

RESEARCH REVIEW 2022

Addressing DevSecOps Challenges Using Model Based Systems Engineering

Timothy A. Chick
CERT Systems Technical Manager, CMU-Software Engineering Institute
Adjunct Faculty Member, CMU-Institute for Software Research

© 2022

**Carnegie
Mellon
University**
Software
Engineering
Institute

CyLab Carnegie Mellon University
Security and Privacy Institute

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Document Markings

Copyright 2022 Carnegie Mellon University.

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. The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon® and CERT® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM22-0848

Agenda

About DevSecOps

Challenges associated with DevSecOps

- **Challenge 1: Connecting process, practice, and tools**
- **Challenge 2: Cybersecurity of pipeline and product**

Addressing challenges with MBSE

RESEARCH REVIEW 2022

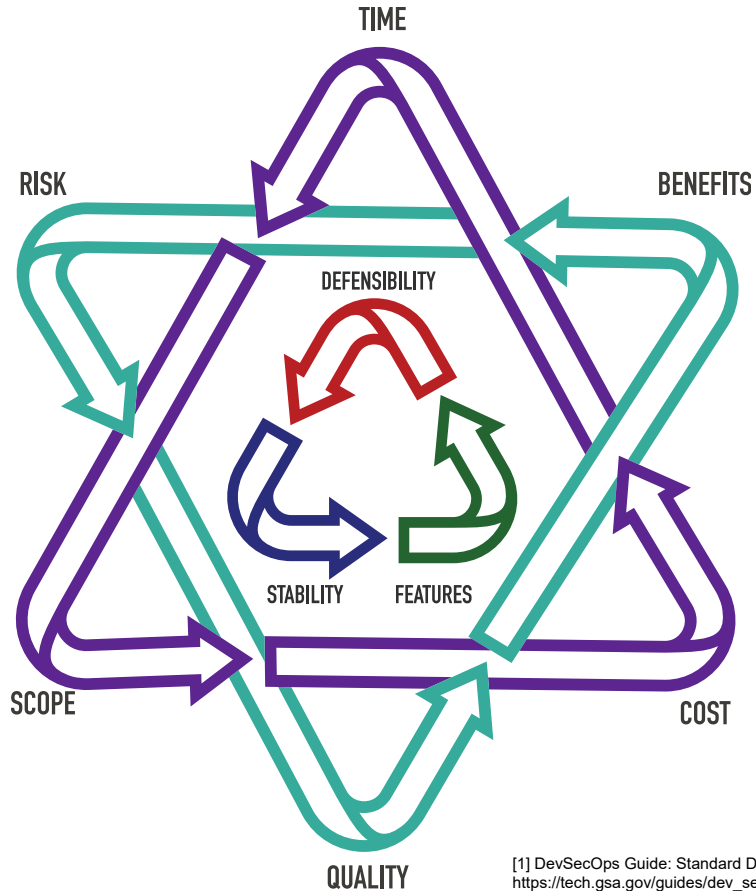
Addressing DevSecOps Challenges Using Model Based Systems Engineering

About DevSecOps

**Carnegie
Mellon
University**
Software
Engineering
Institute

CyLab Carnegie Mellon University
Security and Privacy Institute

DevSecOps: Modern Software Engineering Practices and Tools that Encompass the Full Software 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 Pipeline** attempts to seamlessly integrate “three traditional factions that sometimes have opposing interests:

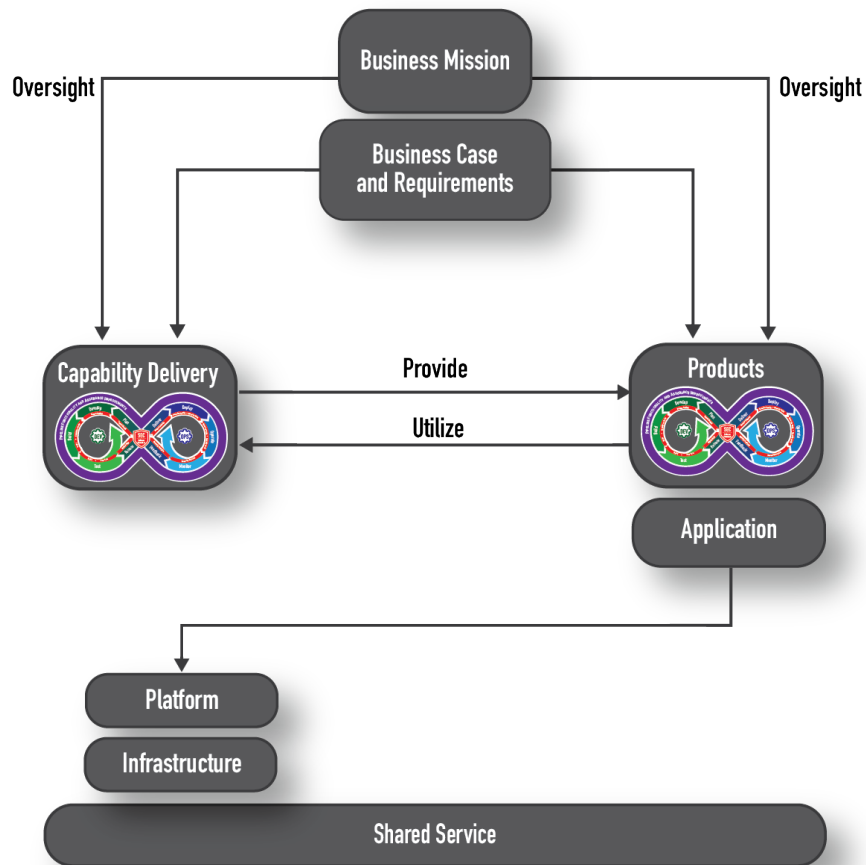
- **development**; which values features;
- **security**, which values defensibility; and
- **operations**, which values stability [2].”

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/>

An Enterprise View



All DevSecOps-oriented enterprises are driven by three concerns:

- **Business Mission** – captures stakeholder needs and channels the whole enterprise in meeting those needs. It answers the questions *Why* and *For Whom* the enterprise exists
- **Capability to Deliver Value** – covers the people, processes, and technology necessary to build, deploy, and operate the enterprise's products
- **Products** – the units of value delivered by the organization. Products utilize the capabilities delivered by the software factory and operational environments.

RESEARCH REVIEW 2022

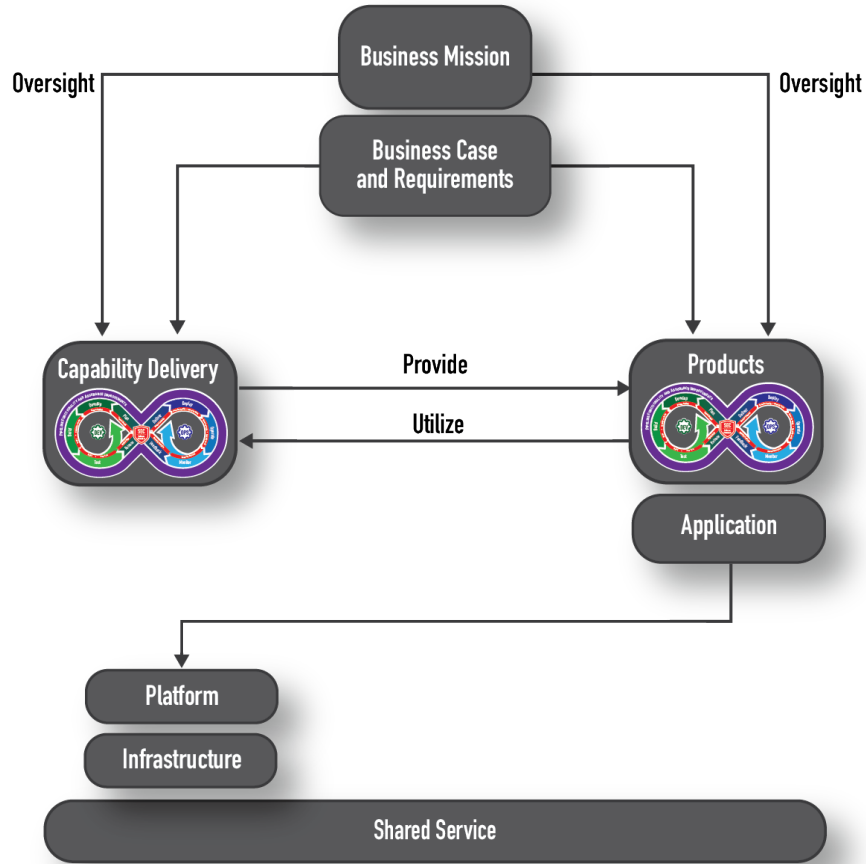
Addressing DevSecOps Challenges Using Model Based Systems Engineering

Challenges Associated with DevSecOps

**Carnegie
Mellon
University**
Software
Engineering
Institute

CyLab Carnegie Mellon University
Security and Privacy Institute

Challenge 1: connecting process, practice, and 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
- Infrastructure and shared services are often maintained across multiple organizations (Cloud for infrastructure, third parties for tools and services, etc.)
- Processes, practices, and tools must evolve to meet the needs of the products being built and operated

Many valid approaches to implementation¹



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

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

The 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...

Many Valid Approaches to Implementation²

- The family of Agile and DevSecOps 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 “doing DevSecOps correctly,” or not.
- To succeed, you must select the correct techniques, regardless of chosen methods, to meet your organization's and customer's goals, objectives, and missions.

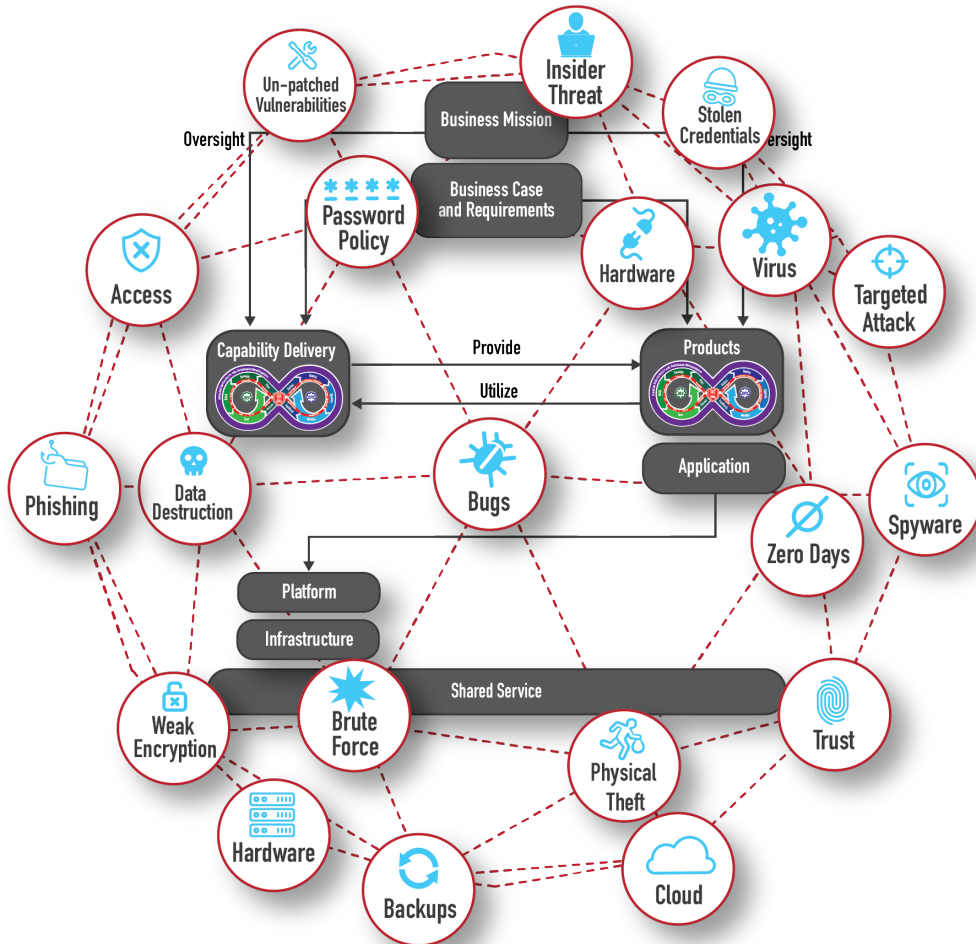
Selecting the Appropriate Techniques

Three Fundamental Factors

1. Identifying **the ability of the organization** to adopt new techniques
 - Successful adoption requires the absorption of associated costs, as well as expending the required time and effort.
2. Determining **the suitability of Agile and DevSecOps practices in the development** of a given product or system
 - Development and product characteristics play a large role in determining the suitability of a particular agile technique.
 - The desired product qualities also play a role in determining appropriate agile technique
3. Determining **the suitability of Agile and DevSecOps 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.

Challenge 2: Cybersecurity of Pipeline and Product



The tight integration of Business Mission, Capability Delivery, and Products, using integrated processes, tools, and people, increases the attack surface of the product under development.

Managing and monitoring all the various parts to ensure the product is built with sufficient cybersecurity and the pipeline is maintained to operate with sufficient cybersecurity is complex.

How do you focus attention to areas of greatest concern for security risks and identify the attack opportunities that could require additional mitigations?

Software Assurance (SwA)

DoD definition:

“the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle, and that the **software functions in the intended manner.**”

[CNSS Instruction No. 4009; DoDi 5200.44 p.12]

SwA Curriculum Model definition:

Application of technologies and processes to achieve a required level of confidence that **software systems and services function in the intended manner**, are free from accidental or intentional vulnerabilities, provide security capabilities appropriate to the threat environment, and recover from intrusions and failures.

[Mead, Nancy; Allen, Julia; Ardis, Mark; Hilburn, Thomas; Kornecki, Andrew; Linger, Richard; & McDonald, James. *Software Assurance Curriculum Project Volume I: Master of Software Assurance Reference Curriculum*. CMU/SEI-2010-TR-005. Software Engineering Institute, Carnegie Mellon University. 2010. <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9415>]

Risk

The perception of risk drives assurance decisions

- Assurance implementation choices (policies, practices, tools, restrictions) are based on the perception of threat and the expected impact should that threat be realized
- Perceptions are primarily based on knowledge about successful attacks
 - the current state of assurance is largely reactive
 - successful organizations learn from attacks and figure out how to react and recover faster and be vigilant in anticipating and detecting attacks
- Misperceptions are failures to recognize threats and impacts – “how could it happen to us?” or “it could not happen here!”

Interactions

Highly connected systems require alignment of risk across all stakeholders and systems otherwise critical threats will be unaddressed (missed, ignored) at different points in the interactions.

- There are costs to addressing assurance which must be balanced against the impact of the risk.
- Risk must also be balanced with other opportunities/needs (performance, reliability, usability, etc.).
- Interactions occur at many technology levels (network, security appliances, architecture, applications, data storage, etc.) and are supported by a wide range of roles.

Trusted Dependencies

Your assurance depends on other people's decisions and the level of trust you place on these dependencies:

- Each dependency represents a risk
- Dependency decisions should be based on a realistic assessment of the threats, impacts, and opportunities represented by an interaction.
- Dependencies are not static and trust relationships should be reviewed to identify changes that warrant reconsideration.
- Using many standardized pieces to build technology applications and infrastructure increases the dependency on other's assurance decisions.

Attacker

There are no perfect protections against attacks.

There exists a broad community of attackers with growing technology capabilities able to compromise the confidentiality, integrity, and availability of any and all of your technology assets, and the attacker profile is constantly changing.

- The attacker uses technology, processes, standards, and practices to craft a compromise (socio-technical responses).
- Attacks are crafted to take advantage of the ways we normally use technology or designed to contrive exceptional situations where defenses are circumvented.

Mitigating Risk with Assurance Cases

Understanding risk is hard!

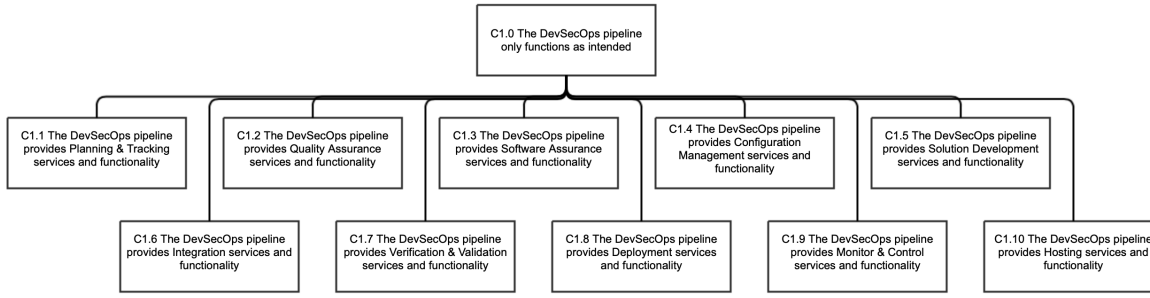
Without being able to quantify, or reason around, the cybersecurity risks associated with your product and DevSecOps pipeline, you will not be able to:

- properly balance between features, defensibility, and stability
- make necessary trade-off choices to achieve your organization's mission and vision in a cost-effective way

An assurance case can be used to reason about the adequacy for both the pipeline and the product.

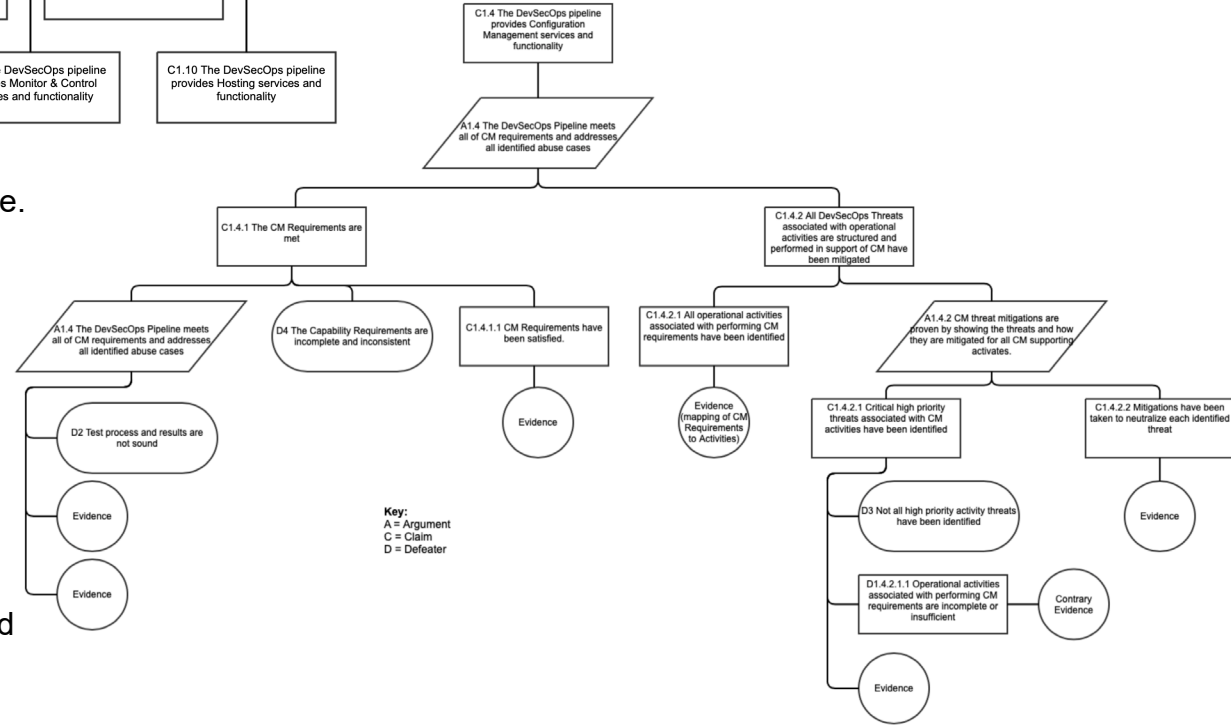
- It is a structured approach used to argue that available evidence supports a given claim
- It provides the organization with the basis for making risk-based choices tied to assuring that the pipeline only functions as intended.
- It provides requirements for automated systems testing, or other evidence collection techniques.
- Actual test results provide the evidence needed to support the assurance claims.

Structuring a DevSecOps Assurance Case



Assurance cases are composed of the following elements:

- **Claims**— “assertions put forward for general acceptance. They are typically statements about a property of the system or some subsystem. Claims that are asserted as true without justification become assumptions and claims supporting an argument are called subclaims [1].”
- **Arguments** – “link the evidence to the claim [1]” by stating the assumption(s) on which the claim and the evidence are built upon.
- **Evidence** – “Evidence that is used as the basis of the justification of the claim. Sources of evidence may include the design, the development process, prior field experience, testing, source code analysis or formal analysis [1].”
- **Defeaters** – “possible reasons for doubting the truth of a claim [2].”



[1] Bloomfield, R. E. and Netkachova, K. Building Blocks for Assurance Cases. Paper presented at the International Symposium on Software Reliability Engineering (ISSRE), 03-11-2014 - 06-11-2014, Naples, Italy.
[2] Goodenough, John B., Charles B. Weinstock, Ari Z. Klein. Toward a Theory of Assurance Case Confidence, CMU/SEI-2012-TR-002 September 2012.

RESEARCH REVIEW 2022

Addressing DevSecOps Challenges Using Model Based Systems Engineering

Addressing Challenges with MBSE

**Carnegie
Mellon
University**
Software
Engineering
Institute

CyLab Carnegie Mellon University
Security and Privacy Institute

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” [1]
 - “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) ... “[2]
- Programs lack sufficient capability to design, build, and implement a DevSecOps continuous integration/continuous delivery (CI/CD) pipeline.
- Current guidance
 - fails to prepare a program to address the full socio-technical aspects of DevSecOps
 - is not definitive and require a considerable amount of interpretation, resulting in:
 - DevSecOps perspectives not being fully integrated in guidance and policy documents
 - programs being unable to perform an analysis of alternative (AoA) in regard to the DevSecOps 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 system have already embraced model-based engineering but have not applied the same techniques to their DevSecOps CI/CD pipelines.

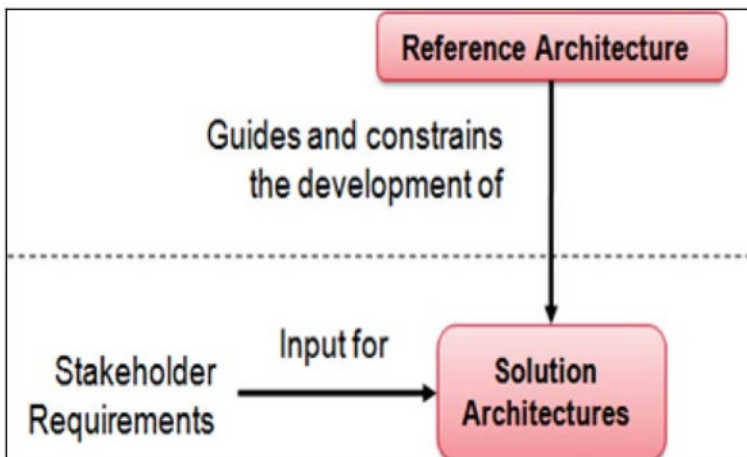
[1] DoD Enterprise DevSecOps Reference Design,

https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf?ver=2019-09-26-115824-583

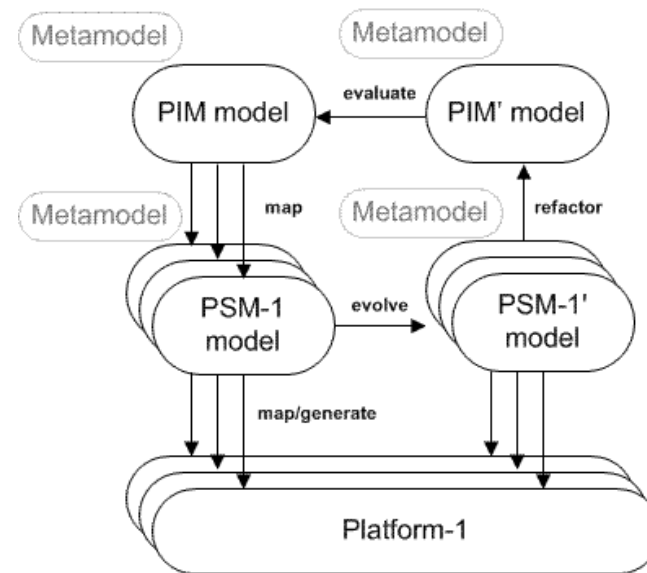
[2] DoD Software Acquisition Pathway Interim Policy and Procedures, <https://aaf.dau.edu/aaf/software/>

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 [1].



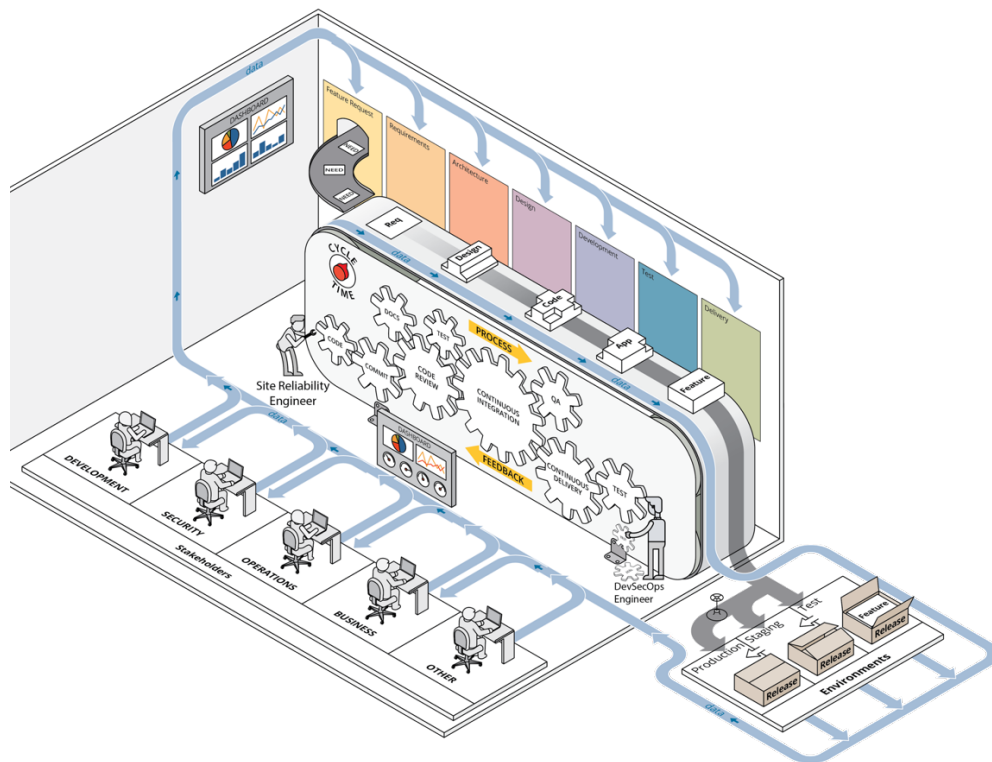
A PIM is a general and reusable model of a solution to a commonly occurring problem in software engineering within a given context and is independent of the specific technological platform used to implement it.



NOTE: PSM = Platform Specific Model

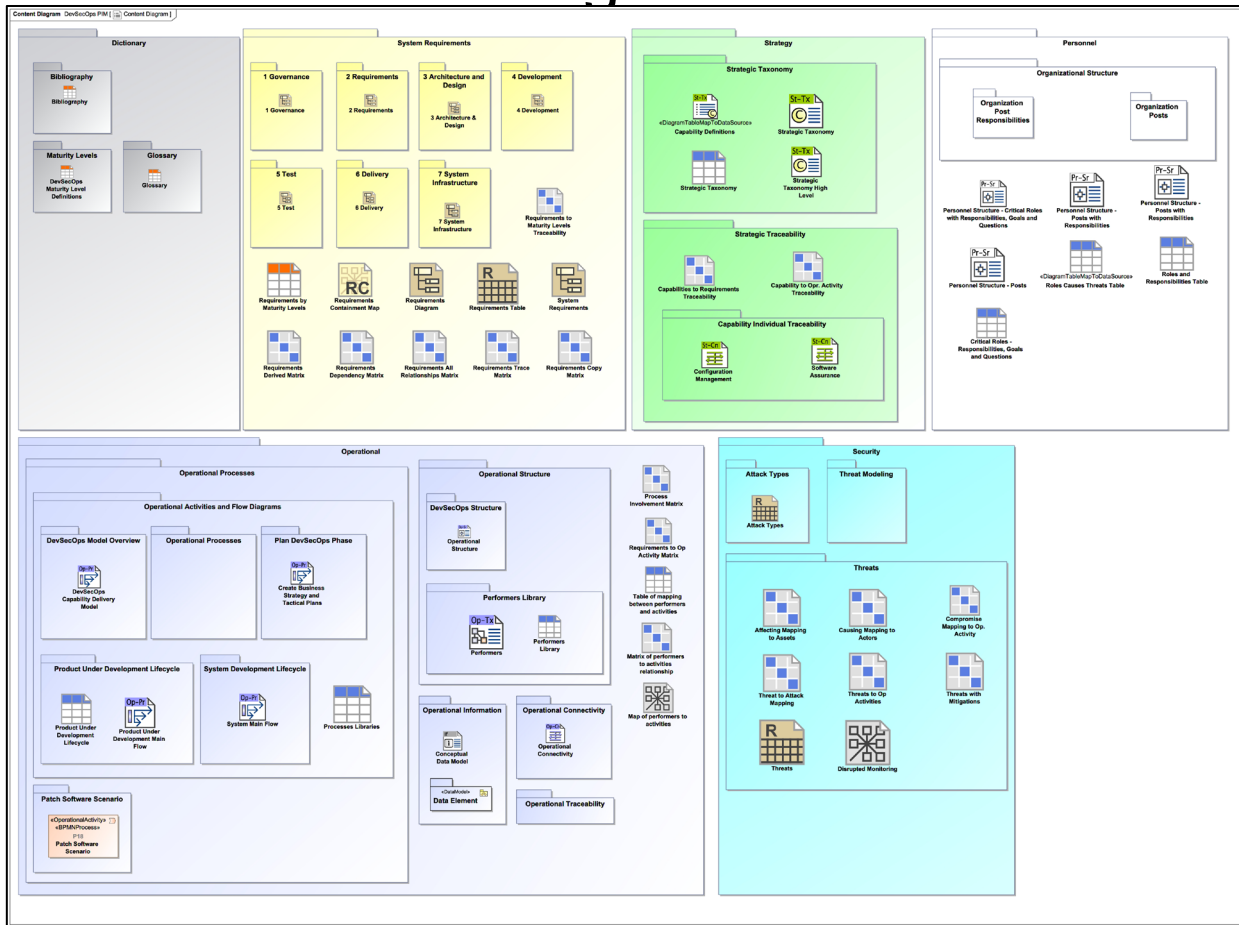
[1] DoD Reference Architecture Description, https://dodcio.defense.gov/Portals/0/Documents/DIEA/Ref_Archi_Description_Final_v1_18Jun10.pdf

DevSecOps Platform Independent Model (PIM)



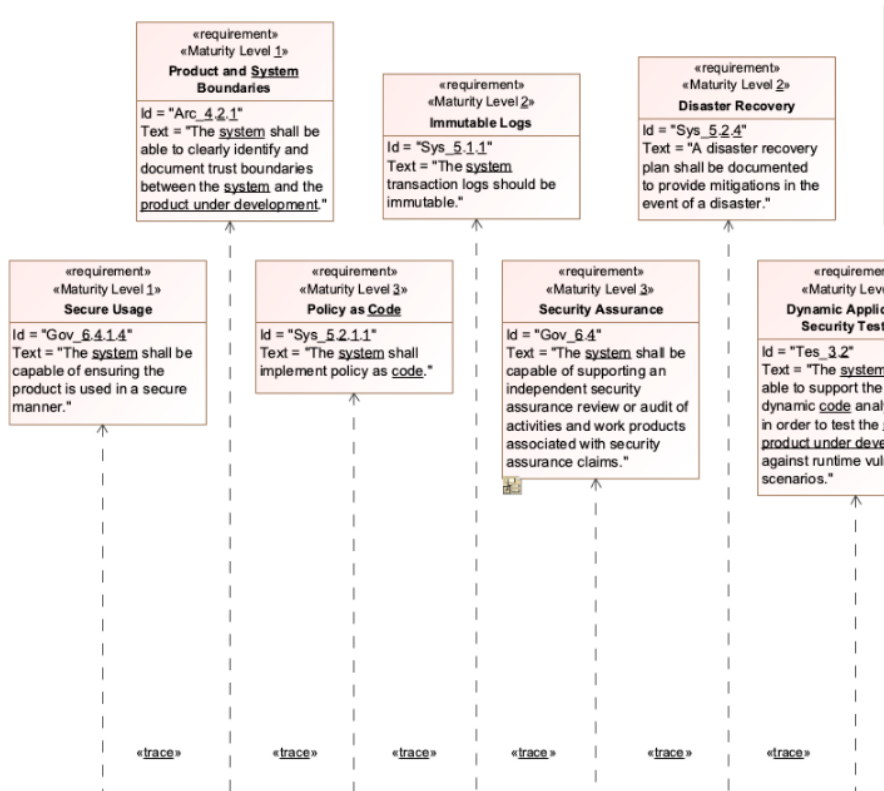
- is an authoritative reference to fully design and execute an integrated Agile and DevSecOps strategy in which all stakeholder needs are addressed
- enables organizations to implement DevSecOps in a secure, safe, and sustainable way in order to fully reap the benefits of flexibility and speed available 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/>

DevSecOps Requirements



All requirements are organized into categories based on logical and functional groupings:

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

Example of Requirements Representation in Diagrams from PIM

DevSecOps Capability/Strategic Viewpoint

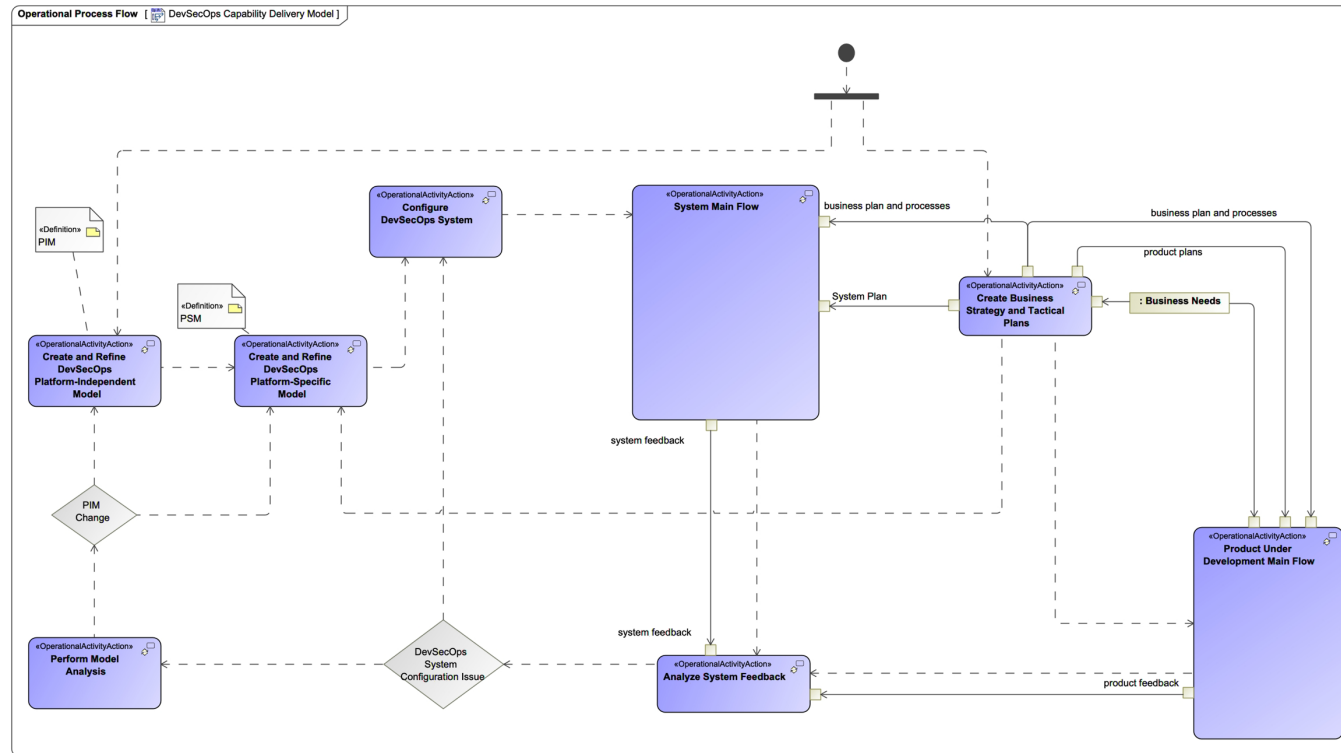
A capability is a high-level concept that describes the ability of a system to achieve or perform a task or a mission.

All requirements in the DevSecOps PIM were allocated to corresponding capabilities.

Legend		
	Trace	
	System Requirements	
	DevSecOps Pipeline [Strategic Taxonomy]	
	Configuration Management	28
	Deployment	10
	Hosting Services	37
	Integration	6
	Monitor & Control	50
	Planning & Tracking	34
	Quality Assurance	17
	Software Assurance	65
	Solution Development	41
	Verification & Validation	25

Legend		System Requirements		2 Requirements		4 Development		5 Test		6 Delivery		7 System Infrastructure	
	Trace	1	1	1	1	1	1	1	1	1	1	1	1
Strategic Taxonomy		1	1	1	1	1	1	1	1	1	1	1	1
DevSecOps Pipeline		1	1	1	1	1	1	1	1	1	1	1	1
Configuration Management		28	3										
Governance		1	1	1	1	1	1	1	1	1	1	1	1
Knowledge Management		1	1	1	1	1	1	1	1	1	1	1	1
Planning and Tracking		1	1	1	1	1	1	1	1	1	1	1	1
Assumptions and Change Manager		1	1	1	1	1	1	1	1	1	1	1	1
Documented Policies		1	1	1	1	1	1	1	1	1	1	1	1
Software Lifecycle		2	2	2	2	2	2	2	2	2	2	2	2
Service and Operations		1	1	1	1	1	1	1	1	1	1	1	1
Maintenance and Agreement		1	1	1	1	1	1	1	1	1	1	1	1
Roles and Responsibilities		1	1	1	1	1	1	1	1	1	1	1	1
Measurement Strategy		1	1	1	1	1	1	1	1	1	1	1	1
Software Certification		1	1	1	1	1	1	1	1	1	1	1	1
System Assurance		1	1	1	1	1	1	1	1	1	1	1	1
System Monitoring and Error		1	1	1	1	1	1	1	1	1	1	1	1
Infrastructure as Code		1	1	1	1	1	1	1	1	1	1	1	1
System Accountability		1	1	1	1	1	1	1	1	1	1	1	1
Permissions Based on Engineering to Product		1	1	1	1	1	1	1	1	1	1	1	1
Document Requirements		1	1	1	1	1	1	1	1	1	1	1	1
Requirement Metrics		1	1	1	1	1	1	1	1	1	1	1	1
Test Association		1	1	1	1	1	1	1	1	1	1	1	1
Definition of Requirements		1	1	1	1	1	1	1	1	1	1	1	1
Planning and Mapping		1	1	1	1	1	1	1	1	1	1	1	1
Architecture Assumptions		1	1	1	1	1	1	1	1	1	1	1	1
Minimum Variable Requirements		1	1	1	1	1	1	1	1	1	1	1	1
Requirements Abstraction		1	1	1	1	1	1	1	1	1	1	1	1
Requirements Prioritization		1	1	1	1	1	1	1	1	1	1	1	1
Requirements Validation		1	1	1	1	1	1	1	1	1	1	1	1
Change Management		1	1	1	1	1	1	1	1	1	1	1	1
Requirements Process		1	1	1	1	1	1	1	1	1	1	1	1
Requirements Authorization		1	1	1	1	1	1	1	1	1	1	1	1
Mapping to Requirements		1	1	1	1	1	1	1	1	1	1	1	1
Mapping to Architecture		1	1	1	1	1	1	1	1	1	1	1	1
Mapping to Tests		1	1	1	1	1	1	1	1	1	1	1	1
Secure Software Development		2	2	2	2	2	2	2	2	2	2	2	2
Origin Analysis		1	1	1	1	1	1	1	1	1	1	1	1
Static Code Analysis		1	1	1	1	1	1	1	1	1	1	1	1
Product Accountability		1	1	1	1	1	1	1	1	1	1	1	1
Product Source Code		1	1	1	1	1	1	1	1	1	1	1	1
Product Artifact Repository		1	1	1	1	1	1	1	1	1	1	1	1
Product Test Repository		1	1	1	1	1	1	1	1	1	1	1	1
Product Software Repository		1	1	1	1	1	1	1	1	1	1	1	1
System Source Code Repository		1	1	1	1	1	1	1	1	1	1	1	1
System Artifact Repository		1	1	1	1	1	1	1	1	1	1	1	1
System Test Repository		1	1	1	1	1	1	1	1	1	1	1	1
Chain of Custody		1	1	1	1	1	1	1	1	1	1	1	1
Immutable Versus Unimmutable		1	1	1	1	1	1	1	1	1	1	1	1
Unauthorized Code		1	1	1	1	1	1	1	1	1	1	1	1
Source Code Editor		1	1	1	1	1	1	1	1	1	1	1	1
Compiler and Interpreter		1	1	1	1	1	1	1	1	1	1	1	1
Build Automation		1	1	1	1	1	1	1	1	1	1	1	1
Debugger		1	1	1	1	1	1	1	1	1	1	1	1
Static Code Integrator		1	1	1	1	1	1	1	1	1	1	1	1
Version Control		1	1	1	1	1	1	1	1	1	1	1	1
Integrated Development Environment		1	1	1	1	1	1	1	1	1	1	1	1
Development Information Repository		1	1	1	1	1	1	1	1	1	1	1	1
Product Simulations		1	1	1	1	1	1	1	1	1	1	1	1
Hardware Emulator		1	1	1	1	1	1	1	1	1	1	1	1
Manual Testing		1	1	1	1	1	1	1	1	1	1	1	1
Manual Test Cases		1	1	1	1	1	1	1	1	1	1	1	1
Manual Test Results		1	1	1	1	1	1	1	1	1	1	1	1
Link Manual Testing to Requirement Association		1	1	1	1	1	1	1	1	1	1	1	1
Automated Testing		1	1	1	1	1	1	1	1	1	1	1	1
Link Automated Test Results to Dynamic Application Security		1	1	1	1	1	1	1	1	1	1	1	1
Dynamic Application Security		1	1	1	1	1	1	1	1	1	1	1	1
Test Tool Compatibility		1	1	1	1	1	1	1	1	1	1	1	1
Quality Evaluation		1	1	1	1	1	1	1	1	1	1	1	1
Code Coverage		1	1	1	1	1	1	1	1	1	1	1	1
Penetration and Fuzz Testing		1	1	1	1	1	1	1	1	1	1	1	1
Testing Information Radiator		1	1	1	1	1	1	1	1	1	1	1	1
Release Management		1	1	1	1	1	1	1	1	1	1	1	1
Internet Technology Service		1	1	1	1	1	1	1	1	1	1	1	1
Product Recovery		1	1	1	1	1	1	1	1	1	1	1	1
System Recovery		1	1	1	1	1	1	1	1	1	1	1	1
Configuration Item Integrity		1	1	1	1	1	1	1	1	1	1	1	1
System's Nonfunctional Requirements		1	1	1	1	1	1	1	1	1	1	1	1
Automated Provisioning		1	1	1	1	1	1	1	1	1	1	1	1
Communication		1	1	1	1	1	1	1	1	1	1	1	1
Information Management		1	1	1	1	1	1	1	1	1	1	1	1
System Logs		1	1	1	1	1	1	1	1	1	1	1	1
Immutable Logs		1	1	1	1	1	1	1	1	1	1	1	1
Log Visualization		1	1	1	1	1	1	1	1	1	1	1	1
Information Stairs		1	1	1	1	1	1	1	1	1	1	1	1
Information		1	1	1	1	1	1	1	1	1	1	1	1
Policy as Code		1	1	1	1	1	1	1	1	1	1	1	1
Vulnerability		1	1	1	1	1	1	1	1	1	1	1	1
Need to Know		1	1	1	1	1	1	1	1	1	1	1	1
Information Security		1	1	1	1	1	1	1	1	1	1	1	1
Disaster Recovery		1	1	1	1	1	1	1	1	1	1	1	1
Infrastructure Configuration		1	1	1	1	1	1	1	1	1	1	1	1
Asset Inventory		1	1	1	1	1	1	1	1	1	1	1	1
Infrastructure as Code		1	1	1	1	1	1	1	1	1	1	1	1

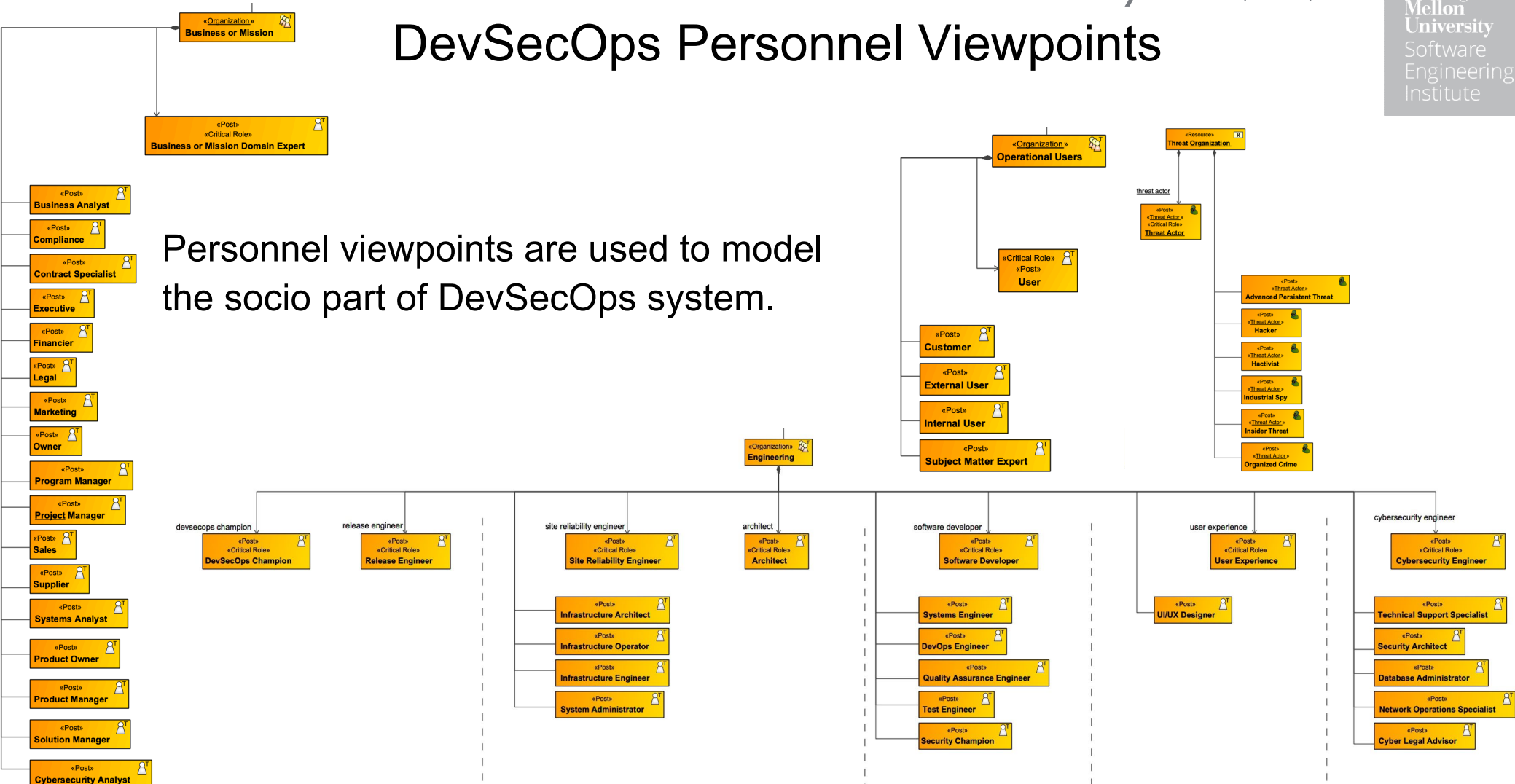
DevSecOps Operational Viewpoints



An operational model for a system describes behavior of the system to conduct enterprise operations. The main operational processes for DevSecOps includes development process for the product, as well as the DevSecOps process itself.

DevSecOps Personnel Viewpoints

Personnel viewpoints are used to model the socio part of DevSecOps system.



sed Systems Engineering

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

Everyone Plays a Role in DevSecOps

Legend		Organization Posts																																															
<ul style="list-style-type: none"> Approves ContributesTo Is Capable To Perform Observes Multiple (one-way) 		Architect	Business Analyst	Business or Mission Domain	Compliance	Contract Specialist	Customer	Cyber Legal Advisor	Cybersecurity Analyst	Cybersecurity Engineer	Database Administrator	DevOps Engineer	DevSecOps Champion	Executive	External User	Financier	Infrastructure Architect	Infrastructure Engineer	Infrastructure Operator	Internal User	Legal	Marketing	Network Operations Specialist	Owner	Product Manager	Product Owner	Program Manager	Project Manager	Quality Assurance Engineer	Release Engineer	Relevant Stakeholders	Sales	Security Architect	Security Champion	Site Reliability Engineer	Software Developer	Solution Manager	Subject Matter Expert	Supplier	System Administrator	Systems Analyst	Systems Engineer	Technical Support Specialist	Test Engineer	UI/UX Designer	User	User Experience		
Operational Activities and Flow Diagrams		93	69	1	3			99	3	58									2					4	14			10	1			10	10					14								29	92		
DevSecOps Model Overview		6	5					5		5																			5			5	5													2	5		
Plan DevSecOps Phase		17	16					15		15																			15			15	16														7	15	
Product Under Development Lifecycle		70	47	1	3			79	3	38									2						4	14			81			80	81							14							20	72	
P2 Product Under Development Main Flow																																																	
P2-1 Plan Product		40	23	2				40	1	18										2				4	11			41			41	41									14					15	40		
P2-2 Develop Product		10	3					10		4																			11			9	10															10	
P2-4 Validate Product		2	1					4																	1				4			5	5															3	
P2-5 Deploy Product								6		2																			6			6	5															2	
P2-6 Operate Product		7																																															
P2-7 Monitor Product		11	11					11		11																			11			11	11																11
P2-8 Manage Contracts, Licenses and Agreements		8																																															
P2-9 Provide Feedback		9																																															
P2-10 Perform Quality Assurance		9																																															
P2-11 Perform Data Analysis		8																																															
P2-12 Monitor Development and Test Environment		7																																															
P2-13 Perform Configuration Management		2																																															
P2-14 Store and Manage Code and Artifacts		8																																															
P2-15 Aggregate, Store and Report on Product Collected Monitoring, Pla		9																																															

Critical Roles are mapped to Operational Activities.

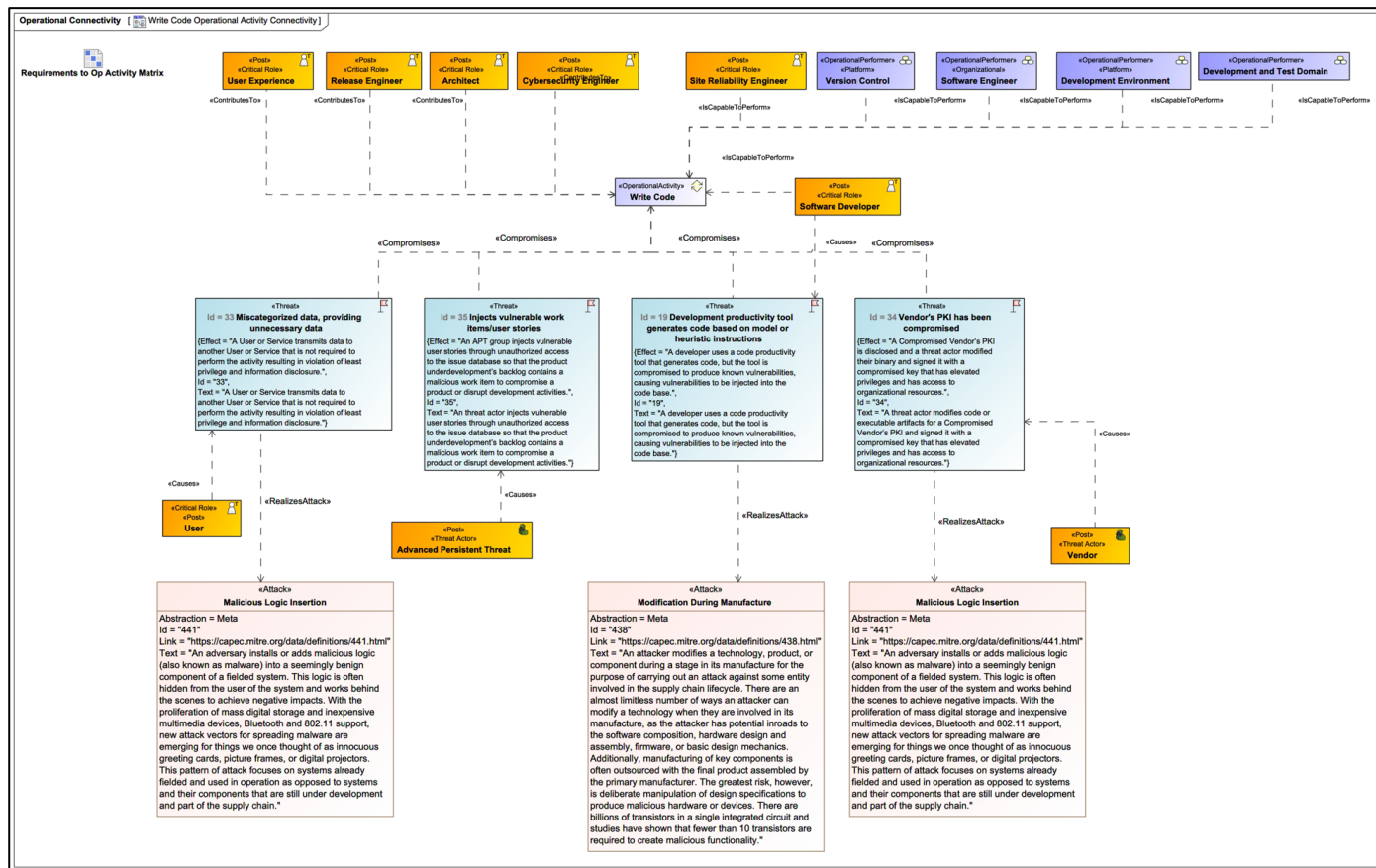
DevSecOps Threat to Operational Activity Matrix

Legend	Product Under Development Lifecycle																											
	P2-1 Plan Product							P2-2 Develop Product							P2-4 Validate Product							System D						
Compromises																												
Threats	1		1		1		2		4		3		5													1	13	3
1 Reduced monitoring																												
2 Disrupted Monitoring																												
3 Unauthorized Access/Modifies logs to divert attribution																												
4 Inadequately configures system logging																												
5 Intentionally misconfiguring																												
6 Intentionally locks out accounts responsible for recovering, inw																												
7 Intentionally misconfiguring 2																												
8 Intentionally misconfiguring 3																												
9 Decrease Document Markings																												
10 Unauthorized Access/Modifies logs to divert attribution 2																												
11 Insert Malicious Code in tool chain, code repository, build art																												
12 Patch Tools in the pipeline																												
13 Slow Approval Process																												
14 Disable the static analysis																												
15 Alters Automated analysis reports																												
16 Configures analyzer in a way that is not best practice																												
17 Results from analysis are disclosed for effect																												
18 Production data (configurations, tokens, accounts, PII, etc) is																												
19 Development productivity tool generates code based on mod																												
20 Tool generates code based on predetermined code snippets																												
21 Perform a code review without sufficient security review crite																												
22 Review is skippedd for items not covered by other defect ider																												
23 Poisoning data while aggregating it																												
24 Requirements exploration and documentation																												
25 Modifies measurement Metrics																												
26 Misleading Contracting Practices																												
27 Misinterpreting the results of the analysis																												
28 Using careless or naive code idioms																												
29 Build tools are misconfigured																												
30 Upstream activity provide false/modified data																												
31 Tampering without data																												
32 Data is intercepted between activities																												
33 Miscategorized data, providing unnecessary data																												
34 Vendor's PKI has been compromised																												
35 Injects vulnerable work items/user stories																												
36 Compromises a vendor																												
37 Injects exploitable/malicious code into upstream open sourc																												
38 Encryption																												

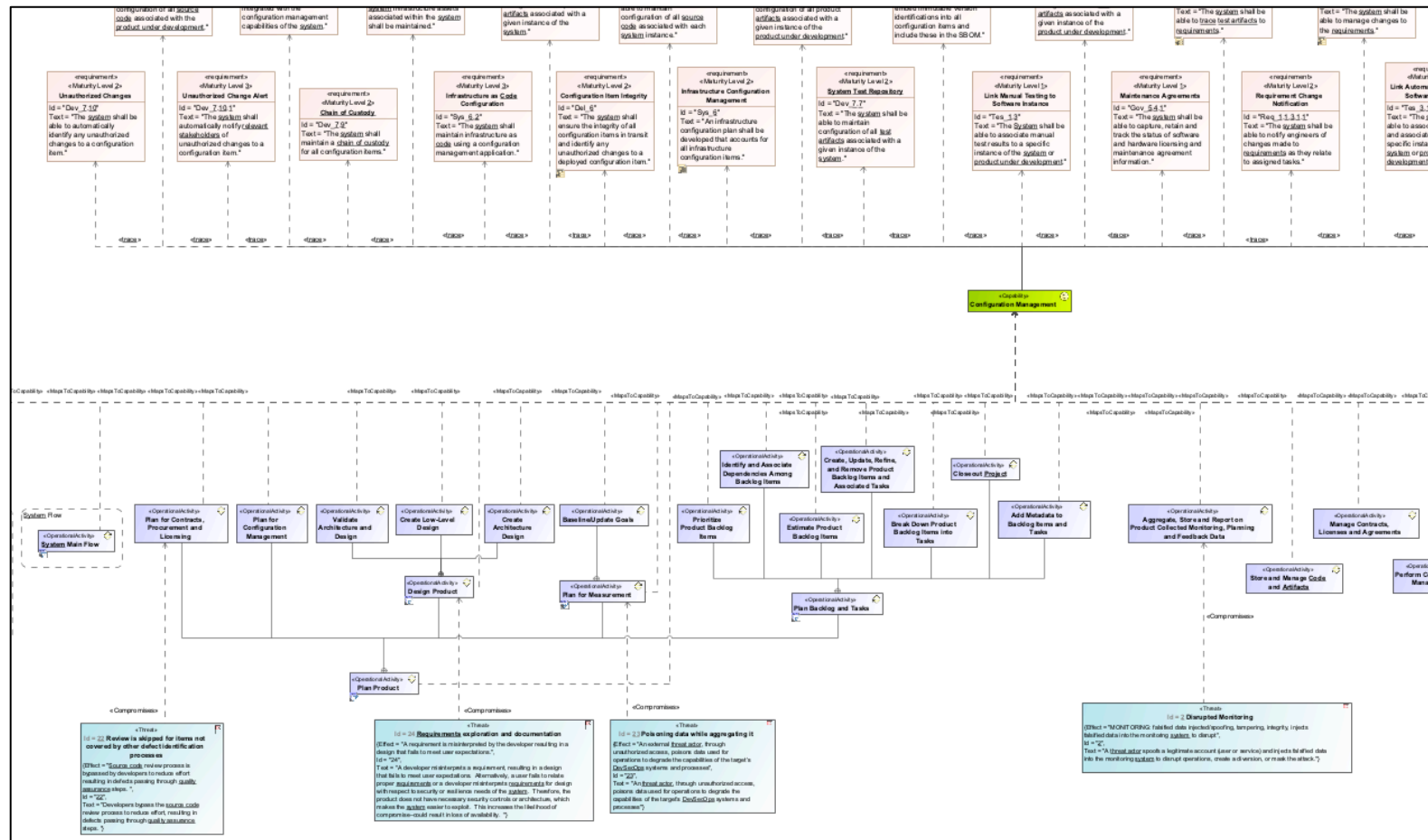
DevSecOps Threats with Attributes

Id	Name	Text	Effect	Compromises	Realized By Attack	Caused By	Mitigated By	Document
1	Reduced monitoring	A <u>threat actor</u> is made aware of a monitoring <u>system</u> 's reduced capacity resulting in regular service outages leaving an open window of opportunity for an unobservable attack.	Reduced or misconfigured monitoring allows for nefarious activity to occur	P2-15 Aggregate, Store and Report on Product Collected Monitoring, Planning and Feedback Data	607 Obstruction	Insider Threat		Much of this was pulled from CAPEC info https://capec.mitre.org/data/definitions/1000
2	Disrupted Monitoring	A <u>threat actor</u> spoofs a legitimate account (user or service) and injects falsified data into the monitoring <u>system</u> to disrupt operations, create a diversion, or mask the attack.	MONITORING: falsified data injected/spoofing, tampering, integrity, injects falsified data into the monitoring <u>system</u> to disrupt	P2-15 Aggregate, Store and Report on Product Collected Monitoring, Planning and Feedback Data	161 Infrastructure Manipulation	Advanced Persistent Threat Insider Threat Architect Cybersecurity Engineer	SC1 Mitigation Strategy 1	Keep at the Meta Level and better explained in the "star"
3	Unauthorized Access/Modifies logs to divert attribution	A <u>threat actor</u> gains unauthorized access to logging data, alters <u>system</u> logs to conceal illicit activity from forensic audits, automated responses and alerts, or to divert attribution.	Logs: insider threat modifies the logs to conceal activity	P2-15 Aggregate, Store and Report on Product Collected Monitoring, Planning and Feedback Data	161 Infrastructure Manipulation	Insider Threat Site Reliability Engineer Cybersecurity Engineer		
4	Inadequately configures <u>system</u> logging	A <u>threat actor</u> has configured the collection of <u>system</u> logs in a way that limits the effectiveness of forensic audit activities.	Accidentally misconfiguring Logging – can't perform forensics work against what is captured	P2-15 Aggregate, Store and Report on Product Collected Monitoring, Planning and Feedback Data	176 Configuration/Environment Manipulation	Software Developer		Could be 161? Most significant improper configuration
5	Intentionally misconfiguring	A <u>threat actor</u> has configured the collection of <u>system</u> logs in a way that limits the effectiveness of forensic audit activities in order to conceal subsequent activities.	Intentionally misconfiguring the <u>system</u>	P2-15 Aggregate, Store and Report on Product Collected Monitoring, Planning and Feedback Data	176 Configuration/Environment Manipulation	Insider Threat		
6	Intentionally locks out accounts responsible for recovering, investigating, or repairing the <u>system</u>	A <u>threat actor</u> spoofs an individual's account in order to create user action logs with the objective of making a targeted user in violation of security policy and reducing the targeted individual's organizational effectiveness.	Targeting individual with the intent that their login is denied, locking out individuals who should have access	P2-15 Aggregate, Store and Report on Product Collected Monitoring, Planning and Feedback Data	212 Functionality Misuse	Insider Threat		Could be a CAPEC - 184 So Attack
		Unit testing is insufficient to cover the <u>requirements</u> and abuse cases. A <u>software</u> or <u>site reliability engineer</u> doesn't		P2-15 Aggregate, Store and Report on Product Collected	176 Configuration/Environment	Software Developer		

Example Threat Modeling Diagram for Write Code Operational Activity



Capturing the Complexity of the DevSecOps System



Example of Threats Traced to Capabilities via Operational Activities

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

Summary



The use of model based systems engineering in the design, implementation, and sustainment of your DevSecOps socio-technical system will assist you in building 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.

Our Team



Timothy A. Chick
Systems Team Technical Manager, CERT Division



Brent Frye
Member of the Technical Staff, CERT Division



Lyndsi Hughes
Systems Engineer, CERT Division



Mary Popeck
Senior Solutions Engineer, CERT Division



Aaron Reffett
MTS - Senior Engineer, CERT Division



Natasha Shevchenko
Senior Member of the Technical Staff, CERT Division



Carol Woody
Principal Researcher, CERT Division



Joseph Yankel
MTS - Senior Engineer, Software Solutions Division

Contact Us

To learn more or to work with us, reach out to info@sei.cmu.edu.

RESEARCH REVIEW 2022

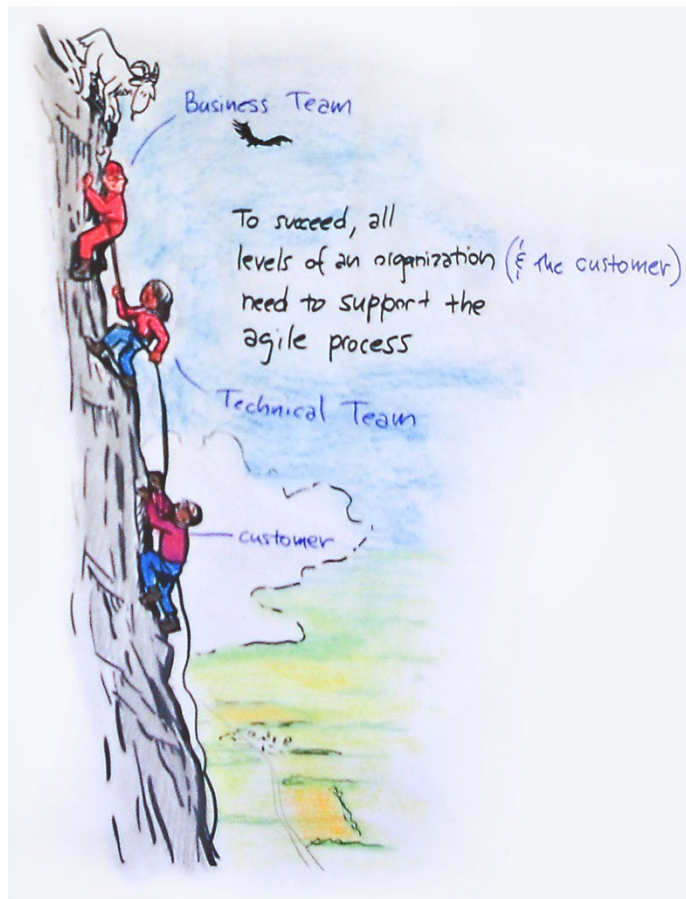
Addressing DevSecOps Challenges using Model Based Systems Engineering

Agile and DevSecOps Principles

**Carnegie
Mellon
University**
Software
Engineering
Institute

CyLab Carnegie Mellon University
Security and Privacy Institute

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 Principles₁

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 Principles₂

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.**

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.