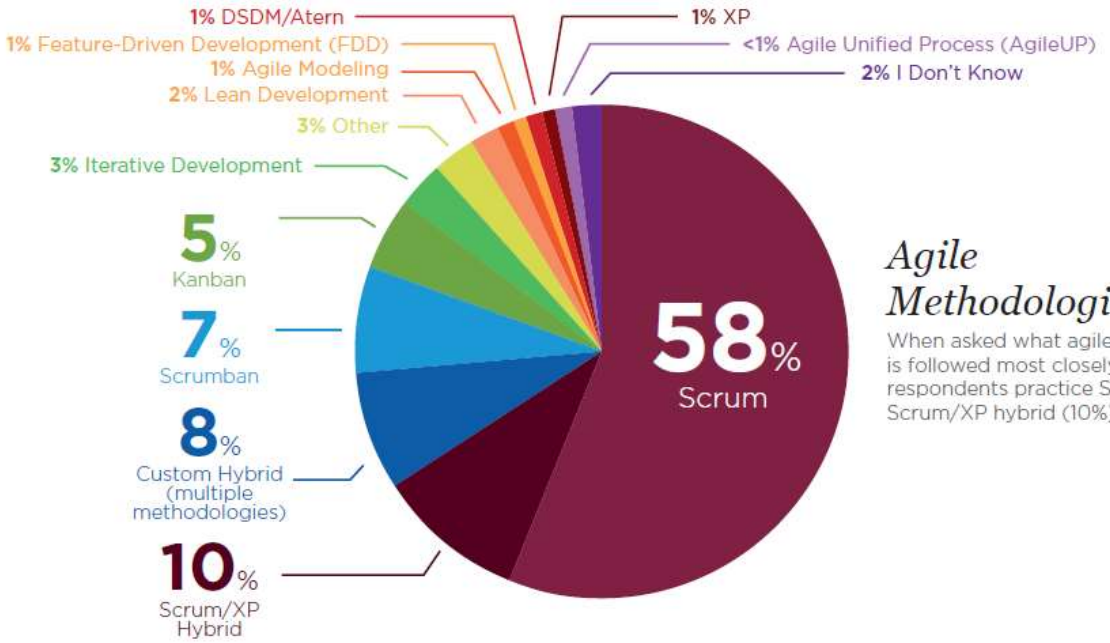
Definitions of "Special" Classes of Tasks (Due Date, Expedited, or others as defined by the project)

Explicit Acceptance Criteria

Scrum - Another Way to Implement the Top 5 Agile Techniques

10th Annual STATE OF AGILE™ REPORT

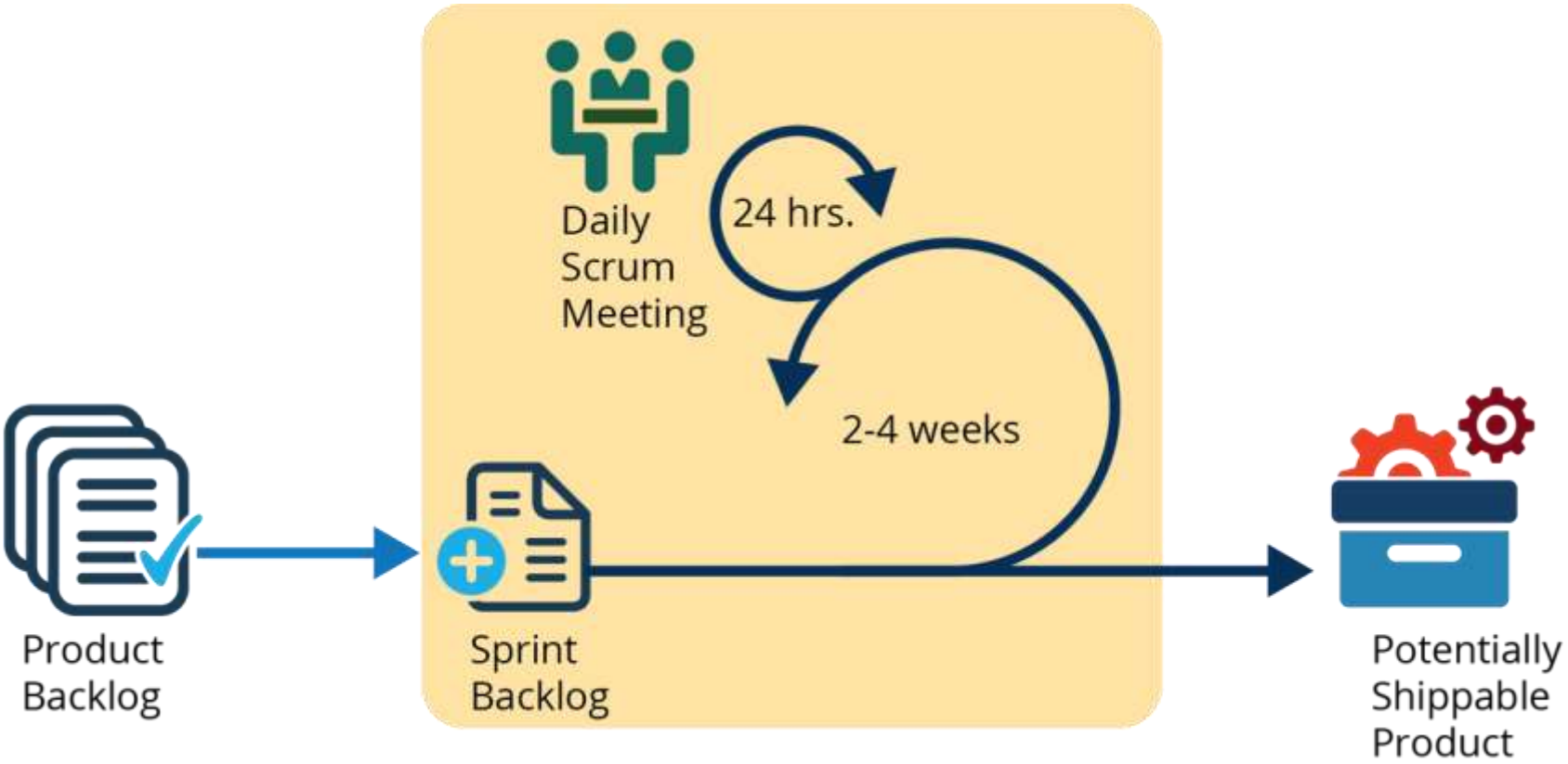
AGILE METHODS AND PRACTICES



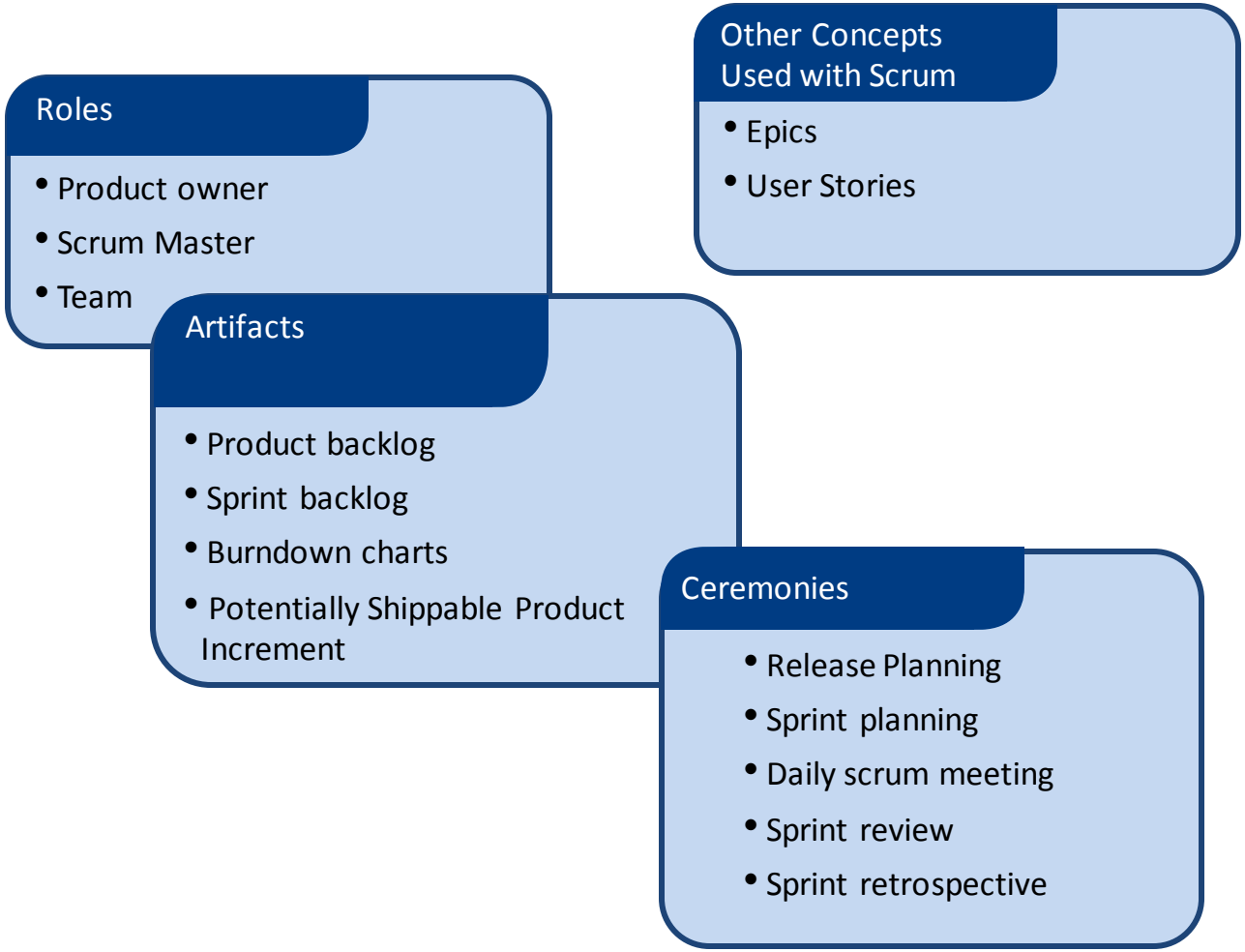
Agile Methodologies Used

When asked what agile methodology is followed most closely, nearly 70% of respondents practice Scrum (58%) or Scrum/XP hybrid (10%).

Key Elements of Scrum



Scrum Framework



New Terms - Roles

Product Owner: the "voice of the customer," accountable for ensuring business value is delivered by creating customer-centric items (typically stories or user stories), prioritizing them, and maintaining them in the product backlog.

Scrum Master: the process facilitator for a development team who works with the team to identify and find solutions to impediments to progress; also maintains the information radiators that show progress to stakeholders and represents the team in management activities related to the project.

Team: cross-functional self-organizing team of 5-10 members, including Scrum Master, Product Owner or Capability Owner, developers, testers, and SMEs.

New Terms - Artifacts

Product Backlog: a prioritized list of (user) stories and defects ordered from the highest priority to the lowest.

Sprint Backlog: a list of tasks that the team believes need to be completed to satisfy the user stories for that sprint and meet the sprint goal.

Burndown Charts: a visual tool displaying progress via a simple line chart representing remaining work (vertical axis) over time (horizontal axis).

Potentially Shippable Product Increment: the result of a sprint. Even though it is unlikely that the project would be cancelled after a particular sprint, the idea is that the product is implemented in such a way that some value could be derived no matter when the project stops.

New Terms – Ceremonies$_1$

Release Planning: planning activities across a defined number of sprints that implement a desired set of features to the point of delivery to the next customer (could be external, often is internal, e.g., system test).

Sprint Planning: planning activities that occur at the beginning of a sprint, including determining the capacity for the sprint, establishing a sprint goal, performing relative estimation on the candidate stories for the sprint, and creating the task list (sprint backlog) the team will use to self-manage the work of the sprint.

Daily Scrum Meeting: daily standups that are used to communicate what was accomplished yesterday, what will be accomplished today, and to inform the Scrum Master of any impediments in order for action to be taken to eliminate the issue.

New Terms – Ceremonies$_2$

Sprint Review: the activity at the end of a sprint that demonstrates the code that has been completed and verifies, with the product owner, that the sprint goal has been met.

Sprint Retrospective: an activity at the end of the sprint review/demo in which the team and the Scrum Master review the processes and practices used in the sprint to identify improvements to be tried in the next or future sprints—Agile's way of "inspect and adapt" for processes.

New Terms - Other

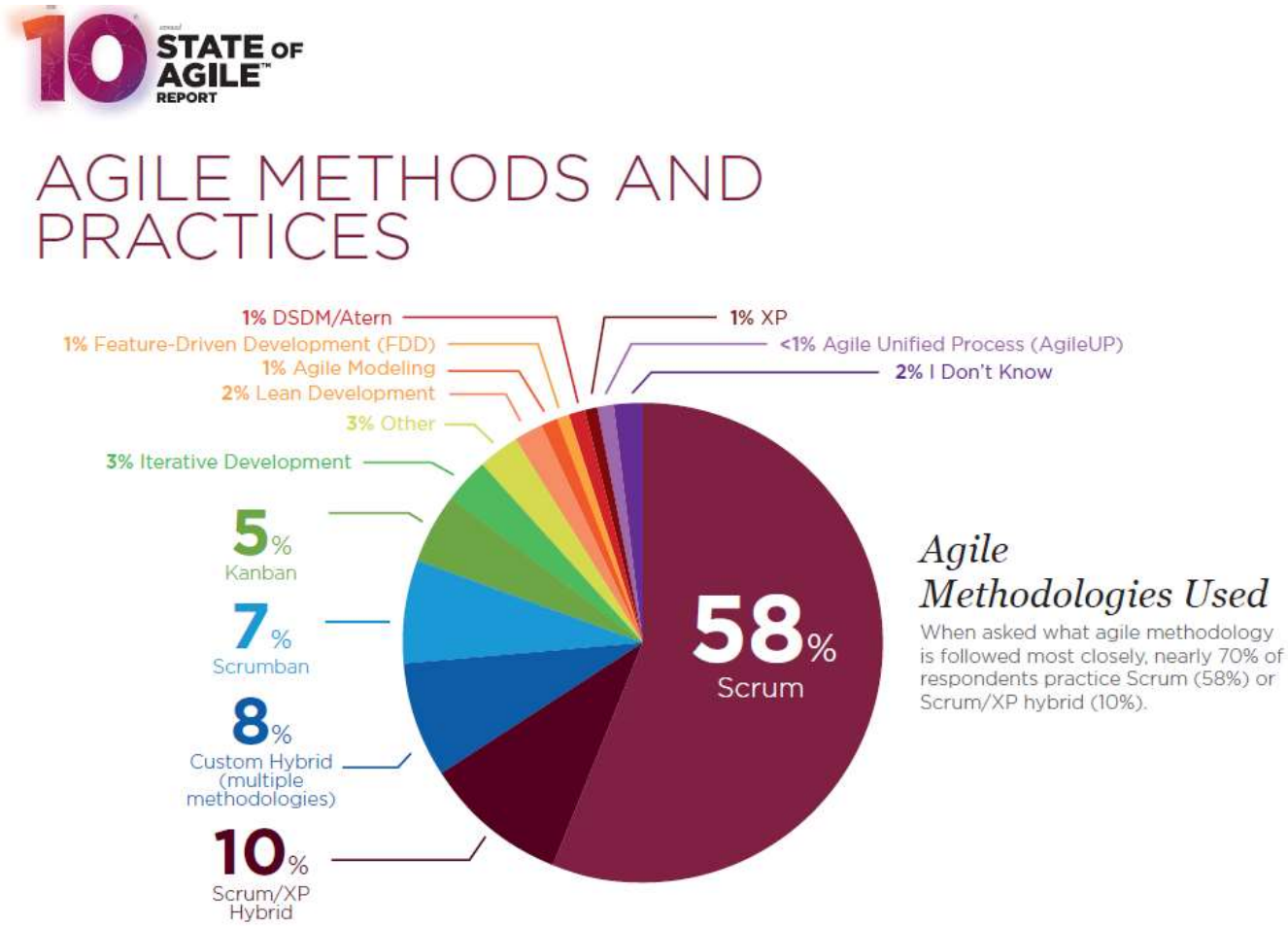
Epics: user stories that are too large to directly implement. There is no official threshold to differentiate between an epic and a user story.

User stories: used in several Agile methods; derived from Extreme Programming; used as the basis for defining the functions a system must provide, they include a written sentence or two and a series of conversations about the desired functionality to shift the focus from writing about requirements to talking about them.

Sprint: an iteration of a defined, consistent time span (2-4 weeks is typical) during which the backlog items selected for the iteration are planned, designed, implemented, tested, and demonstrated to the Product Owner/Customer.

Information Radiator: a physical or virtual display of tasks and progress that is accessible to all stakeholders throughout the project.

Scaled Agile Framework (SAFe) – Goes beyond Small Teams



BUT, the Agile Principles Were Designed and Focused on Small Teams

There are lots of things beyond supporting a small team that you have to focus on when scaling above a few small teams:

- managing the interfaces among the many products/system components that multiple teams are working on…
- figuring out how to synchronize releases and events across multiple teams…
- figuring out how to get the inventory (backlog) of requirements organized productively to support the development pace of multiple small teams….
- dealing with specialty disciplines (UX, security, etc.) that have significant inputs to the evolving product, but aren't needed as full time team members….

Agile is a Team Approach

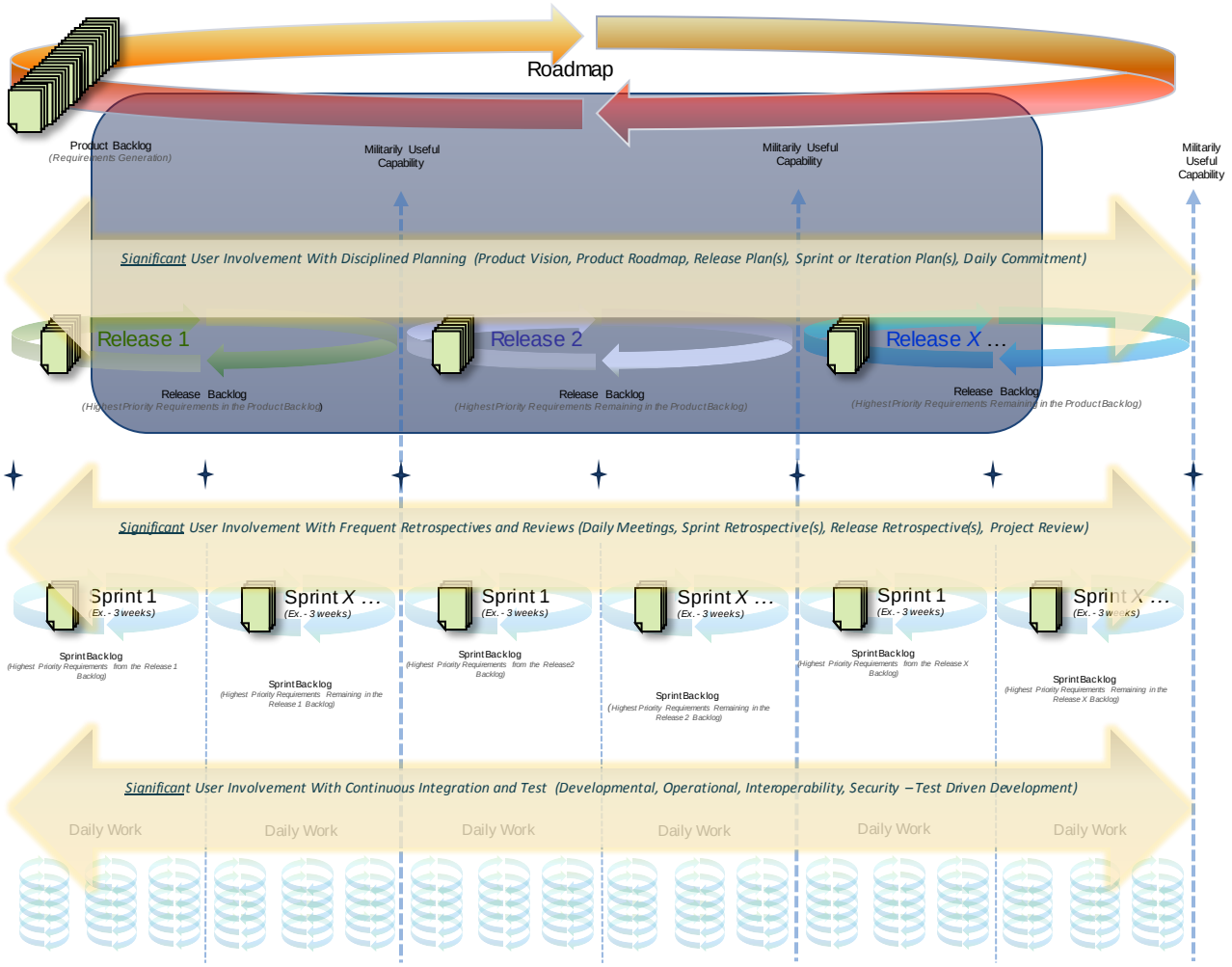
Agile can't succeed in a vacuum. There are Different roles to play by:

- Developers
- Testers
- End Users
- Customer Representative
- Subject Matter Experts
- Program Office
- Contracts
- Finance
- Certifiers
- ….

Not a Myth: Agile *is* likely to fail if it's "only the development team" that adopts the new practices

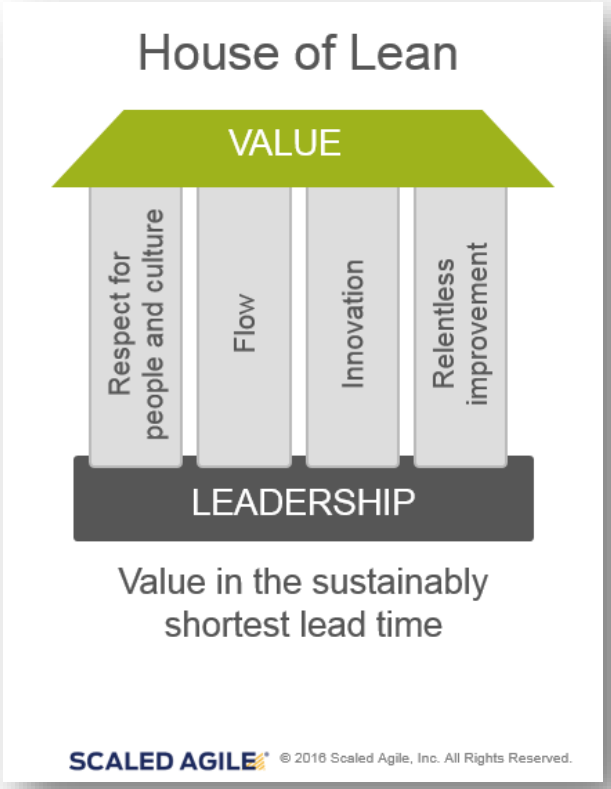
Frequent failure mode:
Business and/or operations doesn't keep up with what the development teams can deliver.

Why?
Shift to Agile-based requirements definition and management is more of a change than practices of development alone.



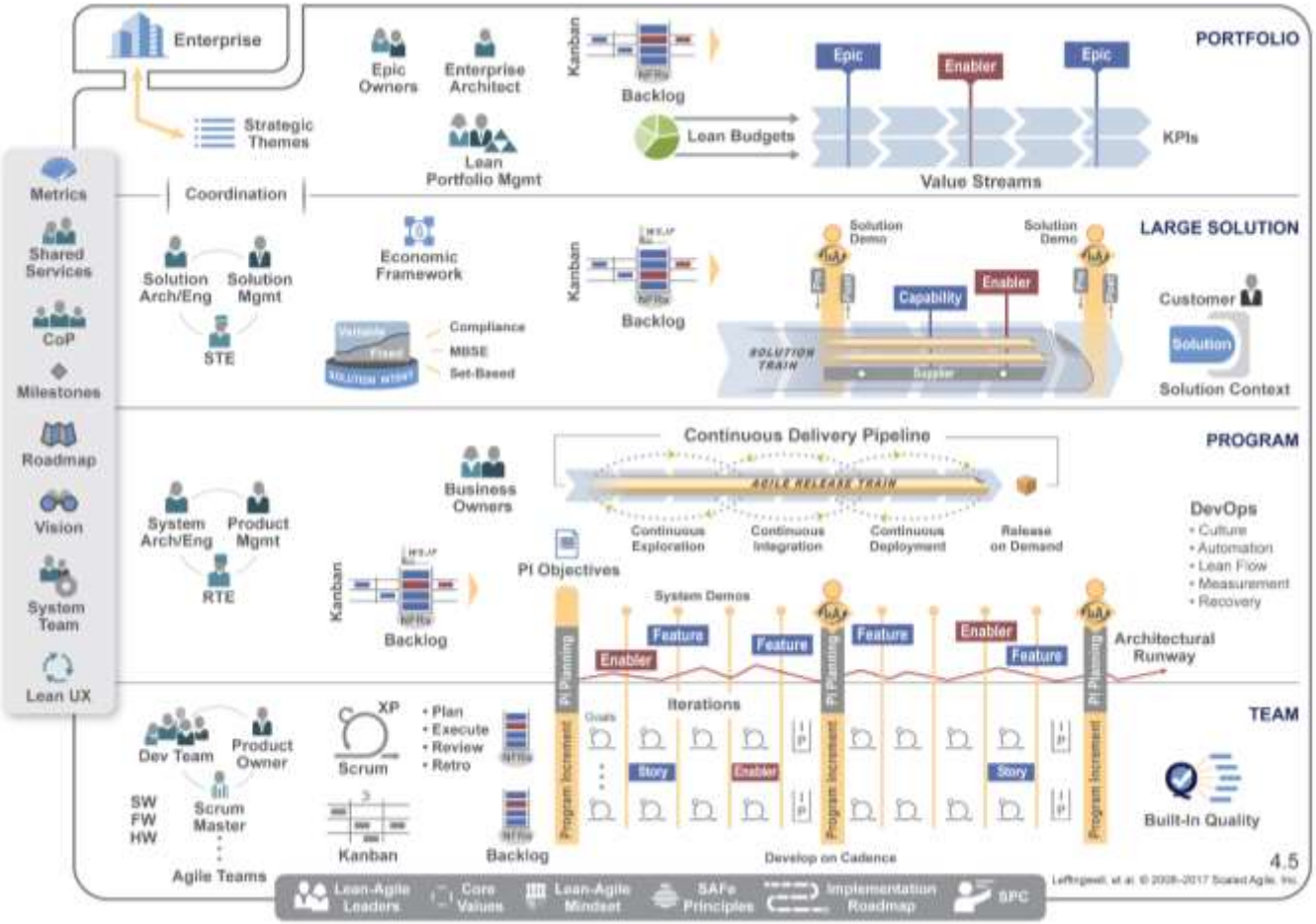
Graphic Source: Figure 4, *Parallel Worlds: Differences in Agile and Waterfall Differences & Similarities,* S. Palmquist et al, SEI-2013-TN-021, October 2013.

Lean Plus Agile=Scaled Agile Framework (SAFe)



SAFe® is a freely revealed knowledge base of integrated, proven patterns for enterprise Lean-Agile development.

Scaled Agile Framework (SAFe)—
Scaling Framework that is Most Frequently Seen in Large Organizations



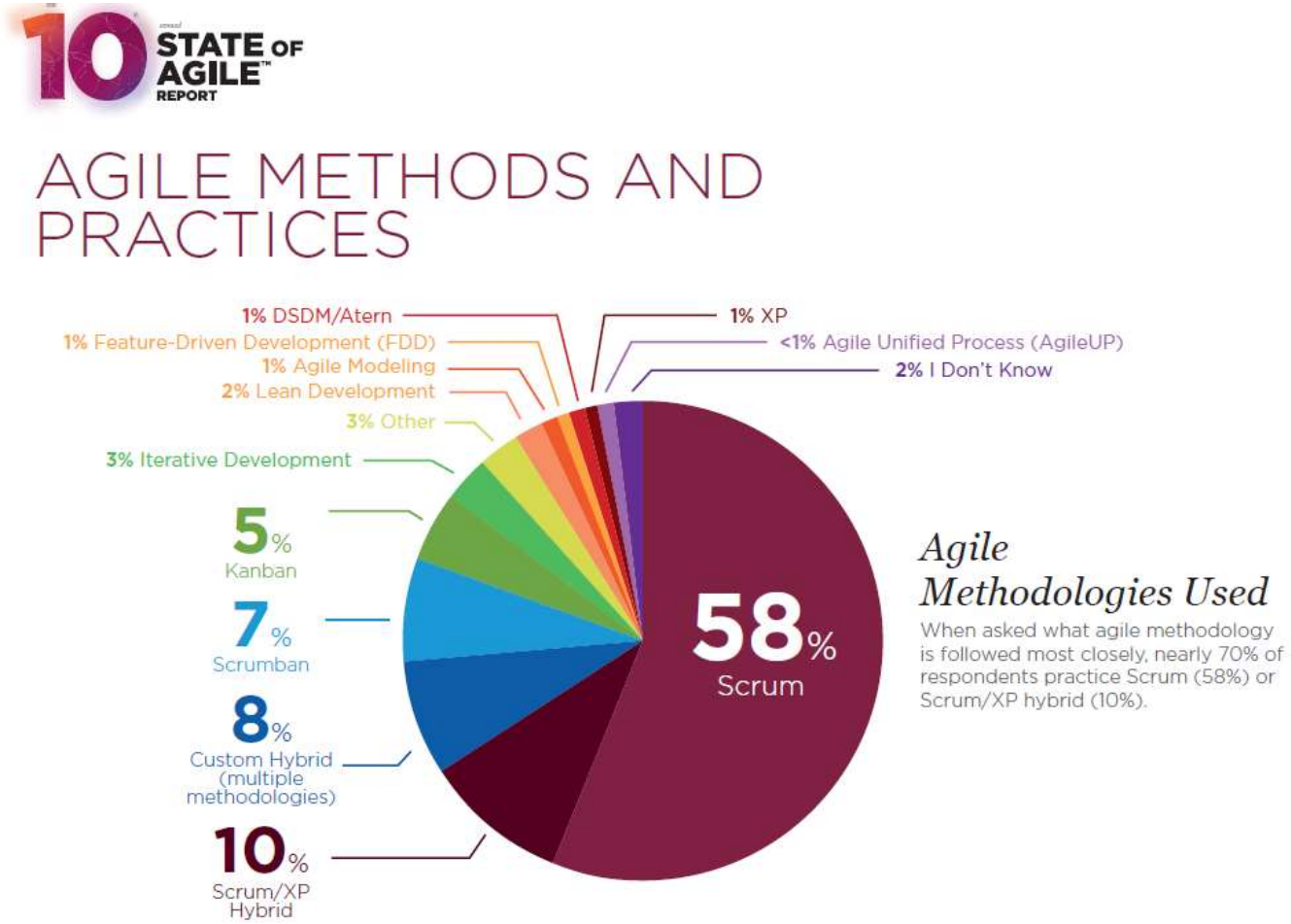
SAFe leverages lean engineering practices at Portfolio, Value Stream, and Program levels.

SAFe leverages Scrum, XP, and Kanban practices at the Team level.

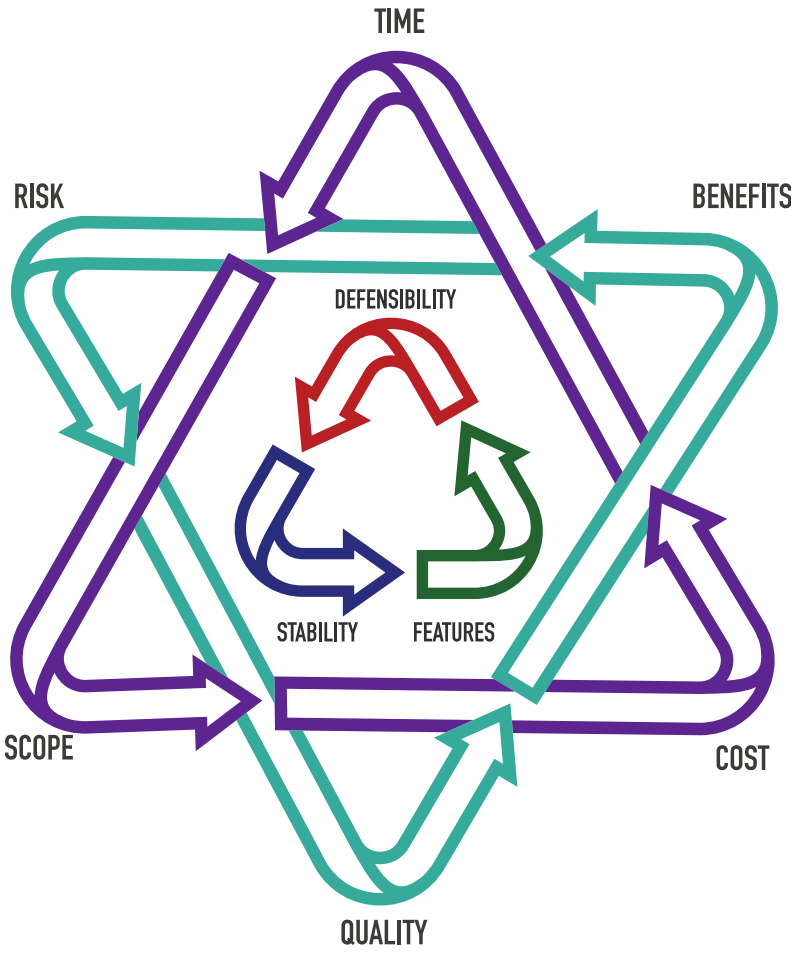
SAFe is Also Principles Based—Lean Engineering and Startup Principles

1. Take an economic view.
2. Apply systems thinking.
3. Assume variability; preserve options.
4. Build incrementally with fast, integrated learning cycles.
5. Base milestones on objective evaluation of working systems.
6. Visualize and limit WIP, reduce batch sizes, and manage queue lengths.
7. Apply cadence, synchronize with cross-domain planning.
8. Unlock the intrinsic motivation of knowledge workers.
9. Decentralize decision-making.

DevSecOps–
Agile goes beyond management techniques



DevSecOps: A fancy term for modern software engineering practices and tools that encompasses the full software development lifecycle



DevSecOps is a cultural and **engineering practice** that breaks down barriers and opens **collaboration between development, security, and operations** organizations **using automation** to focus on rapid, frequent delivery of secure infrastructure and software to production. It encompasses intake to release of software and manages those flows predictably, transparently, and with minimal human intervention/effort [1].

A DevSecOps CI/CD Pipeline attempts to seamlessly integrate "three traditional factions that sometimes have opposing interests: **development**; which values features; **security**, which values defensibility; and **operations**, which values stability [2]."

Not only does one need to balance the factions. They must do so in a way that balances **risk**, **quality** and **benefits** within their **time**, **scope**, and **cost** constraints.

[1] DevSecOps Guide: Standard DevSecOps Platform Framework. U.S. General Services Administration. https://tech.gsa.gov/guides/dev_sec_ops_guide. Accessed 17 May 2021
[2] DevSecOps Platform Independent Model, https://cmu-sei.github.io/DevSecOps-Model/

Who are Dev?

Follow Agile methodologies

- Use Scrum, Kanban and modern development approaches
- Self directing, self managed, self organized

Use any new technology

- Each Dev has its own development strategy
- OpenSource

Have

- Close relationships with the business
- Software driven economy

Want to deliver software faster with new requirements…

Who are Ops?

Operations

- Runs the application
- Manages the infrastructure
- Support the applications

Operations provides

- Service Strategy
- Service Design
- Service Transition
- Service Operations
- Secure Systems

Want to maintain stability, reliability, and security…

DevOps has four Fundamental Principles

Collaboration: between project team roles.

Infrastructure as Code: all assets are versioned, scripted, and shared where possible.

Automation: deployment, testing, provisioning, any manual or human-error-prone process.

Monitoring: any metric in the development or operational spaces that can inform priorities, direction, and policy.

Security Whac-A-Mole

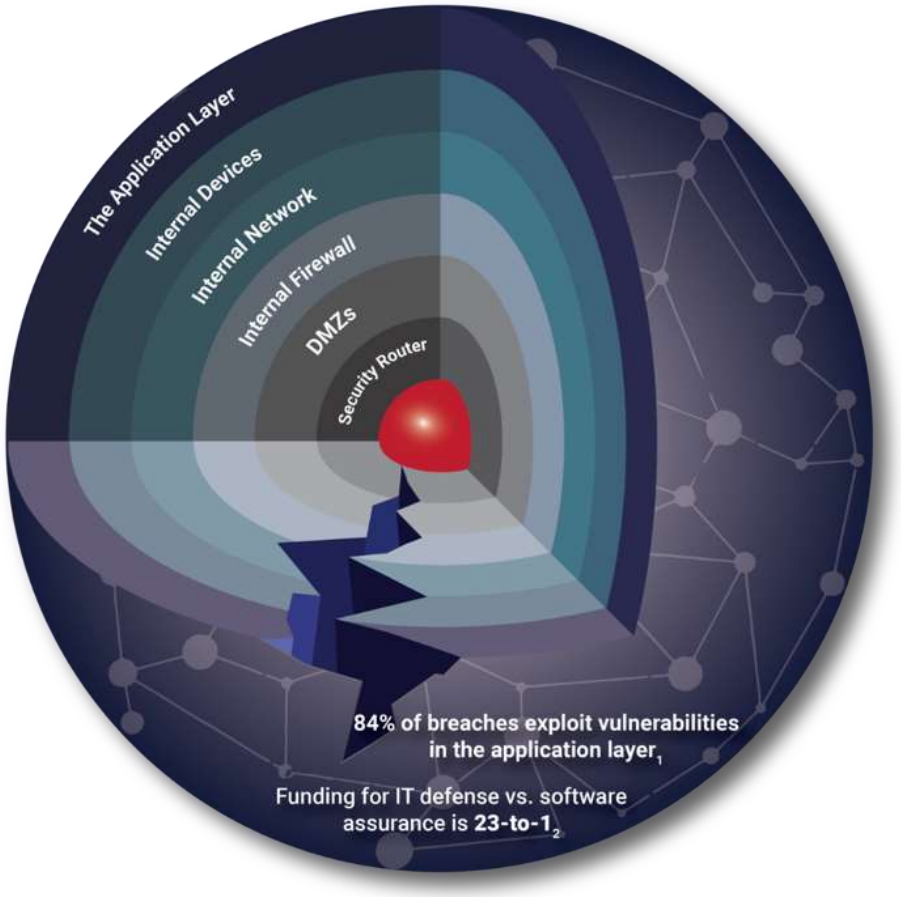
Winning in Features and Effectiveness, but Losing in Defensibility and Stability

In June of 2020 a generally successful DoD program completed an **8 week "Hardening the Software Factory" effort** in order to address **accumulated technical debt** and to address **insufficient security and operations** practices **due to the narrow focus on speed of delivery**.

These things occur, even in small relatively successful programs, when technical debt and insufficient security and operational practices are in place **due to lack of knowledge, experience, and reference material to fully design and execute an integrated DevSecOps strategy in which all stakeholder needs, including cybersecurity, are addressed.**

While playing Whac-A-Mole is inevitable, instead of missing the holes, or constantly hitting the same hole, the key is to fill in the holes.

Effective Security Requires a Holistic Approach



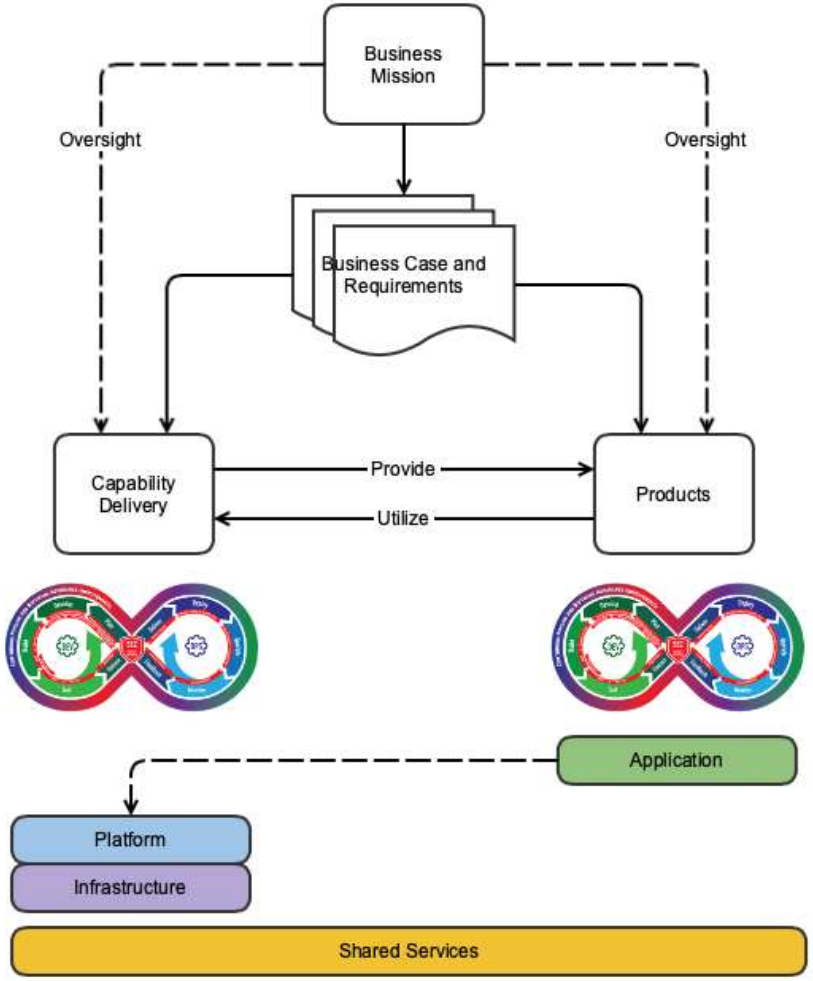
The Application Layer is the new perimeter exploited by 84% of breaches

Security must be Engineered into the Lifecycle of Applications changing the way we build and buy technology

- "76 percent of U.S. developers use no secure application program process"[4]
- "More than 40 percent of software developers globally say that security isn't a top priority for them"[4]
- 2017 less than 5% of DevOps initiatives have achieved the level of security automation required to be considered fully DevSecOps.[3]

1. Clark, Tim, *Most cyber Attacks Occur from this Common Vulnerability,* Forbes. 03-10-2015
2. Feiman, Joseph, *Maverick Research: Stop Protecting Your Apps; It's Time for Apps to Protect Themselves,* Gartner. 09-25-2014. G00269825
3. *Horvath, Mark, Neil MacDonald, Ayal Tirosh: Integrating Security Into the DevSecOps Toolchain, Gartner. 11-16-2017. G00334264*
4. Microsoft[1]– http://visualstudiomagazine.com/articles/2013/07/16/majority-of-us-devs-dont-practice-secure-coding.aspx

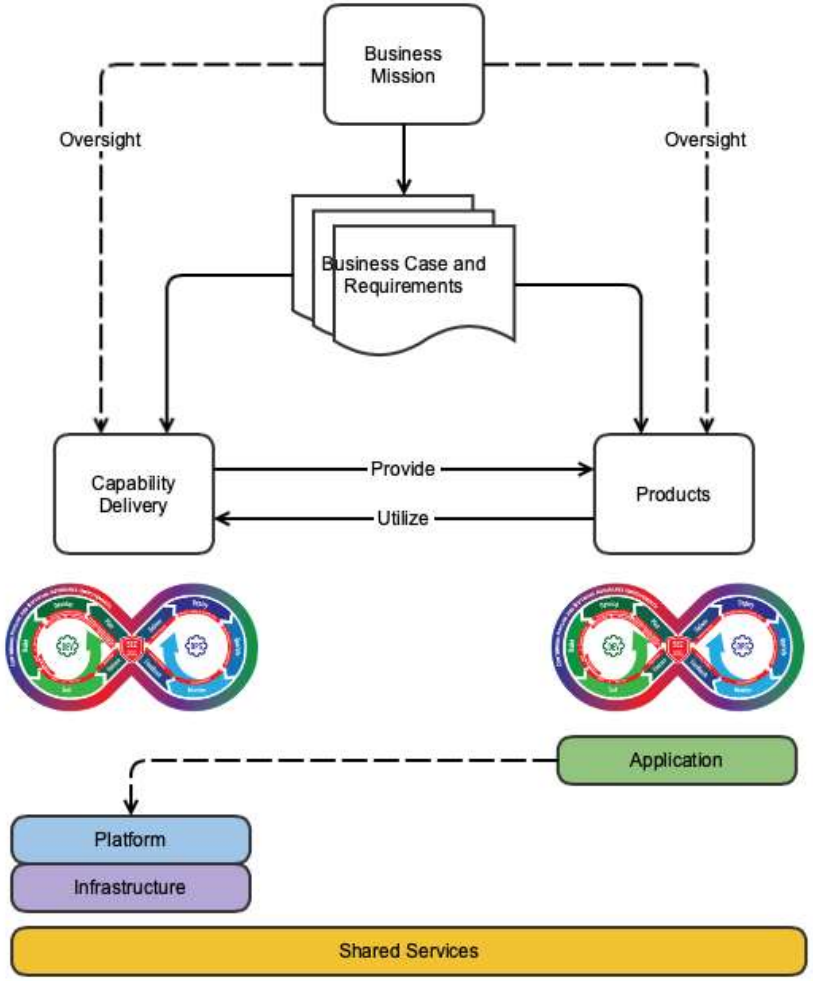
Challenge 1 for DevSecOps: connecting process, practice, & tools



Creation of the DevSecOps (DSO) pipeline for building the product is not static.

- Tools for process automation must work together and connect to the planned infrastructure
- Everything is software and all pieces must be maintained but responsibility will be shared across multiple organizations (Cloud for infrastructure, 3rd parties for tools and services, etc.)

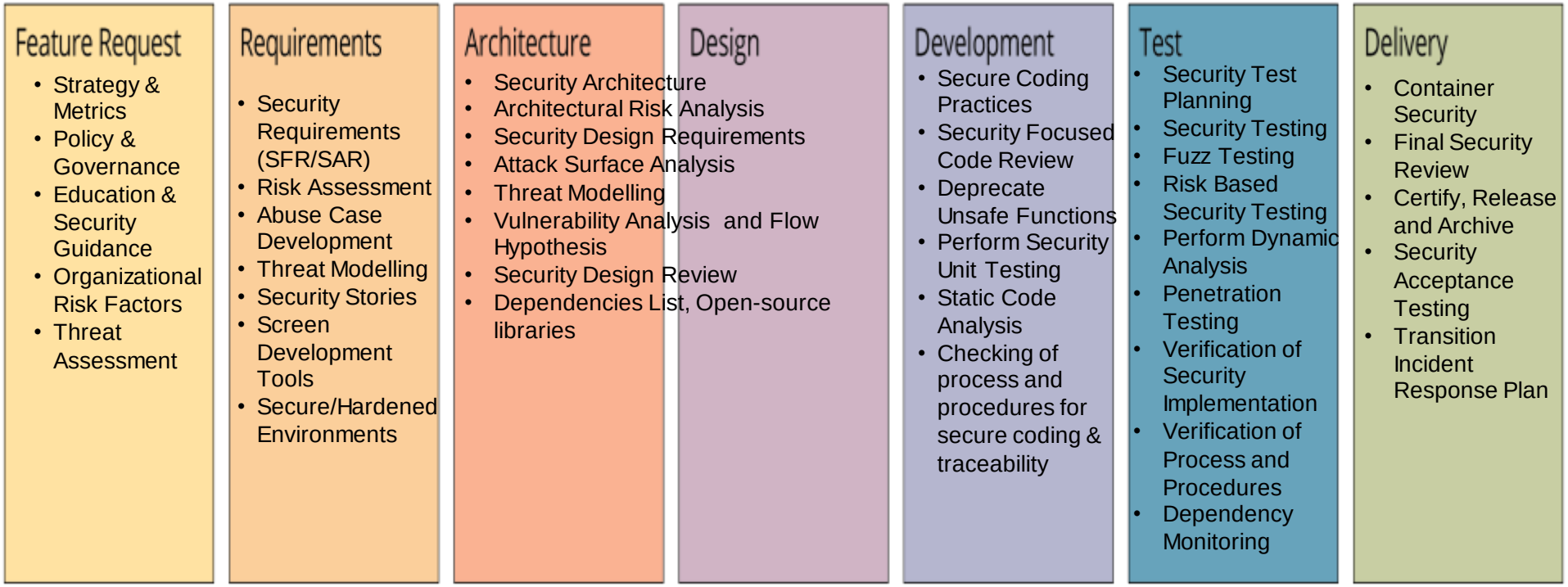
Challenge 2 for DevSecOps: cybersecurity of pipeline and product



Managing and monitoring all of the various parts to ensure the product is built with sufficient cybersecurity and the pipeline is maintained to operate with sufficient cybersecurity is complex. Cybersecurity demands effective governance to address:

- What trust relations will be acceptable, and how will they be managed?
- What flow control and monitoring are in place to establish that the pipeline is working properly? Are these sufficient for the level of cybersecurity required?
- What compliance mandates are required? How are they addressed by the pipeline? Is this sufficient?

DevOps Pipeline Phases need Security



These phase are continuously integrated. They are not traditional large batch hand-offs.

All Agile Techniques are Wrong, but Some are Useful[1]

George Box is famously quoted as saying, "All models are wrong but some are useful." The same can be said for the various Agile methods, as much of the material around Agile assumes a simplification or idealization of a model development team.

The key to successful Agile implementation is understanding how you will instantiate the Agile manifesto and principles.

The Agile principles have implications for the characteristics of the lifecycle that can be used.

But there's still more than one valid way of implementing the principles….

All Agile Techniques are Wrong, but Some are Useful$_2$

The family of Agile methods has grown since 2000 to incorporate techniques that address team, project, and enterprise levels of scaling.

Hybrids of multiple methods and techniques are common practice in both industry and government.

This is one reason it's so difficult to say a program is "Agile" or not.

As leaders, you must facilitate the team and organization in selecting the correct Agile techniques, regardless of chosen methods, to meet your organization's and customer's goals and objectives.

Selecting the Appropriate Agile Methods

Three Fundamental Factors

1. Identifying **the ability of the organization** to adopt agile practices
2. Determining **the suitability of agile practices in the development** of a given product or system
3. Determining **the suitability of agile practices for the organization** developing the product or system.

Adapted from Sidky, Ahmed; James Arther, *Determining the Applicability of Agile Practices to Mission and Life-critical Systems,* Proceedings of the 31st IEEE Software Engineering Workshop (SEW 2007). pp 3-12.

Ability of the Organization to Adopt Agile

1. Agile is like any other process improvement initiative.

   Successful adoption of agile practices requires the absorption of associated costs, as well as expending the required time and effort.

2. Use the following method for deciding whether to proceed or not:

   a. Identify success factors, such as
      - level of executive sponsorship
      - organizational desire to improve
      - availability of resources

   b. Assess the extent to which the success factors are present.

   c. Make a go/no-go decision based on assessment results.

Adapted from Sidky, Ahmed; James Arther, *Determining the Applicability of Agile Practices to Mission and Life-critical Systems,* Proceedings of the 31st IEEE Software Engineering Workshop (SEW 2007). pp 3-12

Determining the Suitability of Agile Techniques

1. Development and product characteristics play a large role in determining the suitability of a particular agile technique.
   For example, life-critical systems are usually characterized as being large, complex, and having long development periods

2. The desired product qualities also play a role in determining appropriate agile technique.
   For example, maintainability and reliability

An agile practice is considered unsuitable and must be discarded or tailored when it conflicts with integral product/development characteristics, or precludes the attainment of desirable qualities

Adapted from Sidky, Ahmed; James Arther, *Determining the Applicability of Agile Practices to Mission and Life-critical Systems,* Proceedings of the 31st IEEE Software Engineering Workshop (SEW 2007). pp 3-12

When is this unsuitable or impractical?[1]

On-site customer: The customer must be available full-time to answer questions as they arise, preferably face-to-face, and preferably on-site.

> **There are many stakeholders** who are unable to speak to one another's requirements and priorities (need for a traditional System's Engineer or Requirements Analysis).

Simple design: The design is kept as simple as possible. You only do enough design to implement the next feature.

> **Refactoring is most difficult without an architecture.** It is important to distinguish between basic infrastructure services and application functionality. The former must be stable very fast and experience very little change, because in infrastructure code can have serious ripple effects that result in cost prohibitive changes.

Source of examples: Emam, Khaled El; *Finding Success in Small Software Projects,* Cutter Consortium, Agile Project Management, Vol. 4, No. 11.

When is this unsuitable or impractical?$_2$

Minimal documentation: The discouragement of writing requirements and design because of the cost of development and maintenance and the fact they do not deliver value to the customer, only working software does.

1. **Staffing flux** results in the tacit knowledge of the developers being lost, which can result in a considerable slowdown in the work.
2. The lack of documentation demonstrating the origins of some technical material can be costly when faced with an **intellectual property lawsuit.**

Refactoring – making existing code simpler, faster, more robust, cleaner, or eliminating duplication, without changing the function of the code.

Refactoring a Database Schema: Because the system would already be in place, all data would need to be migrated to a new database, which is not a trivial exercise. Also, all modules that interact with the database would require modification or at least inspection. Reinforces the need for a stable design up front.

Source of examples: Emam, Khaled El; *Finding Success in Small Software Projects,* Cutter Consortium, Agile Project Management, Vol. 4, No. 11.

Suitability of Agile for the Organization

1. When mismatches between the Agile method and the organization are identified before the adoption effort, the probability of failing during the adoption effort or introducing undesirable agile practices is reduced.

   Is this a mismatch?

   1. Assuming **Test-First Programming** (TFP) is a suitable method in meeting the required product/development characteristics and enables the attainment of desirable qualities. It has also been determined that **your organization already has a dedicated integration and system test team.**

      No, as TFP addresses how the developers do unit testing and has no effect on the dedicated test team.

   2. Assuming **Pair Programming** is a suitable method in that it meets the required product/development characteristics and enables the attainment of desirable qualities. It has also been determined **that the organization has major collaborative issues.**

      Yes, as Pair Programming works best in a collaborative environment.

Adapted from Sidky, Ahmed; James Arther, *Determining the Applicability of Agile Practices to Mission and Life-critical Systems,* Proceedings of the 31st IEEE Software Engineering Workshop (SEW 2007). pp 3-12

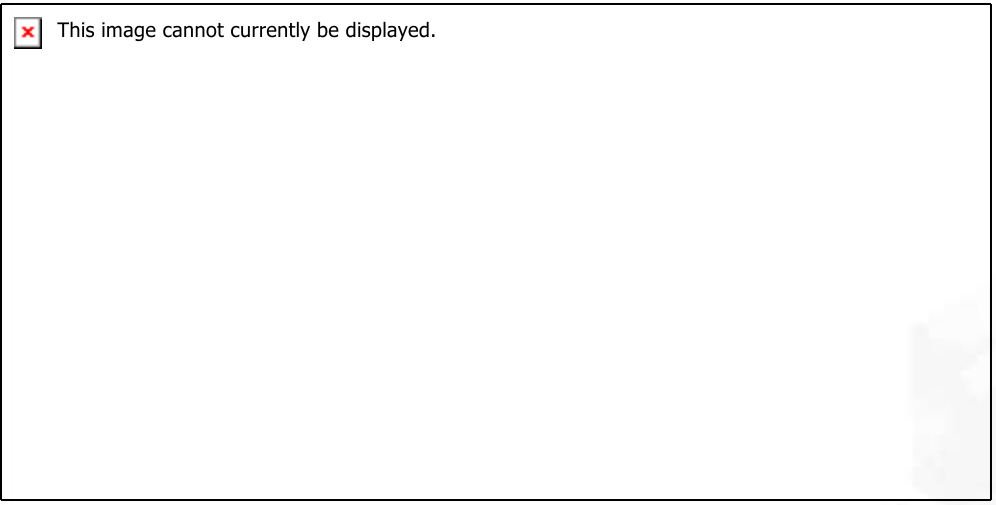
Critical Agile Implementation Questions$_1$

Agile Principle	Question
1. Our highest priority is to satisfy the customer through early and continuous delivery of **valuable** [**assume working**] software.	How would you measure the software to establish that it is both working and valuable?
2. Welcome changing requirements, **even late in development**. Agile processes harness change for the **customer's competitive advantage**.	How would you evaluate **late requirements change to ensure that it provides competitive advantage for the customer?**
3. Deliver **working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	How do you determine the optimal frequency? Can you deliver too frequently?
4. Business people and developers must **work together daily** throughout the project.	Is daily availability realistic? What if there is more than one stakeholder position?
5. Build projects around motivated individuals. Give them the environment and support they need, and **trust them to get the job done**.	Trust is earned, not given. How will you establish and sustain a trusting relationship with the software development team?
6. The most efficient and effective method of conveying information to and within a development team is **face-to-face** conversation.	Is face-to-face practical? What if the teams are large or distributed? What happens to the tacit system knowledge that the team has after the team is dissolved?

Critical Agile Implementation Questions$_2$

Agile Principle	Discussion Question
7. **Working software** is the primary measure of progress.	How would you use "working software" as a progress measure?
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to **maintain a constant pace indefinitely**.	What does **maintain a constant pace indefinitely** mean?
9. Continuous attention to technical excellence and good design enhances agility.	
10. Simplicity--the art of maximizing the amount of work not done--is essential.	What's the danger here?
11. The best **architectures, requirements, and designs emerge** from self-organizing teams.	Working code is the progress measure but the architecture, requirements, and designs emerge. What does this imply about the **working software** progress measure?
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	

Picking Agile and DevSecOps Techniques



How Is It Done Today, and What Are the Limits of Current Practice?

- Currently, guidance and policies focus on functionality and leave the major decision making to the programs:
  - "DoD organizations should define their own processes, choose proper activities, and then select tools suitable for their systems to build software factories and DevSecOps ecosystems" [8]
  - "The PM shall ensure that software teams use iterative and incremental software development methodologies (such as Agile or Lean), and use modern technologies (e.g. DevSecOps pipelines) … "[9]
- Program offices lack sufficient capability to design, build, and implement a DSO continuous integration/continuous delivery (CI/CD) pipeline.
- Current guidance
  - fails to prepare the program office to address the full socio-technical aspects of DSO
  - is not definitive and require a considerable amount of interpretation, resulting in:
    - DSO perspectives not being fully integrated in DoD guidance and policy documents
    - Program offices being unable to perform an analysis of alternative (AoA) in regards to the DSO pipeline tools and processes
    - Multiple programs using similar infrastructure and pipelines in different and incompatible ways, even within the same program
    - Suboptimal tools and security controls
- Large and complex DoD weapon system acquisition programs have already embraced model-based engineering, but have not applied the same techniques to their DSO CI/CD pipelines. This limits a program office's ability to build a cyber-physical software factory that is fit for purpose.

Enterprise Architecture (EA) Approach

Why EA?: To provide the knowledge and tools that will enable program offices to adopt DevSecOps in support of the new software acquisition pathway [9].

An EA approach will expand the focus of DevSecOps to the whole enterprise.

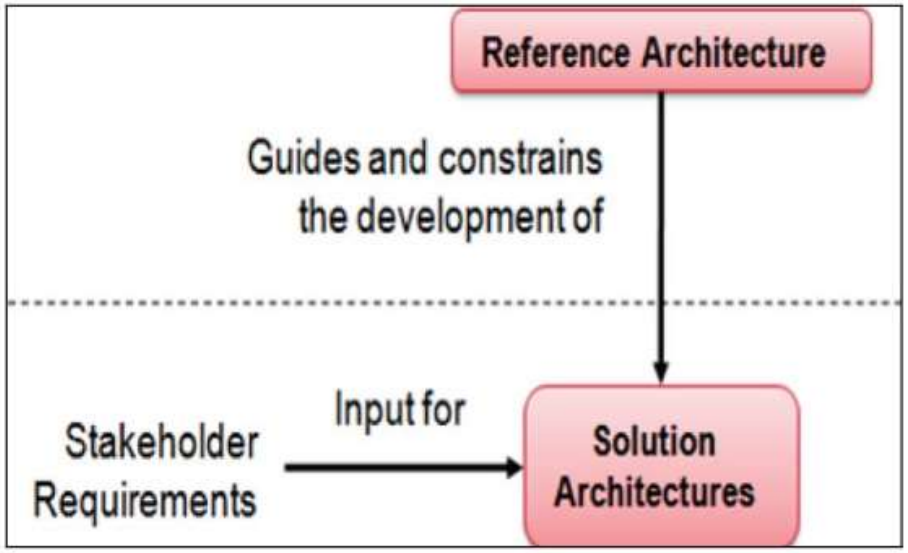
- What capabilities does the enterprise need to be successful?
- How must existing functional roles (e.g., program managers, finance, contracting, logistics, security, engineers) adapt to succeed in DevSecOps?
- What new roles (e.g., product development, software engineering, information security, IT operations) are required?
- What critical processes must be implemented for DevSecOps and what information do they need?
- What existing policies, guidelines, and best practices provide requirements and constraints for the enterprise to achieve its DevSecOps goals?

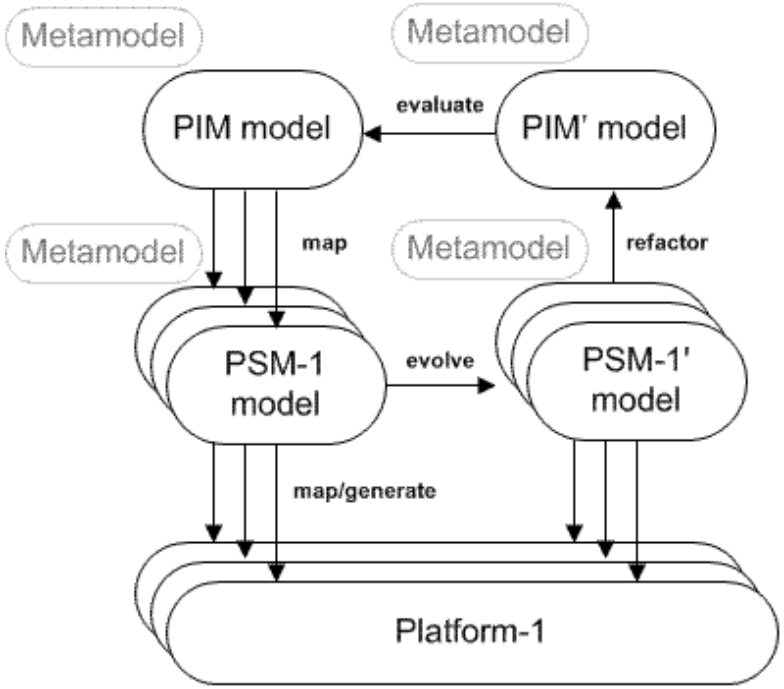
Goal: What is the *Minimum Viable Enterprise* that is capable of defining, developing, and operating a *Minimum Viable Product?*

Reference Architecture/Platform Independent Model (PIM)

A **Reference Architecture** is an authoritative source of information about a specific subject area that guides and constrains the instantiations of multiple architectures and solutions [18].

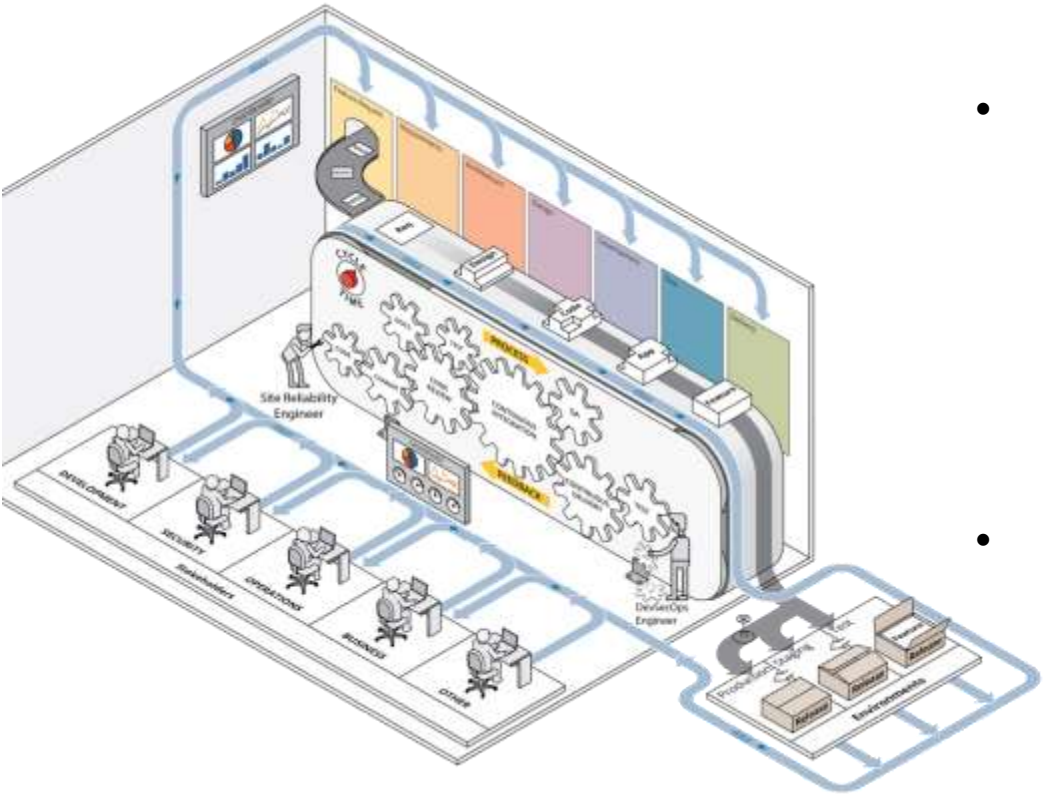
A PIM is a general and reusable model of a solution to a commonly occurring problem in software engineering within a given context, and is independent of the specific technological platform used to implement it.





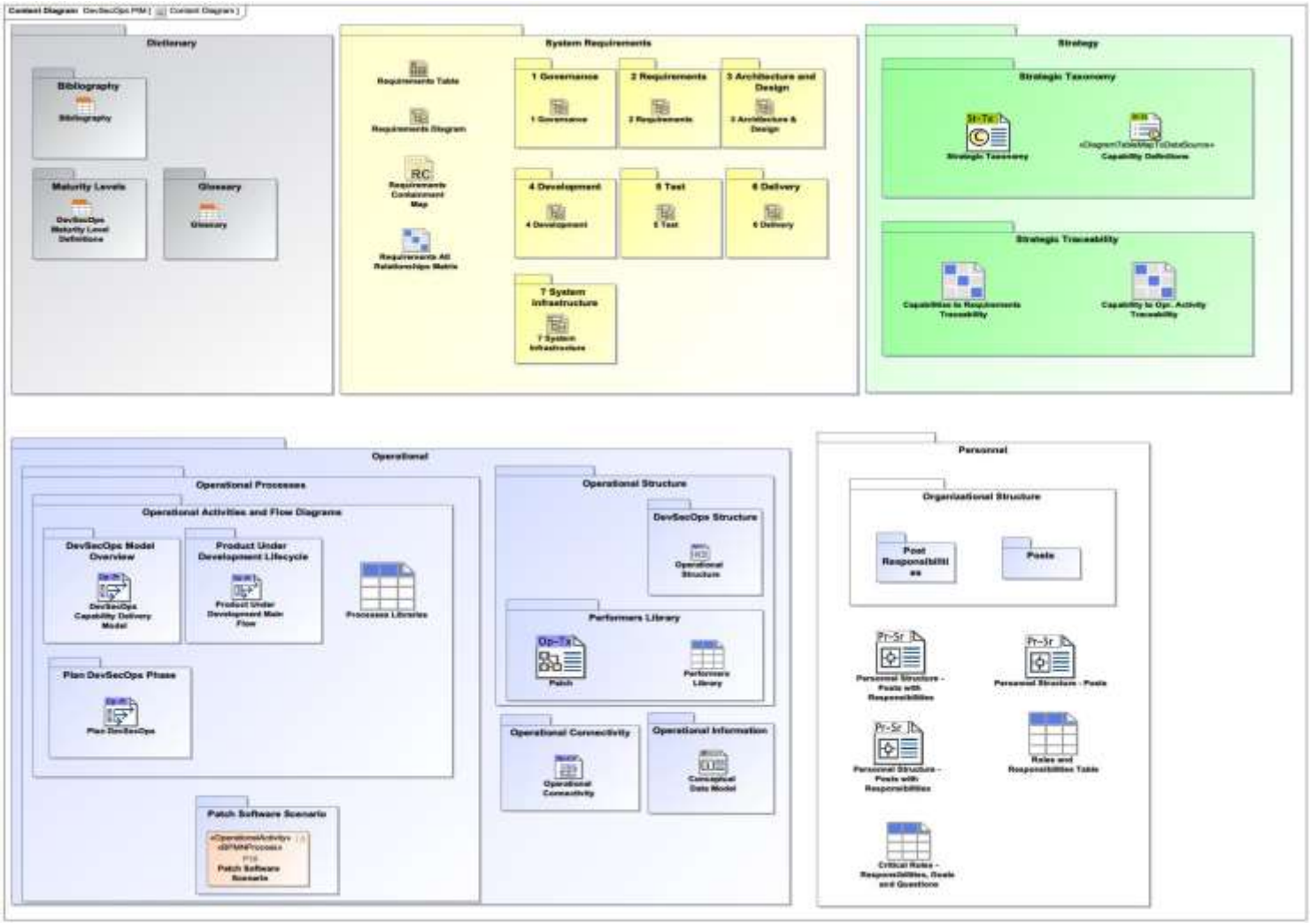
NOTE: PSM = Platform Specific Model

DevSecOps PIM



- Is an authoritative reference to fully design and execute an integrated Agile and DevSecOps strategy in which all stakeholder needs are addressed
- Enables organizations to implement DevSecOps in a secure, safe, and sustainable way in order to fully reap the benefits of flexibility and speed available from implementing DevSecOps principles, practices, and tools
- was developed to outline the activities necessary to consciously and predictably evolve the pipeline, while providing a formal approach and methodology to building a secure pipeline tailored to an organization's specific requirements.

DevSecOps PIM - Content Diagram



https://cmu-sei.github.io/DevSecOps-Model/

The DevSecOps PIM Provides

- consistent guidance and modeling capability that ensure all proper layers and development concerns relevant to the organization's, project's, and team's needs are captured
- the basis for creating a DevSecOps platform-specific model (PSM) that can be incorporated into the product's model-based engineering approach as the DevSecOps master model is included in the product's model. This allows proper modeling of DevSecOps design trades within a project's Analysis of Alternatives (AoA) processes, resulting in less costly and more secure products.
- the basis for metrics and documentation of trade-offs to be captured and analyzed through the model-based engineering approach. The model provides dynamic matrices of if those points were addressed, how they were addressed, and how well the corresponding (to the points) module is covered.
- the basis for performing risk modeling against decisions and DevSecOps model-based engineering to ensure security controls and processes are properly selected and deployed

The DevSecOps PIM enables Organizations, Projects, Teams, and Acquirers to

- specify the DevSecOps requirements to the lead system integrators tasked with developing a platform-specific solution that includes the designed system and continuous integration/continuous deployment (CI/CD) pipeline
- assess and analyze alternative pipeline functionality and feature changes as the system evolves
- apply DevSecOps methods to complex products that do not follow well-established software architectural patterns used in industry
- provide a basis for threat and attack surface analysis to build a cyber assurance case to demonstrate that the product and DevSecOps pipeline are sufficiently free from vulnerabilities and that they function only as intended

Questions? Next Steps?

Through proper balance, programs should be able “to maintain a constant pace (i.e., play Topple) indefinitely” that results in a system that is:

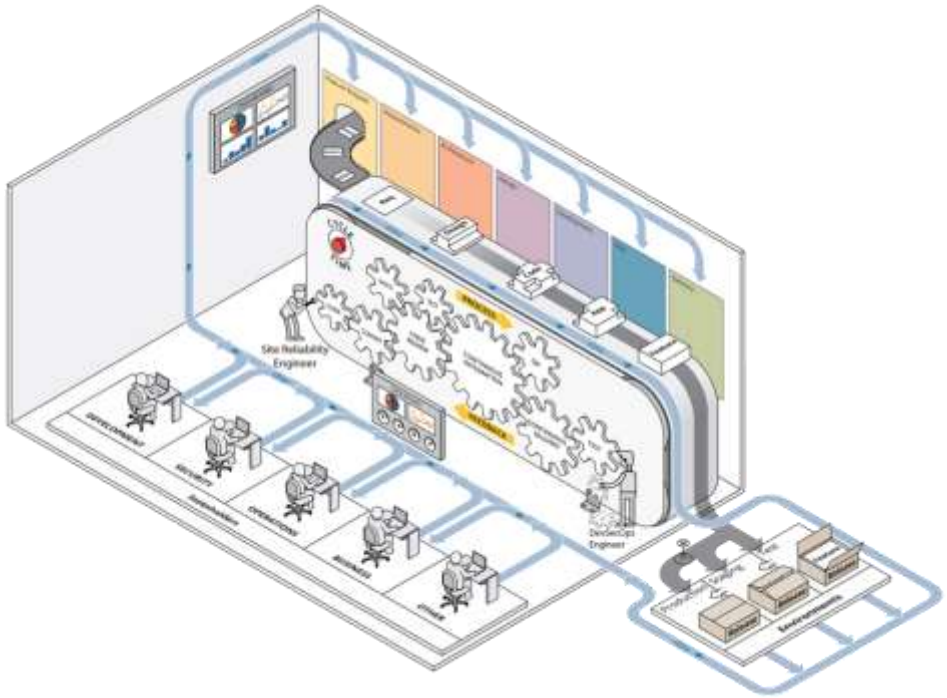
- Trustworthy - No exploitable vulnerabilities exist, either maliciously or unintentionally inserted
- Predictable - When executed, software functions as intended and only as intended
- Timely – Features are delivered as the speed of relevance

Backup

DevSecOps Maturity

What is DevSecOps?

A cultural and engineering practice that breaks down barriers and opens collaboration between development, security, and operations organizations using automation to focus on rapid, frequent delivery of secure infrastructure and software to production. It encompasses intake to release of software and manages those flows predictably, transparently, and with minimal human intervention/effort [1].



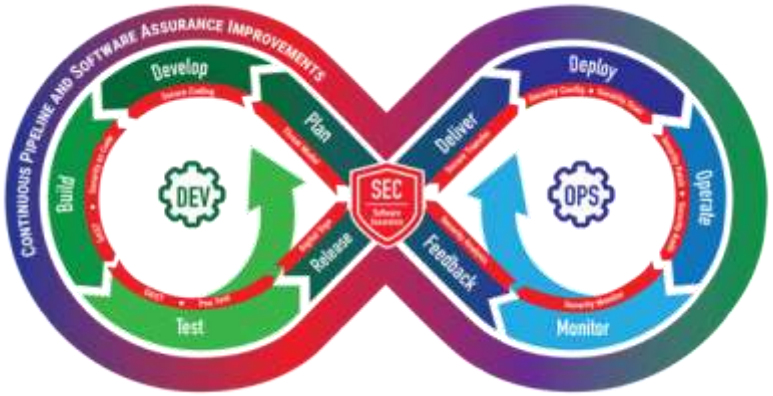
[1] DevSecOps Guide: Standard DevSecOps Platform Framework. U.S. General Services Administration. https://tech.gsa.gov/guides/dev_sec_ops_guide. Accessed 17 May 2021.

DevSecOps Maturity Levels

Term	Documentation
Maturity Level 1	Performed Basic Practices: This represents the minimum set of engineering, security, and operational practices that is required to begin supporting a product under development, even if only performed in an ad-hoc manner with minimal automation, documentation, or process maturity. This level is focused on minimal development, security, and operational hygiene.
Maturity Level 2	Documented/Automated Intermediate Practices: Practices are completed in addition to meeting the level 1 practices. This level represents the transition from manual, ad-hoc practices to the automated and consistent execution of defined processes. This set of practices represents the next evolution of the maturity of the product under development's pipeline by providing the capability needed to automate the practices that are most often executed or produce the most unpredictable results. These practices include defining processes that enable individuals to perform activities in a repeatable manner.
Maturity Level 3	Managed Pipeline Execution: Practices are completed in addition to meeting the level 1 and 2 practices. This level focuses on consistently meeting the information needs of all relevant stakeholders associated with the product under development so that they can make informed decisions as work items progress through a defined process.
Maturity Level 4	Proactive Reviewing and Optimizing DevSecOps: Practices are completed in addition to meeting the level 1-3 practices. This level is focused on reviewing the effectiveness of the system so that corrective actions are taken when necessary, as well as quantitively improving the system's performance as it relates to the consistent development and operation of the product under development.

https://cmu-sei.github.io/DevSecOps-Model/

What is a DevSecOps Pipeline

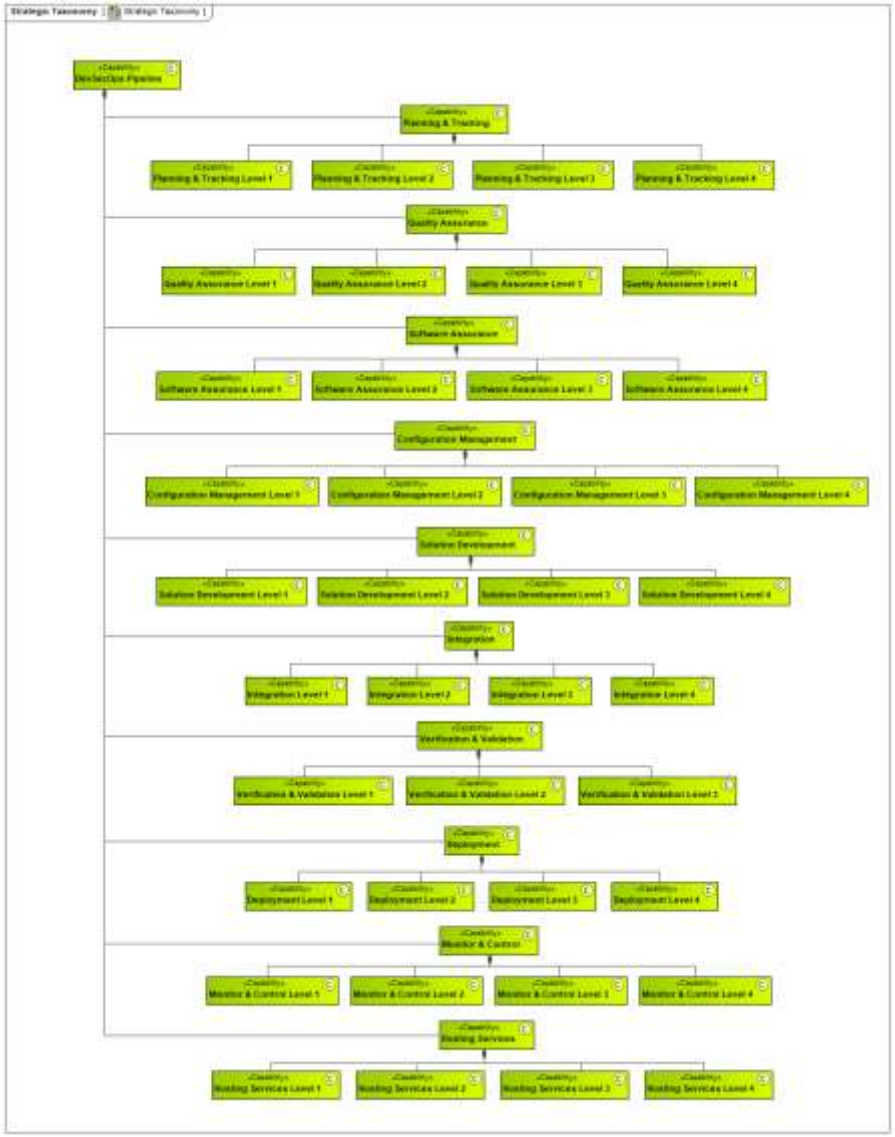


The DevSecOps pipeline is a socio-technical system composed of both software tools and processes. As the capability matures, it seamlessly integrates three traditional factions that sometimes have opposing interests:

- development; which values features
- security, which values defensibility
- operations, which values stability

A DevSecOps pipeline emerges when continuous integration of these three factions is used to meet organizational, project, and team objectives and commitments.

As a DevSecOps system matures, so will its capabilities



https://cmu-sei.github.io/DevSecOps-Model/

Configuration Management

Configuration management is the set of activities used to establish and maintain the integrity of the system and product under development, and associated supporting artifacts, throughout their useful lives.

Different levels of control are appropriate for different supporting artifacts and implementation elements and for different points in time.

- For some supporting artifacts and implementation elements, it may be sufficient to maintain version control of the artifact or element that is traced to a specific instance of the system or product under development in use at a given time, past or present, so that all information related to a given instance, or version, of the system or product under development is known. In this case, all other variations of the artifacts and elements can be discarded as subsequent iterations are generated or updated.
- Other supporting artifacts and implementation elements may require formal configuration, in which case baselines are defined and established at predetermined points in the lifecycle. Baselines, and subsequent changes, are formally reviewed and approved and serve as the basis for future efforts.

The configuration management capability of a system matures as the consistency and completeness of the integrity controls are put in place to capture all supporting artifacts and implementation elements associated with the system and product under development while keeping pace with the DevSecOps pipeline through automation and integration with all aspects of the lifecycle. This includes

- the relationship between artifacts and elements for a given instance, or version, of the system or product under development
- capturing sufficient information to identify and maintain configuration items, even if those who created them are no longer available
- defining the level of control each artifact and element requires based on technical and business needs
- systematically controlling and monitoring changes to configuration items
- enforcement and logging of all required relevant stakeholder reviews and approvals, based on the organization, project, and team policies and procedures.

https://cmu-sei.github.io/DevSecOps-Model/

Configuration Management

Name	Documentation
Configuration Management Level 1	• All supporting artifacts and implementation elements that require configuration control are identified and documented. • The level of configuration control for each supporting artifact and implementation element is defined. • While the configuration management of supporting artifacts may be a fully manual process, an automated version control system, or set of systems, must be in place to track current and historical versions of files used to create implementation elements.
Configuration Management Level 2	• Automated configuration management system(s) are in place for all identified supporting artifacts and implementation elements. • Immutable logging is in place for all changes to configuration items and associated metadata, such as who made the change, when the change occurred, and what was changed. • Changes to the system and product under development are associated with an approved requirement or change request. • All relevant stakeholders are notified when changes to configuration items are requested. • Some integration between the automated version control system used for file tracking and other aspects of the DevSecOps pipeline has occurred in order to enable the automatic triggering of other activities. • The automated version control system traces relationships between test artifacts and requirements, and test results and associated artifacts, to a specific instance of the system or product under development in use at a given time, past or present.
Configuration Management Level 3	• Manage and control the volatility of change. Be able to identify impacted supporting artifacts and implementation elements a given change request will impact. • Use automatic discovery tools to scan current instance of system and product under development, and associated configurations, to identify mismatches between current instance and approved versions under configuration management in order to ensure integrity of the instantiated instances. Automatically report all mismatches to relevant stakeholders. • The system shall automatically maintain an audit trail of all system configuration changes to include what was changed, who/what changed it, and when the change occurred. • System only allows authorized individuals, or entities, to make specific types of changes to the product under development based on the individual's role, or entity's purpose, and where they are in the DevSecOps pipeline.
Configuration Management Level 4	• Automatically correct any misconfiguration of the currently instantiated system and product under development based on approved supporting artifacts and implementation elements under configuration control. • The system shall monitor user activities and actively identify security-related actions and system configuration changes that are uncharacteristic of the given user and notify relevant stakeholders of the uncharacteristic behavior to validate the change was appropriate and to avoid insider threats. • A fully automated change proposal process is in place, where changes are proposed and automatically routed to relevant stakeholders for approval and implemented by the system.

https://cmu-sei.github.io/DevSecOps-Model/

Deployment

Name	Documentation
Deployment Level 1	• The system can manually recover if a failure occurs in a deployed product, deploying the product at the last known acceptable state.
Deployment Level 2	• A quality criterion for the deployment of the system and product under development is defined. • While monitoring for failures can be a combination of manual and automated detection processes: ○ the system can automatically recover if a failure occurs in a deployed product, deploying the product at the last known acceptable state. ○ the system can automatically recover the product to a previously working state in the event of system failure. ○ the system can track the changes between deployed products and the personnel and reasoning involved in the change.
Deployment Level 3	• Both the system and product under development are fully automated in terms of orchestration and deployment into target environments. • Various release strategies are supported to include canary, Blue-Green, multiple service, batch, rolling, and A/B testing. • The product under development is deployed continuously, supported by sufficient automation in which no human intervention is required to release the product to its users. • The system shall automatically collect the necessary data to monitor the system and product under development for failures and quality issues and alert relevant stakeholders when corrective actions are required. • In the event that a failure or cancellation occurs during deployment of the product or system, the system will automatically restore a the most recent working version. • Automated updating or patching of software used by the system. Patches are rolled out automatically to the various parts of the system.
Deployment Level 4	• Continuous improvement of the testing procedures is performed based on the data collected from the system and product under development tests. • The system shall automatically identify and track when the defined quality criteria have not been met and the automated quality controls have been bypassed. All relevant stakeholders will be automatically notified, and the noncompliance issue will be tracked to closure.

Deployment is the set of processes related to the delivery or release of the product under development into the environment in which users of the product interact with the product.

The deployment capabilities of the system, along with disaster recovery, speed, and accuracy, mature with increased levels of automation and advanced rollback and release functionality.

https://cmu-sei.github.io/DevSecOps-Model/

Hosting Services

Hosting services are made up of the underlying infrastructure and platforms that both the system and product under development operate upon. This includes the various cloud providers, on premises bare-metal and virtualization, networks, and other software as a service (SaaS) that is utilized along with the management, configuration, access control, ownership, and personnel involved.

Name	Documentation
Hosting Services Level 1	• The hosting services adequately support the scalability, reliability, regulatory, and security requirements to operate, maintain, and build an organization's product. • The hosting services provide compatibility with the testing frameworks and tools utilized throughout system and product development lifecycles.
Hosting Services Level 2	• Logs from hosting services are aggregated, auditable, and analyzable. • System transaction logs are available and immutable. • Performance metrics can be visualized and analyzed for hardware, software, database, and network components. • Role-based access control is utilized throughout. • All information collected uses proper techniques to mitigate privacy and sensitivity concerns, and can be properly disposed of when necessary. • All configuration items are identified and resources are planned and executed in order to maintain configuration integrity of the given item. • Disaster recovery processes are documented and supported.
Hosting Services Level 3	• The system infrastructure is provisioned using IaC and is automated. • Captured metrics can generate alerts based off of defined values. • The system is able to automatically alert and communicate metrics associated with security risks of the underlying infrastructure to stakeholders so they can manage risk and make decisions regarding risk and impact to software applications. • Automatic upgrading of operating system software and supporting services is in place.
Hosting Services Level 4	• Qualities such as performance, capacity, security, compliance, and risk tolerance are continuously being monitored using automated tools. Results from the automated tools are automatically reported to all relevant stakeholders to ensure the quality of the automated process and to identify and track improvements to quality attributes. • System configuration and performance are continuously being monitored using automated tools to identify and report all anomalies. Results from the automated tools are automatically reported to all relevant stakeholders so they can manage risk and make decisions. • Infrastructure is immutable and can be automatically replaced versus update in place.

https://cmu-sei.github.io/DevSecOps-Model/

Integration

Integration is the process of merging changes from multiple developers made to a single code base.

Integration can be made manually on a periodic basis, typically by a senior or lead engineer, or it can be made continuously by automated processes as individual changes are made to the code base.

In either case the purpose of integration is to assemble a series of changes, merge and deconflict them, build the product, and ensure that it functions as intended and that no change broke the whole product, even if those changes worked in isolation.

Name	Documentation
Integration Level 1	• Documented, repeatable, processes exist that may be manual, automated, or some combination of the two. • Some individual processes (e.g., merging changes) may require expert subjective judgement. • Processes may require manual intervention between phases and/or to coordinate steps between disparate systems. • Some human-human and human-process contact occurs outside the orchestration pipeline. • Process initiation is manual and irregular.
Integration Level 2	• Most individual processes are scripted and repeatable. • Expert subjectivity has been removed from all processes by adopting processes with objective criteria for success. • An orchestrated integration pipeline exists; however, it may not be fully automated. • Some human-human and human-process contact occurs outside the orchestration pipeline. • Integration process initiation is regular whether manual or automated.
Integration Level 3	• All individual processes are scripted and fully automated. • An orchestrated integration pipeline controls all processes from start to finish. • All human-process contact occurs from within the context of the orchestration pipeline (e.g., approvals captured in ticketing system, SCM, etc., and orchestration continues).
Integration Level 4	• The entire integration pipeline is fully automated, requiring no manual intervention. • The entire integration pipeline runs in near real time as changes are committed to the code base. • Alerts, notifications, and results of integration are sent to relevant engineers automatically. • A successfully integrated product is ready for delivery with no additional manual processes required.

https://cmu-sei.github.io/DevSecOps-Model/

Monitor & Control

Monitor and control involves continuously monitoring activities, communicating status, and taking corrective action in order to proactively address issues and to consistently improve performance.

More mature projects automate as much of this as possible.

Appropriate visibility enables timely corrective action to be taken when performance deviates significantly from what was expected.

A deviation is significant if, when left unresolved, it precludes the project from meeting its objectives.

Items that should be monitored include cost, schedule, effort, commitments, risks, data, stakeholder involvement, corrective action progress, and task and work product attributes like size, complexity, weight, form, fit, or function.

Name	Documentation
Monitor & Control Level 1	• All supporting artifacts and implementation elements that require monitoring and control are identified and documented. • The level of monitoring and control for each supporting artifact and implementation element is defined. • A policy and plan for planning and performing the monitor and control capability is established and maintained. • The work products of the monitor and control capability are placed under appropriate levels of control.
Monitor & Control Level 2	• The people performing or supporting the monitor and control capability are trained as needed. • Automated monitor and control system(s) are in place for all identified supporting artifacts and implementation elements. • Automated collection of work products, measures, and measurement results are in place. • Automated comparison of actual measurements to expected measurements is performed, and deviations are quantified. • Automated alerting when significant deviations occur.
Monitor & Control Level 3	• The relevant stakeholders of the monitor and control capability are identified, involved, and are obtaining the information they need to make decisions. • Sharing of monitor and control information to relevant stakeholders is automated. • Stakeholders can tailor the visualizations of the information provided to meet their needs.
Monitor & Control Level 4	• The monitor and control capability is itself subject to being monitored and controlled, and corrective action is taken when necessary. • Automated collection of monitor and control capability work products, measures, measurement results, and improvement information, including records of significant deviation, criteria for significant deviation, and corrective action results, are in place. • Root causes of defects and other problems in the monitor and control capability are identified and corrected. • Monitor and control capability is itself subject to continuous improvement.

https://cmu-sei.github.io/DevSecOps-Model/

Planning & Tracking

Planning and tracking is the set of practices used to define tasks and activities, along with the resources needed to perform them and achieve an objective or commitment, and track progress, or lack thereof, towards achieving the given objective.

It provides the mechanisms required to inform relevant stakeholders where an effort currently is within the process and whether it is on track to provide the expected outcomes.

These mechanisms allow relevant stakeholders to determine what has been accomplished and what adjustments or corrective actions need to occur to account for impediments and other unforeseen issues. Ideally, impediments and issues are proactively identified and addressed.

Practices include documenting activities and breaking them down into actionable work to which resources can be assigned, capturing dependence, forecasting, mapping work to requirements, collecting data, tracking progress to commitments, and reporting status.

The planning and tracking capability of a system matures as the automation and integration of associated practices increases.

https://cmu-sei.github.io/DevSecOps-Model/

Planning & Tracking

Name	Documentation
Planning & Tracking Level 1	Manual practices are used, with possible use of some rudimentary tools, that collect and store information used to track and report status and outputs from planning and tracking activities.
Planning & Tracking Level 2	• Planning and tracking tools are used to define tasks and activities, along with the resources needed to perform them and achieve an objective or commitment, and track progress, or lack thereof, towards achieving the given objective. • The tools provide the ability to capture and associate planning and tracking metadata, such as estimates, assumptions, prioritization, assignment, status, commitments, assets, association to implementation elements and supporting artifacts, and agreements. Metadata may consist of mostly manually collected information, with minimal automation. • Automated visualization techniques are used to organize activities, understand dependencies, coordinate multiteam efforts, and road map future commitments. The automated system is used to share project plans and status of current activities with relevant stakeholders.
Planning & Tracking Level 3	• The planning and tracking tools are able to coordinate multiple value streams at the organizational level. Planning and tracking activities are integrated to include both technical and non-technical activities, such as quality assurance, documentation, testing, and configuration management. Dependencies between technical and non-technical activities can be visualized in order to coordinate efforts and identify issues. • Metadata is used to support estimation, projections, and what-if scenario simulations. Organizations, projects, and teams are able to customize metadata, and associated use, in order to meet relevant stakeholder needs. • The planning and tracking tools are integrated with other tools in order to automatically collect metadata associated with various value stream activities. This includes defect, issues, and noncompliance efforts as they are automatically discovered and subsequently addressed and tracked to closure and asset management. • Automated stakeholder notification and status reporting, and associated visualizations, are used to notify relevant stakeholders of changes to plans or commitments, status of current activities, deviations from defined thresholds, and asset renewals and maintenance.
Planning & Tracking Level 4	Data is used to • apply statistical analytical methods to planning and tracking practices in order to improve and optimize the team's, project's, and organization's ability to meet objectives and commitments • provide objective, quantitative status to relevant stakeholders • automatically generate tasking and execute processes based on plan

https://cmu-sei.github.io/DevSecOps-Model/

Quality Assurance

Quality assurance is a set of independent activities (i.e., free from technical, managerial, and financial influences, intentional or unintentional) designed to provide confidence to relevant stakeholders that the DevSecOps processes and tools are appropriate for and produce products and services of suitable quality for their intended purposes.

It assumes that the organization's, team's, and project's policies and procedures have been defined based on all relevant stakeholder needs, which will result in a value stream that consistently produces products and services that meet all relevant stakeholder expectations.

The quality assurance capability of a system matures as its ability to assess adherence to, and the adequacy of the defined policies and procedures, improves.

https://cmu-sei.github.io/DevSecOps-Model/

Quality Assurance

Name	Documentation
Quality Assurance Level 1	• All relevant stakeholders associated with the products and services associated with the product under development and the system that supports it have been identified. • All relevant stakeholder expectations and regulatory requirements are documented. • Policies and procedures are developed and documented to describe how the DevSecOps processes and tools are required to be used in order to meet all relevant stakeholder requirements. • Documented policies and procedures may use a traditional document-centered approach, and dissemination may be a manual process. • All current policies and procedures are readily available to all personnel.
Quality Assurance Level 2	• Automated tools are used to maintain configuration control of policies and procedures. • All relevant stakeholders are automatically notified of changes to policies and procedures. • Independent resources have been identified and a plan exists to review or audit activities that have been defined within the documented policies and procedures. • DevSecOps processes and tools are periodically audited based on the plan to identify noncompliance with policies and procedures and inadequacies regarding the value stream's ability to consistently produce products and services that meet all relevant stakeholders' expectations and regulatory requirements. The audits may be conducted manually, use automation, or a combination of both. • All identified noncompliance and inadequacies are independently documented, reported to relevant stakeholders, and tracked to closure.
Quality Assurance Level 3	• DevSecOps tools are configured to automatically enforce policies and procedures as a product under development progresses through the system. • Automated processes are monitored by an independent resource in order to detect and report noncompliance issues to all relevant stakeholders. • Noncompliance and inadequacy issues identified through automated or manual auditing are documented and tracked to closure using an automated issue tracking system that is consistent with the tools used for all other planning and tracking purposes, in order to integrate all efforts that must be planned and tracked to completion. • All quality assurance tools, such as origin and static analysis tools, are fully integrated into the system's pipeline, and associated policies are automatically enforced as the product under development progresses through the system. • The system automatically monitors and enforces compliance to defined quality criteria as defined for both the product under development and the system regarding the implementation of enhancements and modifications.
Quality Assurance Level 4	• All automated activities are continuously being audit for noncompliance issues through the use of automated tools, with regards to both the system and product under development. • Results from the automated auditing tools are automatically reported to all relevant stakeholders to ensure the quality of the automated auditing process, in addition to tracking noncompliance issues to resolution. • The system automatically identifies and tracks when the defined quality criteria have not been met or the automated quality controls have been bypassed. All relevant stakeholders will be automatically notified and the noncompliance issue will be tracked to closure.

https://cmu-sei.github.io/DevSecOps-Model/

Software Assurance

Software assurance is the level of confidence that software functions only as intended and is free from vulnerabilities, either intentional or unintentionally designed or inserted as part of the software, throughout the full software lifecycle. It consists of two independent, but interrelated, assertions:

1. The software functions only as intended. It exhibits only functionality intended by its design and does not exhibit functionality not intended.
2. The software is free from vulnerabilities, whether intentionally or unintentionally present in the software, including software incorporated into the final system.

It is the responsibility of the DevSecOps system to ensure that software that meets the organization's threshold for software assurance is allowed to be deployed and operated.

https://cmu-sei.github.io/DevSecOps-Model/

Software Assurance

Name	Documentation
Software Assurance Level 1	• All relevant stakeholders and expectations with regards to the products and services associated with the product under development and the system that supports it have been identified. • System functional and nonfunctional requirements are documented. • A comprehensive software bill of materials (SBOM) is compiled detailing all components that make up the DevSecOps system. • All relevant system constraints and regulatory requirements are documented. • Software assurance processes and tools are inventoried, and policies and procedures are written setting out how they are to be used to meet assurance requirements. • Documented policies and procedures may use a traditional document-centered approach, and dissemination may be a manual process.
Software Assurance Level 2	• Software assurance related to DevSecOps metrics are defined and collected. • Baseline and threshold levels for software assurance are established. • Metrics are tracked over time and made available to all stakeholders as needed. • Results of system functional testing are collected and periodically analyzed. • Known vulnerabilities in all components that make up the DevSecOps system are periodically collected and analyzed. • Processes and policies are in place to periodically compare present metrics to past and make adjustments as necessary. • Processes and policies are in place and reviewed periodically. • Reports are reviewed from all software assurance products. • Processes and policies are in place to identify when the level of software assurance implied by captured metrics and reports exceeds the organization's threshold and to make adjustments as necessary.
Software Assurance Level 3	• The organization has established a comprehensive risk analysis and management program. • Software assurance metrics, reporting, and analysis are incorporated into the risk management process. • Results of the risk management process are incorporated into software assurance policies and procedures. • Software assurance metrics and thresholds are periodically updated as a result of risk management activities. • The organization prioritizes software assurance tasks based on the level of risk to the organization.
Software Assurance Level 4	• All software assurance tools, or as many as are feasible, are run continuously and reports are disseminated automatically to all relevant stakeholders. • Software that fails to meet the organization's software assurance thresholds is automatically prevented from being delivered or deployed. • Automated procedures are in place to remediate software assurance issues found within the operating DevSecOps system.

https://cmu-sei.github.io/DevSecOps-Model/

Solution Development

Solutions development determines the best way of satisfying the requirements to achieve an outcome. Its goals are to evaluate baseline requirements and alternative solutions to achieve them, select the optimum solution, and create a specification for the solution.

Each development value stream develops one or more solutions, which are products, services, or systems delivered to the customer, whether internal or external to the enterprise.

Name	Documentation
Solution Development Level 1	• All development activities and tools have been identified and documented. • Tools are provided to enable users to edit, compile, and review source code. • The ability for developers to trace links between requirements, architectural elements, and implementation elements is provided. • A repository for all requirements and associated metadata is provided. • Manual processes for assuring security and privacy compliance are defined and used. • Processes for transitioning between development components are defined and used. • Individual processes are scripted and repeatable. • Processes may require manual intervention between phases and/or to coordinate steps. • Process initiation may be manual or automated.
Solution Development Level 2	• Transitions between implementation elements, and supporting artifacts, are automated, possibly manually triggered. • Secure coding practices and development coding standards are identified and documented. • Traceability of software code origins is provided to provide a SBOM and verify use of most recent third-party components.
Solution Development Level 3	• Transitions between development components are fully automated, either triggered on a periodic schedule or automatically triggered based upon completion of another component's activity. • Determination of requirement feasibility and validation analysis is supported. • Model-based software engineering in order to provide continuous, iterative, and traceable requirements model is supported. • Policy as code (e.g., Security Technical Implementation Guide (STIG) enforcement) is supported.
Solution Development Level 4	• Transitions between development components is performed continuously without human intervention. • Code commits are continuously audited, with alerts to relevant stakeholders. • "Digital twin" modeling of the production system is enabled. • Advanced analysis to ensure compliance is supported.

https://cmu-sei.github.io/DevSecOps-Model/

Verification & Validation

Verification and validation is the set of activities that provides evidence that the system or application under development has met the requirements and criteria that are expected.

The scope of verification and validation includes the general realm of testing, verifying, and validation activities and matures as automation, feedback, and integration with other elements increase.

Name	Documentation
Verification & Validation Level 1	• All relevant stakeholders, with regards to the products and services associated with the product under development and the system that supports it, have been identified. • All testing cases, procedures, and their artifacts are configured, stored, and maintained for a given instance of a product under development. • The system and product under development support the necessary technologies to execute tests.
Verification & Validation Level 2	• Automated tools are used to trace tests to requirements. • Automated tools are used to trace tests cases and artifacts to specific versions of a product under development. • Automated tools are used to configure, store, and execute tests. • Test coverage reports are generated and captured for a specific instance of the system or product under development. • Tests are performed across multiple phases of the software lifecycle, such as development, test, and operations, providing feedback continuously. • Security patching is automatically tested, resulting in automated report generation and delivery. • Both functional and nonfunction tests are manually or automatically executed.
Verification & Validation Level 3	• Tests are executed automatically using a continuous integration technique. • An MBSE approach is used to plan and execute testing of the system and product under development. • The system and product under development automatically execute quality tests that either pass or fail the appropriate component under test based on quality metrics for any change being made. Appropriate monitoring of the system and product under development enforces the quality metrics. • The system provides the necessary environment to perform advanced security testing such as fuzz and penetration testing activities.

https://cmu-sei.github.io/DevSecOps-Model/

Example Agile/DevSecOps Practices Assessment Findings

Objective of Assessment

Provide awareness of what practices are already in place, based on a holistic set of Agile and DevSecOps requirements, and to identify practices that are not applicable.

Identify pain points, barriers to collaboration, and technological barriers with respect to DevSecOps and Agile principles.

Propose areas of improvement and strategy regarding implementation of software development tools, and methodologies that seem applicable to the program's mission set.

The goal is not to establish an ideal Agile or DevSecOps state.

The goal is to identify actions that the program, and those in their orbit can take to make themselves a more lethal/effective/efficient/useful/professional/… software engineering team than they are today.

Areas of Focus

Capabilities:

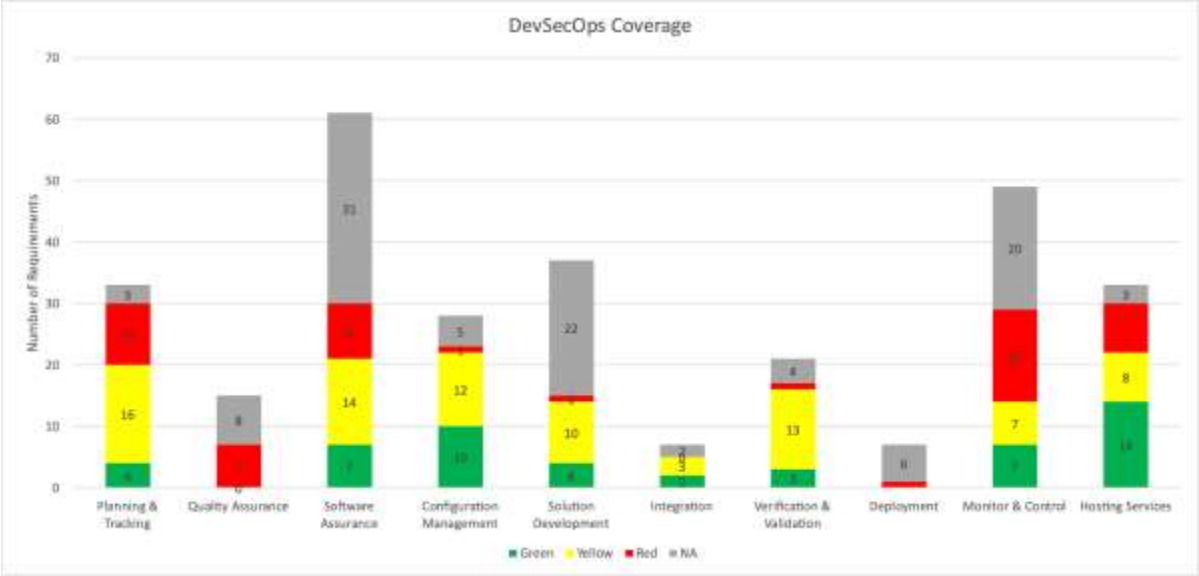
- Configuration Management
- Deployment
- Hosting Services
- Integration
- Monitor & Control
- Planning & Tracking
- Quality Assurance
- Software Assurance
- Solution Development
- Verification & Validation

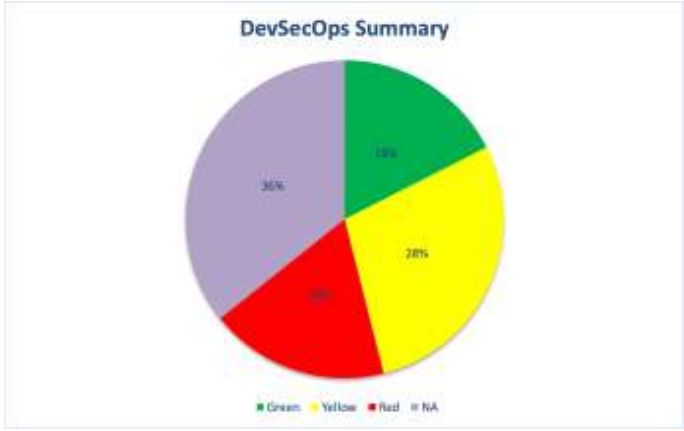
Lifecycle:

- Governance
- Requirements
- Architecture & Design
- Development
- Test
- Delivery
- System Infrastructure

https://cmu-sei.github.io/DevSecOps-Model/

Capability Summary



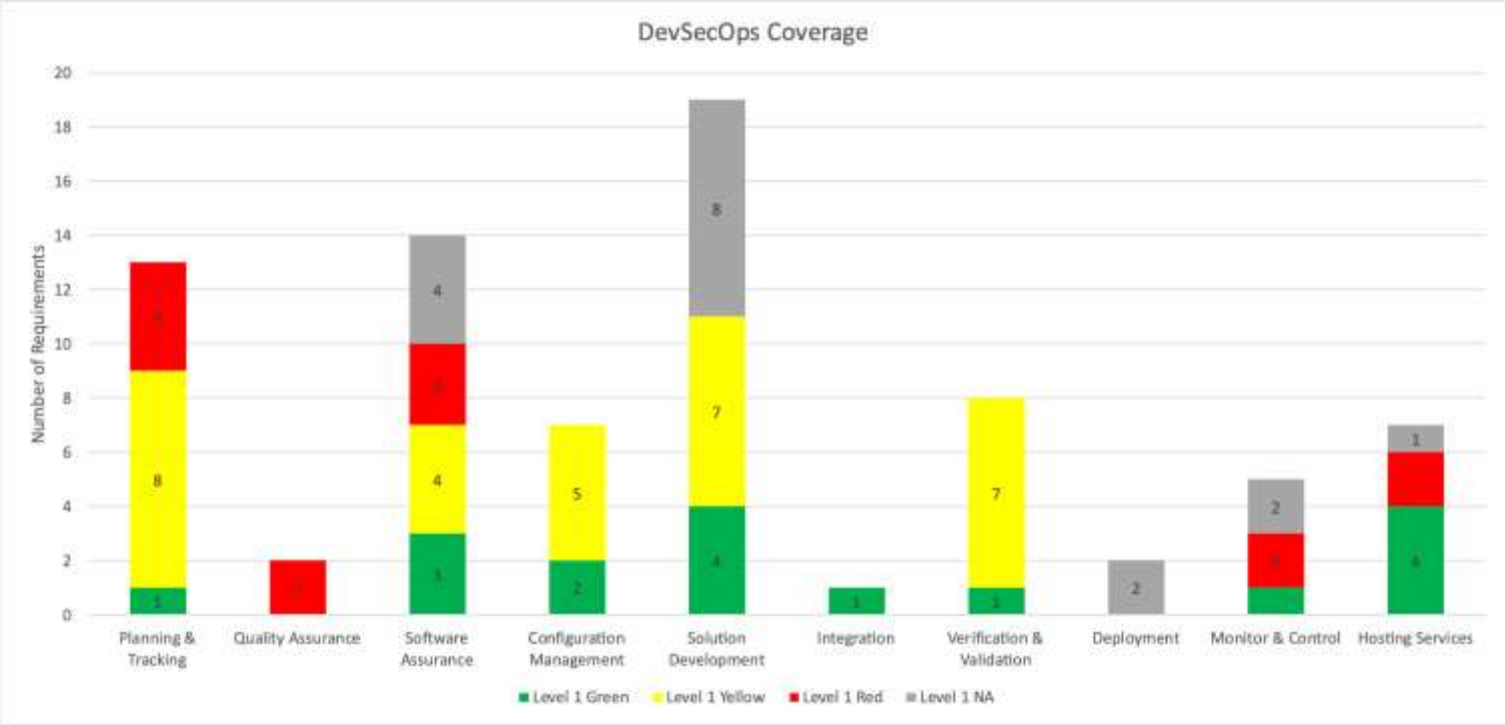


212 Agile/DevSecOps requirements broken into 10 Capabilities and 4 Maturity Levels

Key	Description
Green	Consistently Demonstrated
Yellow	Occasionally Demonstrated
Red	Insufficient Evidence Found
NA	Not Applicable to Program

Level 1

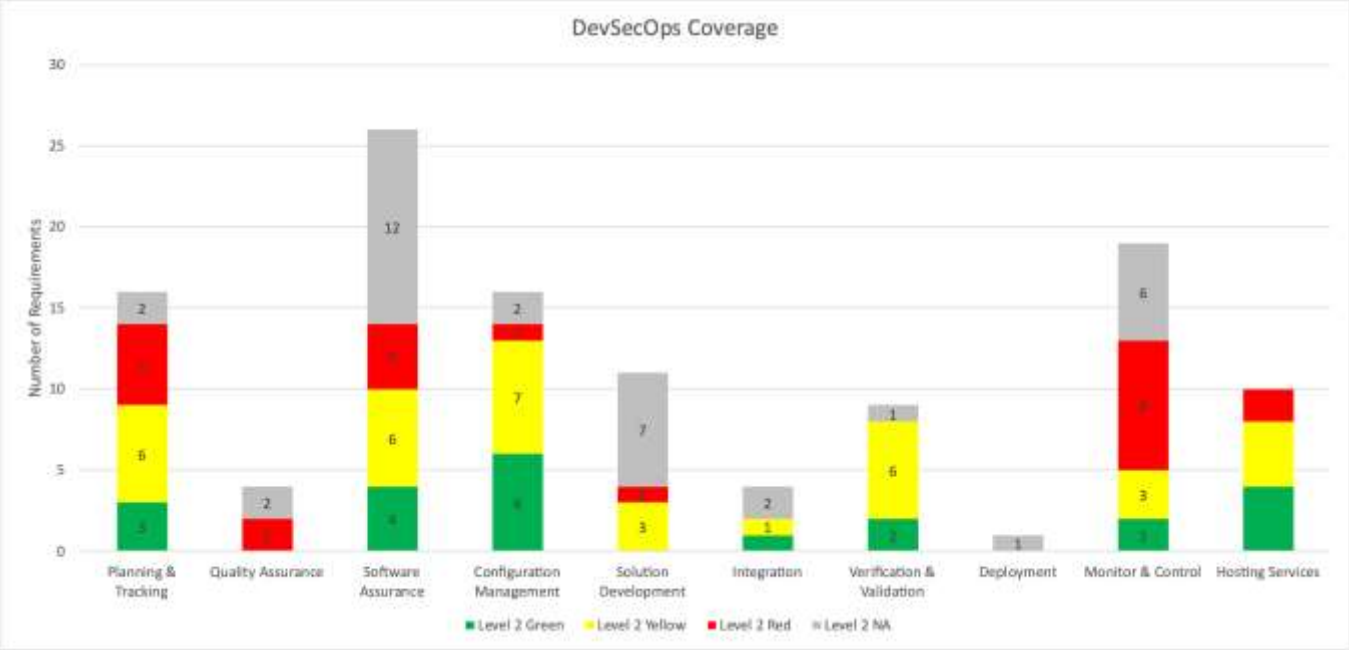
Performed Basic Practices: This represents the minimum set of engineering, security, and operational practices that is required to begin supporting a product under development, even if only performed in an ad-hoc manner with minimal automation, documentation, or process maturity. This level is focused on minimal development, security, and operational hygiene.



63 Maturity Level 1 Agile/DevSecOps requirements broken into 10 Capabilities

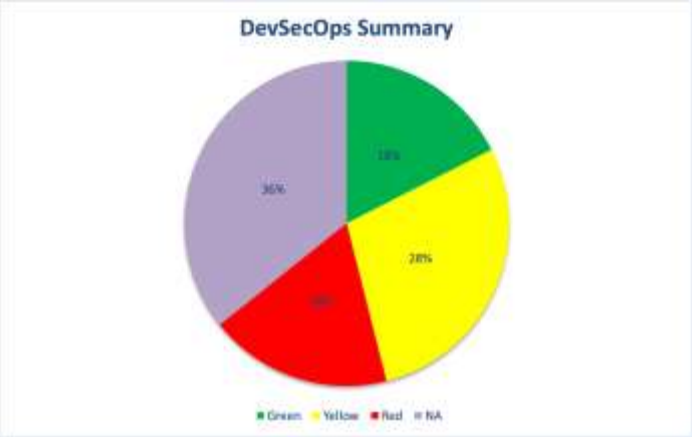
Level 2

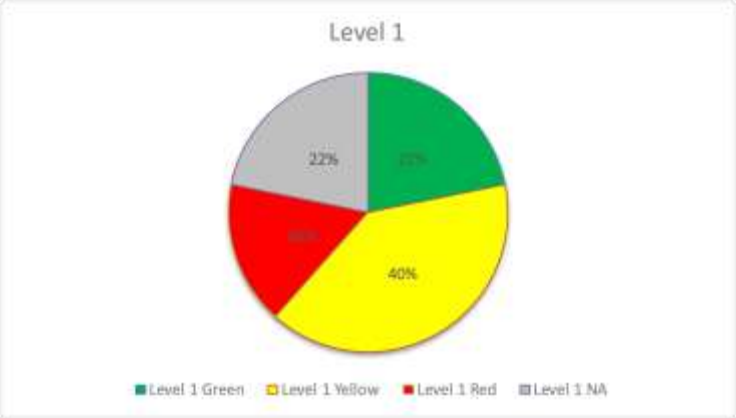
Documented/Automated Intermediate Practices: Practices are completed in addition to meeting the level 1 practices. This level represents the transition from manual, ad-hoc practices to the automated and consistent execution of defined processes. This set of practices represents the next evolution of the maturity of the product under development's pipeline by providing the capability needed to automate the practices that are most often executed or produce the most unpredictable results. These practices include defining processes that enable individuals to perform activities in a repeatable manner.



81 Maturity Level 2 Agile/DevSecOps requirements broken into 10 Capabilities

Capability Distributions





Level 1 – 63 Requirements

Level 2 – 81 Requirements

Level 3 – 58 Requirements

Level 4 – 5 Requirements

Review process

Current Development and Release Processes

- Purpose: identify areas of improvement for development processes

Current development infrastructure and any tooling currently in use by developers

- Purpose: identify technical needs of the organization, including both software development and DevSecOps tools/systems

Current processes and tools that support daily communication and collaboration between various groups and individuals

- Purpose: inform the development and improvement of the organization's development processes.

Approach included interviews throughout the organization and the review of documents/artifacts provided

Summary of Overarching Analysis Themes

- Theme X
- Theme Y
- …
- Theme n

Theme X

Key Observations

Theme X

Potential Courses of Action

Summary Observations