Secure Programming

A.A. 2022/2023

Corso di Laurea in Ingegneria delle Telecomnicazioni

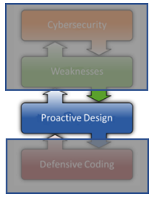
H. Architecture & Processes 2

Paolo Ottolino

Politecnico di Bari

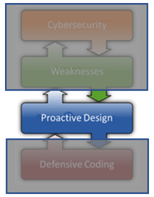
DIPARTIMENTO DI
INGEGNERIA ELETTRICA
E DELL'INFORMAZIONE

Secure Programming Lab: Course Program



A. Intro Secure Programming: «Who-What-Why-When-Where-How»
B. Building Security in: Buffer Overflow, UAF, Command Inection
C. SwA: Weaknesses, Vulnerabilities, Attacks
D. SwA (Software Assurance): Vulnerabilities and Weaknesses (CVE, OWASP, CWE)
E. Security & Protection: Objectives (CIA), Risks (Likelihood, Impact), Rating Methodologies
F. Security & Protection: Security Indicators, BIA, Protection Techniques (AAA, Listing, Duplication etc.)
G. Architecture and Processes: App Infrastructure, Three-Tiers, Cloud, Containers, Orchestration
H. Architecture and Processes 2: Ciclo di Vita del SW (SDLC), DevSecOps (OWASP DSOMM, NIST SSDF)
I. Free Security Tools: OWASP (ZAP, ESAPI, etc), NIST (SAMATE, SARD, SCSA, etc), SonarCube, Jenkins
J. Dynamic Security Test: VA, PT, DAST (cfr. VulnScanTools), WebApp Sec Scan Framework (Arachni, SCNR) :
K. Operating Environment: Kali Linux on WSL
L. Python: Powerful Language for easy creation of hacking tools
M. Exercises: SecureFlag

Development Framework



H.0 SDLC: SW Development Lifecycle

H.1 DevOps: Introduction, CI/CD

H.2 DevSecOps: Manifesto, Phases, Maturity, Tools

H.3 Framework: OWASP DSOMM, NIST SSDF

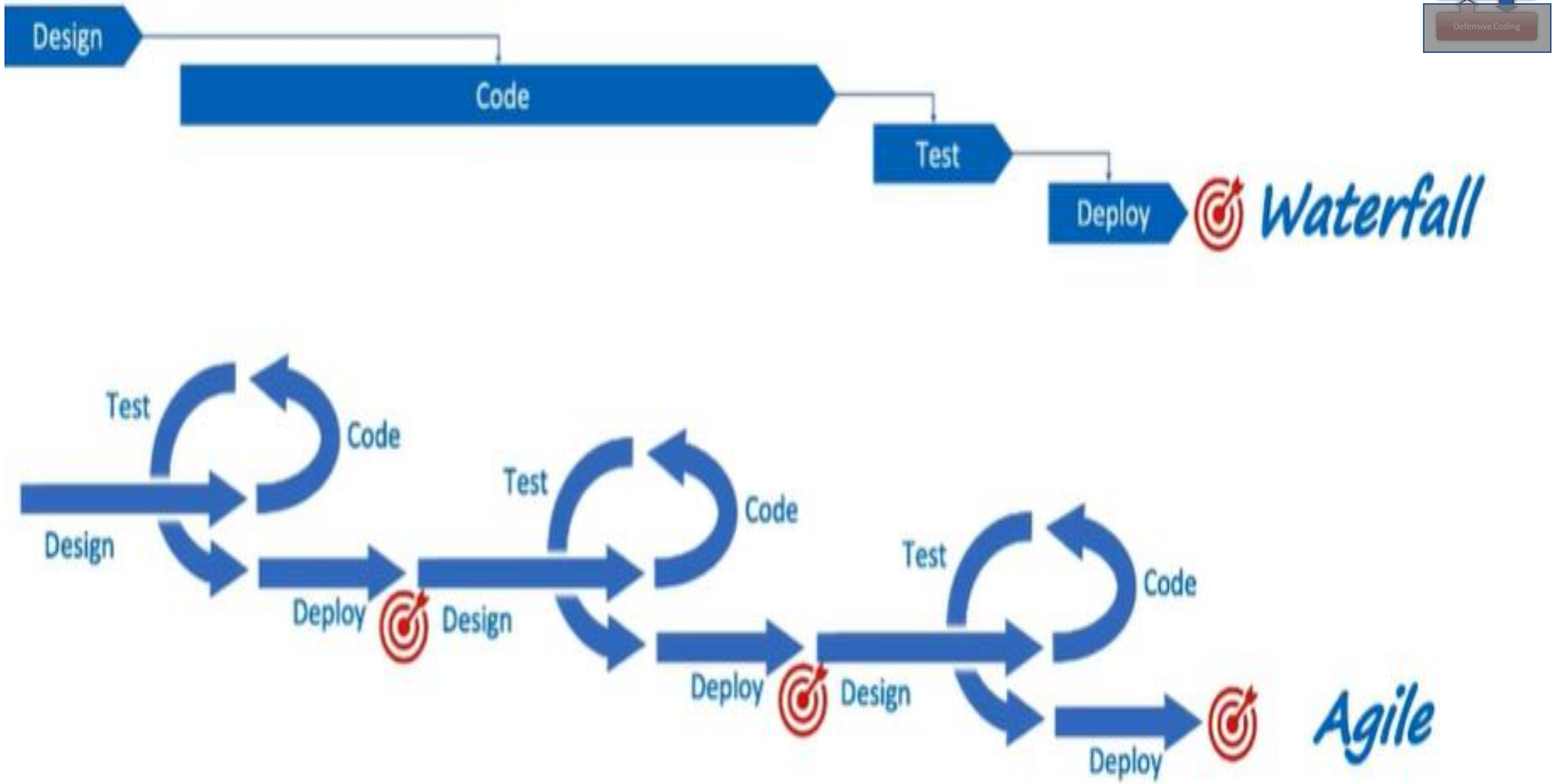
H.0 SDLC: Development Lifecycle

SW Development Methodologies

Developing…

Waterfall is a linear system of working that requires the team to complete each project phase before moving on to the next one

Agile encourages the team to work simultaneously on different phases of the project.



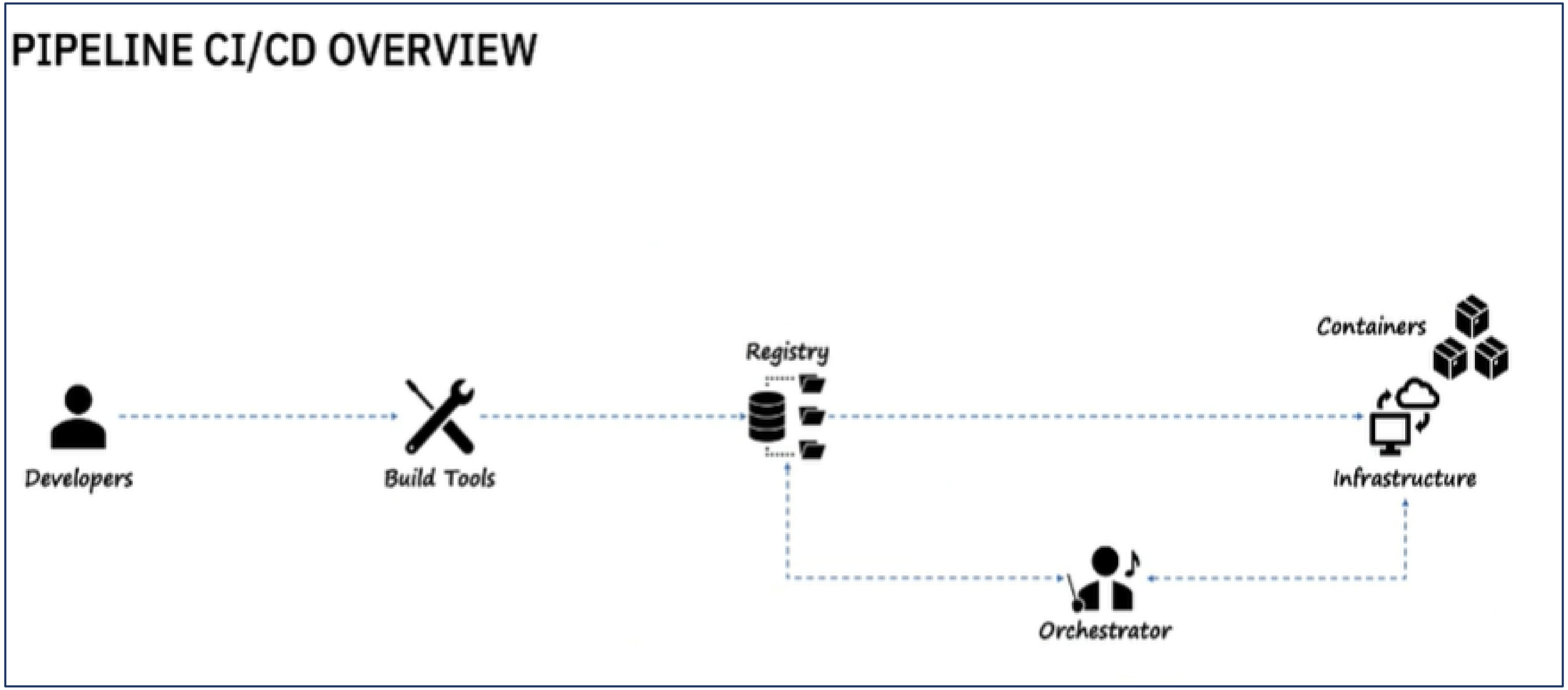
Methodology	Approach	Flexibility	Requires
Waterfall	Hands-off; goals and outcome established from the beginning	Low	Completing deliverables to progress to the next phase
Agile	Frequent stakeholder interaction	High	Team initiative and short-term deadlines

H.0 SDLC: Development Lifecycle

CI/CD Pipelines Overview (see GitLab About)

Continuous Integrazione / Continous Delivering

- falls under DevOps (the joining of development and operations teams)
- combines the practices of continuous integration and continuous delivery.
- automates much or all of the manual human intervention traditionally needed to get new code from a commit into production, encompassing the phases:
  - build,
  - test (including integration tests, unit tests, and regression tests),
  - deploy phases
  - infrastructure provisioning.

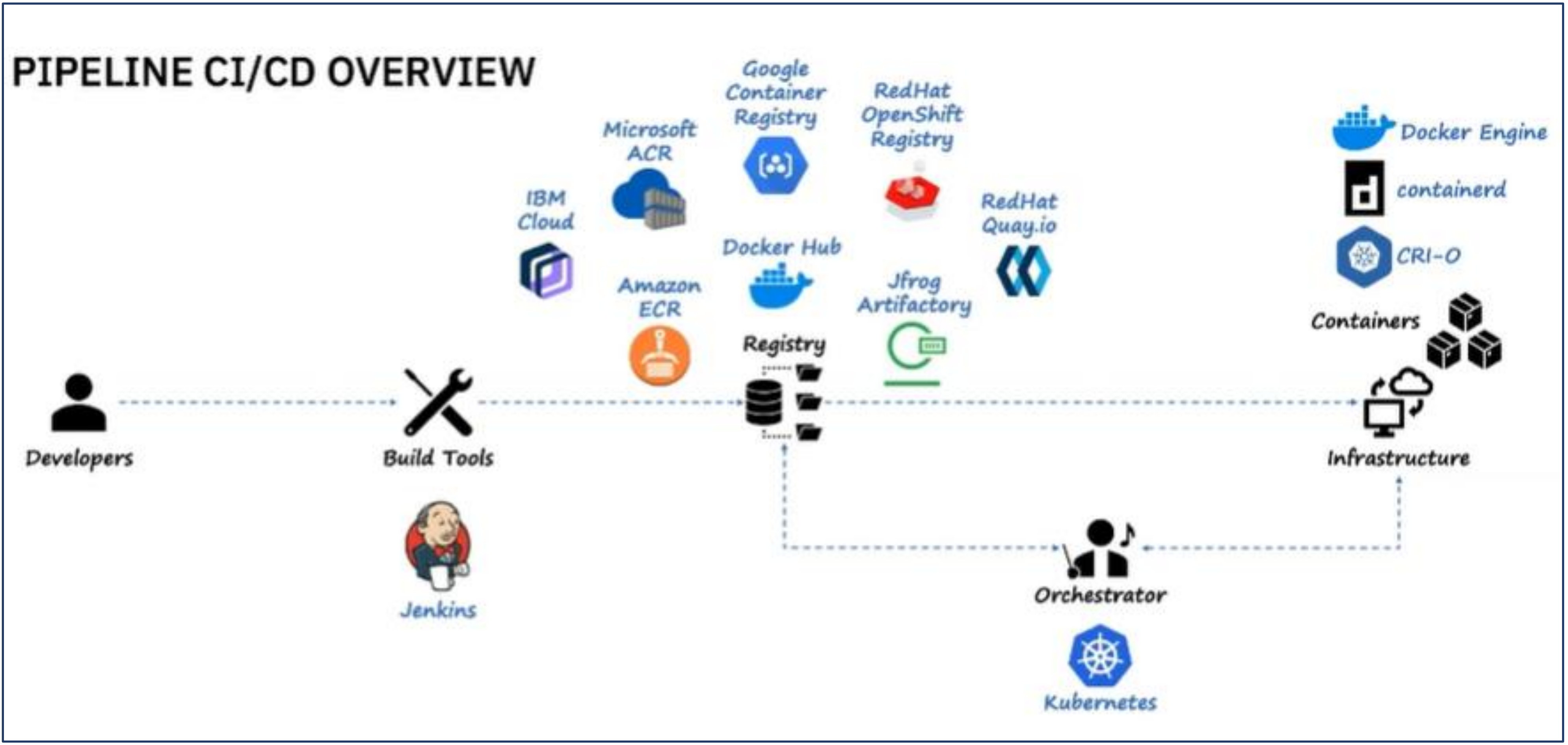


With a CI/CD pipeline, development teams can make changes to code that are then automatically tested and pushed out for delivery and deployment.

H.0 SDLC: Development Lifecycle

CI/CD Pipelines Overview (see GitLab About)

CI/CD fundmentals

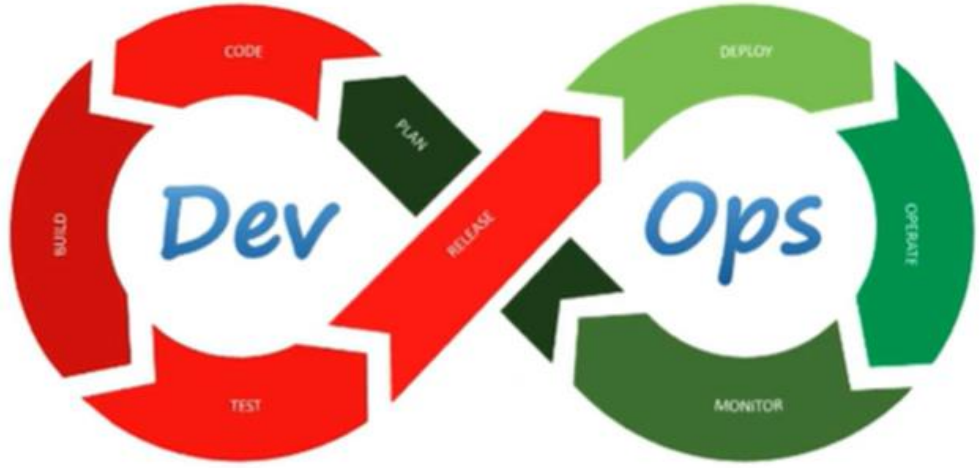


- **SCM** (Source Code Management): single source repository
- **Branchless**: frequent check-ins to main branch. Avoid sub-branches
- **Automated builds**: script for building from a single command
- **Self-testing builds**: test failure implies failed build
- **Frequent iterations**: better than major changes
- **Stable testing environments**: clone of the environment
- **Maximum visibility**: latest executables accessible by every developer
- **Predictable deployments anytime**: routine and low-risk deployment performable anytime

H.0 SDLC: Development Lifecycle

DevOps

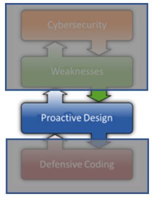
Developing…



1. **Code:** The first step in this DevOps lifecycle is coding. In this step, the developers write the code on any platform to develop the product for a customer.
2. **Build:** The second step is to build where the basic version of the product is built using a suitable **programming language**.
3. **Test:** The third step test where the built products are tested using the automation testing tools such as Selenium web driver, selenium RC, Bugzilla, etc.
4. **Release:** This step involves planning, scheduling, and controlling the built process in a different environment.
5. **Deploy:** All the deployment products and files are executed on the server.
6. **Operate:** After the deployment of the product or application, it is delivered to the customer for use where he uses that product or application for daily life purposes.
7. **Monitor:** In this step, the delivered products or application to a user has been monitored to note any uptime and downtime failures or errors.
8. **Plan:** After monitoring, it gathers all the information and feedback from the customer and plans the changes needed to improve it.

H.1 DevOps: Introduction

Automate Security into CI/CD Pipelines with Jenkins- Introduction to DevSecOps

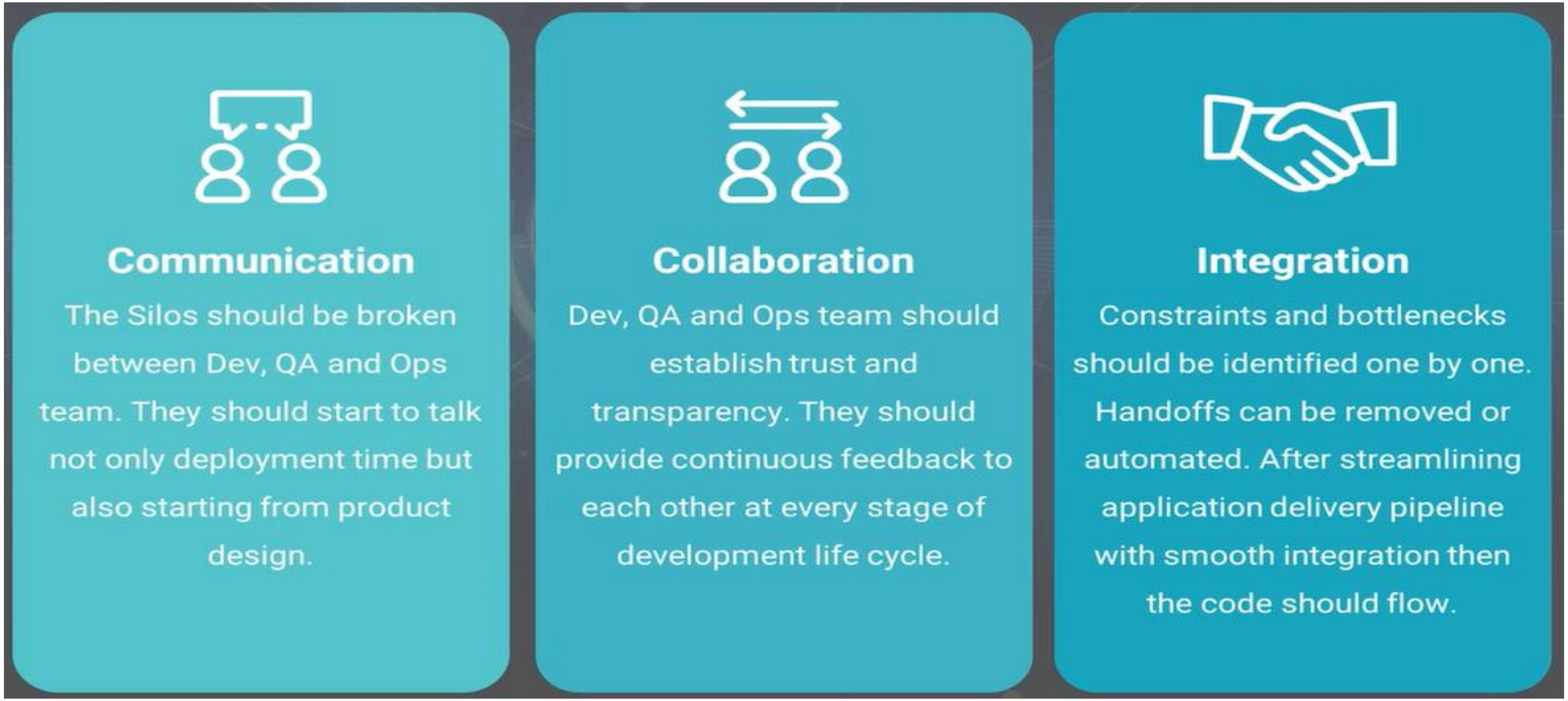


1. **DevOps Concepts**: What is? What problems it is trying to solve
2. **Continous Integration (CI)**: What is, Phases
3. **Continous Delivery (CD)**: What is, Phases
4. **Continous Deployment (CI/CD)**: What is, Where and How deploy the solution
5. **Security Challenge**: secure the CI/CD pipeline
6. **Shift Left**: secure products/solutions in early phases

https://codered.eccouncil.org/courseVideo/04bcb9a8-a3c3-45f3-9860-913cbdb2c40b
https://codered.eccouncil.org/courseVideo/04bcb9a8-a3c3-45f3-9860-913cbdb2c40b?lessonId=54280f83-a379-4071-ade3-a0ea8c5dcbb1

H.1.1. DevOps Concepts: definition

DevOps: practice for increasing Communication, Collaboration and Integration between Development (Dev) <-> Quality Assurance (QA) <-> Operations (Ops) teams.



Communication

The Silos should be broken between Dev, QA and Ops team. They should start to talk not only deployment time but also starting from product design.

Collaboration

Dev, QA and Ops team should establish trust and transparency. They should provide continuous feedback to each other at every stage of development life cycle.

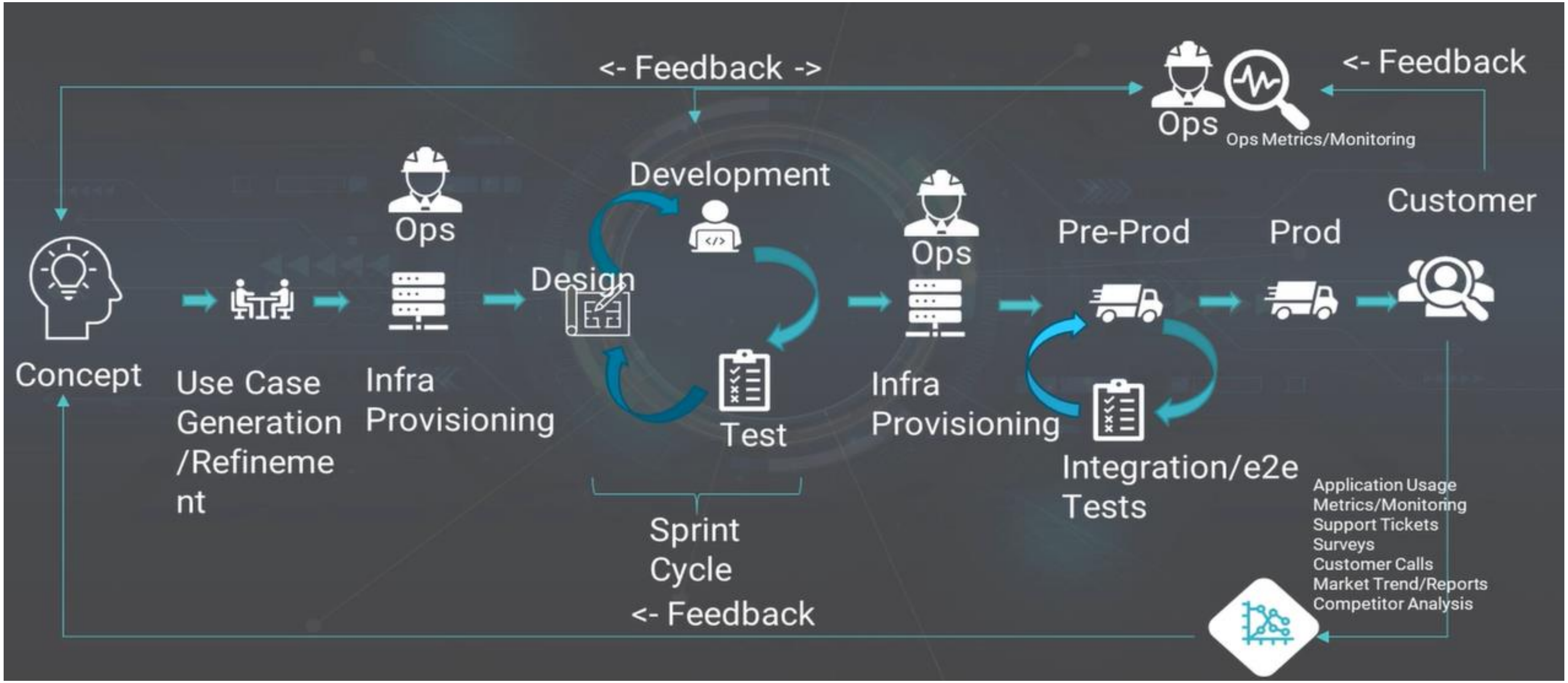
Integration

Constraints and bottlenecks should be identified one by one. Handoffs can be removed or automated. After streamlining application delivery pipeline with smooth integration then the code should flow.

H.1.1. DevOps Concepts: main goal (problem to solve)

DevOps Goal (Why): reduce Lead Time

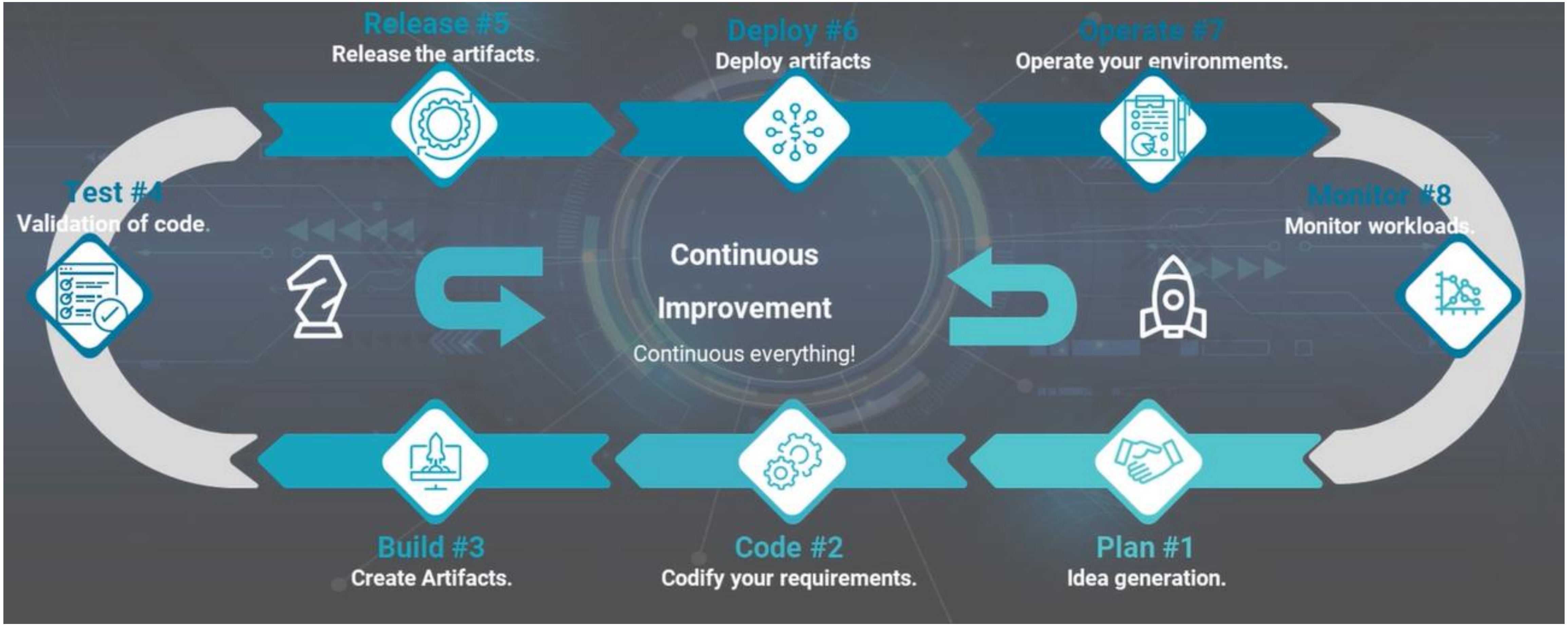
Lead Time: time intercurring from the initial **Idea** to the product/solution **Release** into the market



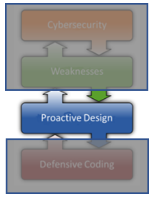
H.1.1. DevOps Concepts: Cycle (Application Life Cycle Management) 1

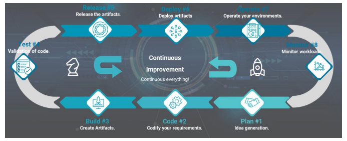
DevOps Mean: reduce step duration (mainly by automation)

Continous Improvement: enhancing the steps, by introducing more and more automation



F1.1. DevOps Concepts: Cycle (App Life Cycle Management) 2/2

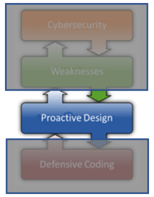




1. **Plan**: Conceiving the business idea. Scheduling, based on the original business idea (e.g. integration needs, number of stakeholders, implementation difficulty, etc)
2. **Code**: put the functional requirements in code (depending on programming language).
3. **Build**: depending on the programming language
4. **Test**: validation of code (to be compliant to functional and non-functional requirements)
5. **Release**: delivery the code to the release manager (into repository)
6. **Deploy**: into staging (non-production) and production environments.
7. **Operation**: Mantain and Manage the environments
8. **Monitor**: Check and Measure Application Usage

Continous Improvement: enhancing the steps, by introducing more and more automation

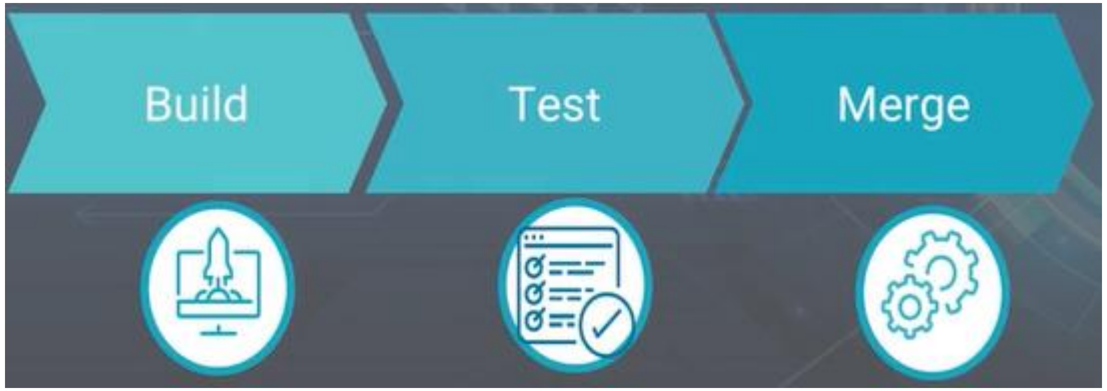
H.1.1. DevOps Concepts: Continous Flow (Continous Improvement)



#1 Continous Integration: Check out the code.

Run unit and integrations tests.

When ok, merge it on the main branch.



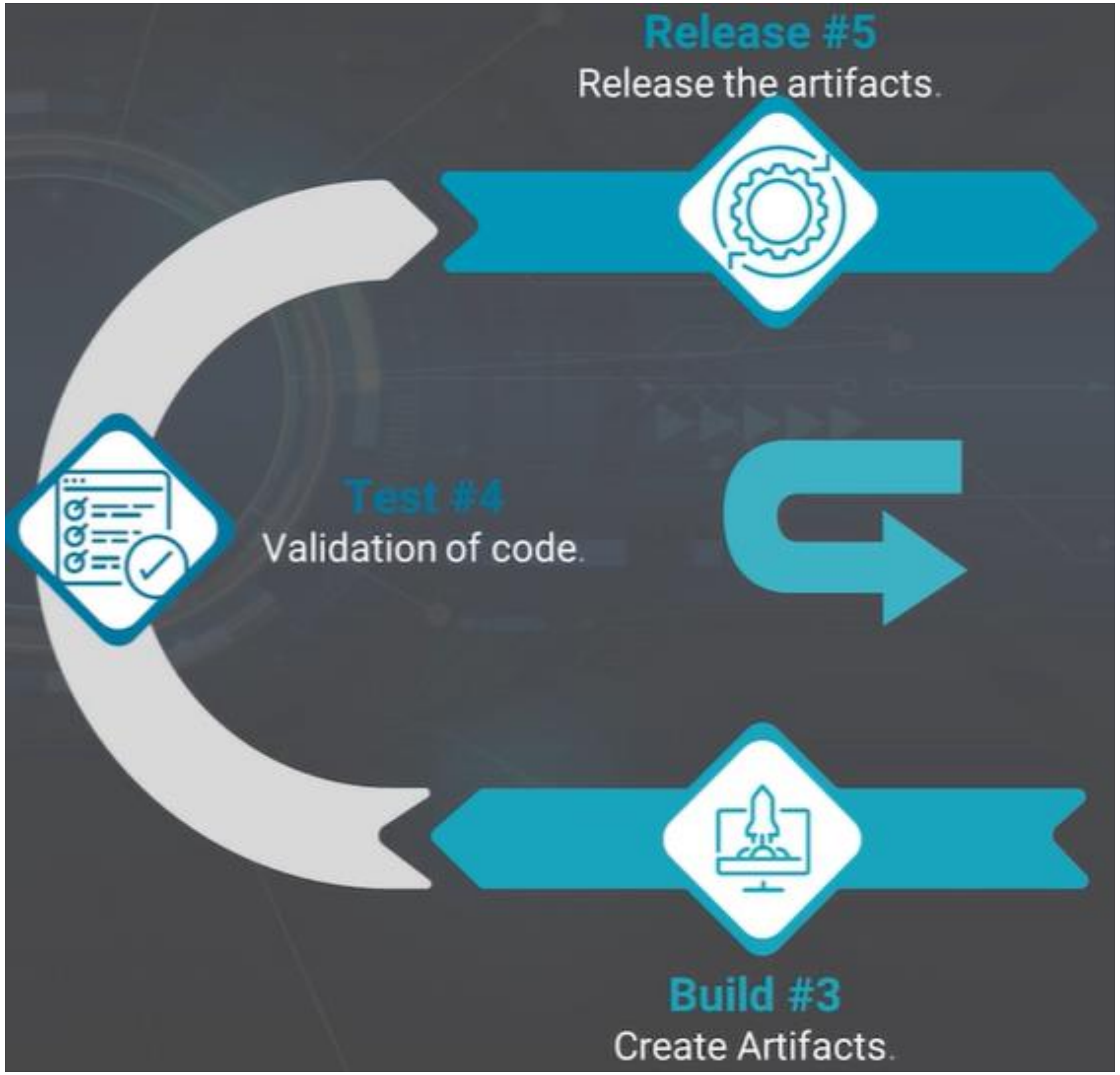
Release to
Repo and
Non-Prod Env

#2 Continous Delivery: deliver artifacts to the repository, automatically

#3 Continous Deployment: when new artifacts arrived automatically, deploy to production.

Deploy to
Prod

H.1.2. Continous Integration (CI) 1/3



Code Merging: developed by by multiple developers, several times in a day

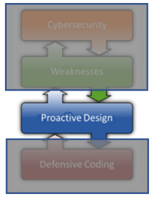
Code Check: code quality tools, syntax/lynter checkers, code review tools

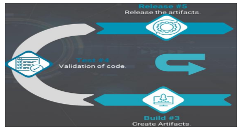
Automatic Testing: Test-Driven Development (TDD)

Early Detection: the team can identify problems in early stage

Deployability: deployable artifact at the end pf the stage

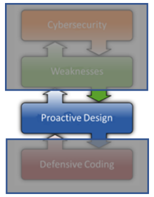
H.1.2. Continous Integration (CI) 2/3
Requirements

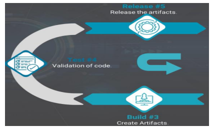




1. **Version Control System** (es. Git): developed by by multiple developers, several times in a day
    - Distributed Development Capabilities
    - User Auth, AuthZ
    - Commit Audit: History Mechanism for all changes
2. **Build Tool** (CI Tool/Server)
    - Dependency Resolver (es. Ant, Maven) for package/lib management
    - Reproducible Building Blocks
    - Orchestration/Pipeline Generation (es. Jenkin, GoCD, CircleCI, TeamCity)
3. **Artifact Repository Manager** (es. Sonatype Nexus, Jfrog)
    - Caching packages/libs
    - User Auth, AuthZ
    - Tagging, versioning, storage

H.1.2. Continous Integration (CI) 3/3
Best Practices





1. **Single Code Repository**: while frequent Code Check-In
2. **Indepedent/Parallel Work**: Multi-developer with own features/local branches
3. **Automatic Builds and Tests**
4. **Commit → Build**. Each commit should trigger a build (fixed immediately if broken)
5. **Short Building Time** (say, 10 minutes)
6. **Shared Build Result** (Succes or Failure): to the team
7. **Automated Test**: environment creation, test execution
8. **Dashboard**: providing report on what is heppening

H.1.3. Continous Delivery (CD) 1/3

Automatize the Delivery Process (obtaining Artifact at the End)

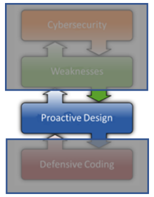
- Practice of automating the entire SW release process and getting artifacts on repository and non-prod environments
- Artifact: different types, depending on target deploymment environment (also based on programming language)
- Es. Artifactory: Sonatype Nexus, Jfrog

Continuous Delivery # 02

Deliver artifacts to the repository automatically.

H.1.3. Continous Delivery (CD) 2/3

Automatize the Delivery Process (obtaining Artifact at the End)

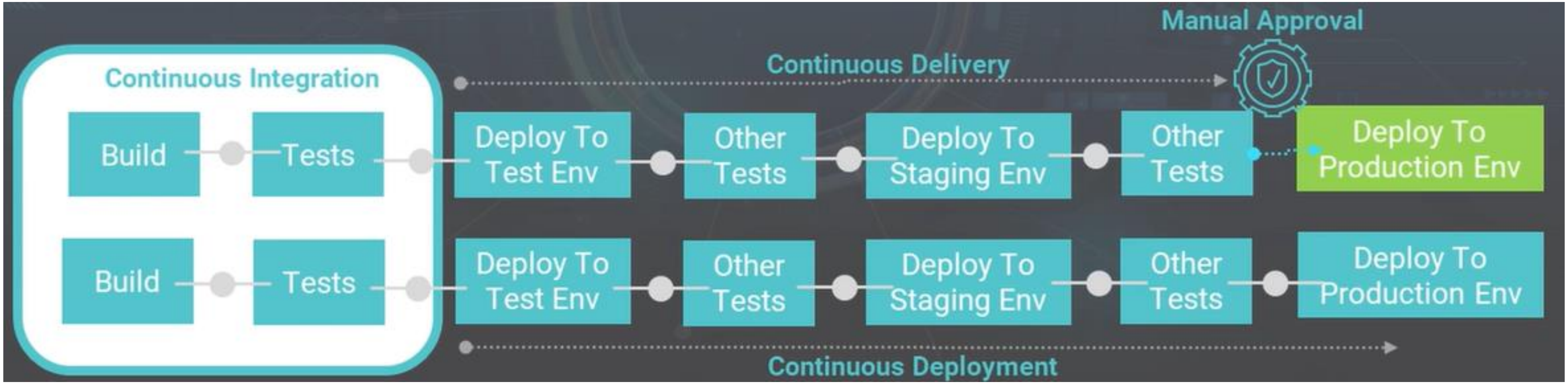


- **Artifactories**: store different kinds of artifact packages (e.g. npm, docker image, mvn dependencies, rpm, etc)
- **Integrated Code**: in deployable state of the production
- **Manual Approval**: the artifact can be deployed to production, since it passed all tests (Integration, User Acceptance, Law, Performance). Business Decision ➔ Human intevention (Release Mgr, Change Mgr, etc)

H.1.3. Continous Delivery (CD) 3/3

Differences between Continous Integration, Continous Delivery, Continous Deployment

- **Continous Integration**: the SW is continosly tested
- **Continous Delivery**: the Business is involved in SW deployment
- **Continous Deployment**: the SW is continosly deployed, without Business Intervention



H.1.4. Continous Deployment 1/3

Automatize the Deployment Process (getting code and directly deploying to Production)

Continous Deployment

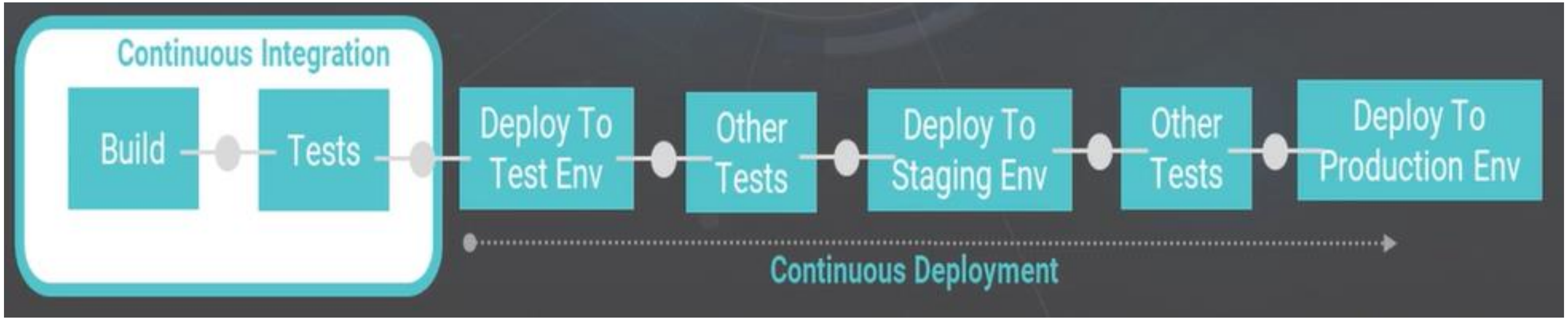
1. The entire chain of moving code from source repository to «Production» environment is automated
2. No manual approval (no human intervention by Business)
3. Artifacts (images, jar, rpm, etc) are tagged when deployed for audit and roll-back purposes

Continuous Deployment # 03

When new artifacts arrived automatically deploy to production.

H.1.4. Continous Deployment 2/3

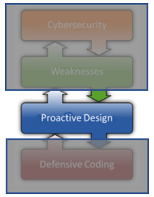
Deployment Process from environment to environment



Not always the tests are easy to get automatic

H.1.4. Continous Deployment 3/3

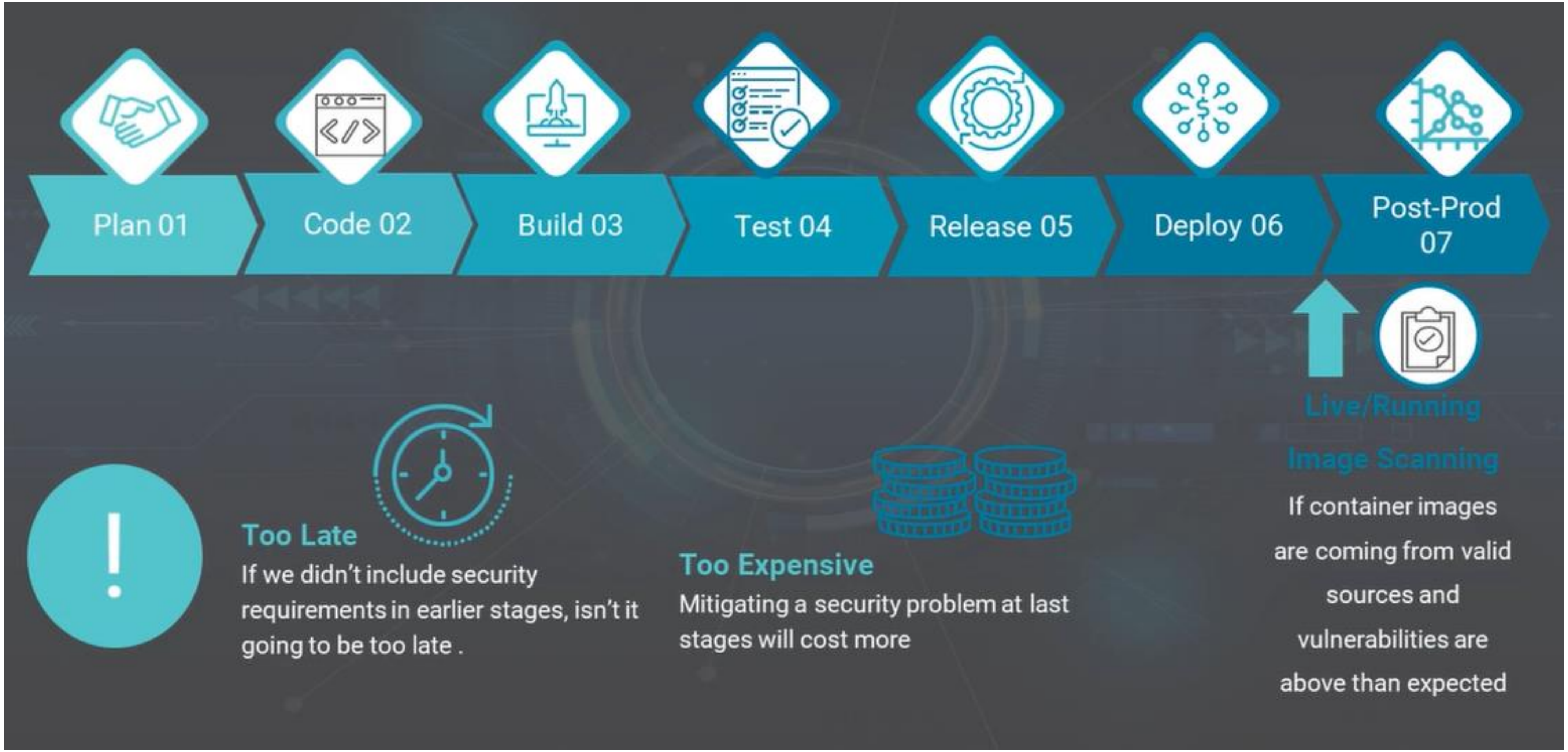
Advantages of Deployment Process



1. Increasing Development productivity and Confidence
2. Eliminating too big long live local branches
3. Reducing (minimalizing) A/B testing, getting customer feedbacks
4. Shortening productization of ideas, testing against real customer behaviours
5. Making easyer to detect and fix problems since smaller packages
6. More satisfactory experiences to customer in product/service, since continous improvements

H.1.5. Security Challenges in CI/CD 1/2

Drawbacks in Traditional Pipelines



Too Late
If we didn't include security requirements in earlier stages, isn't it going to be too late .

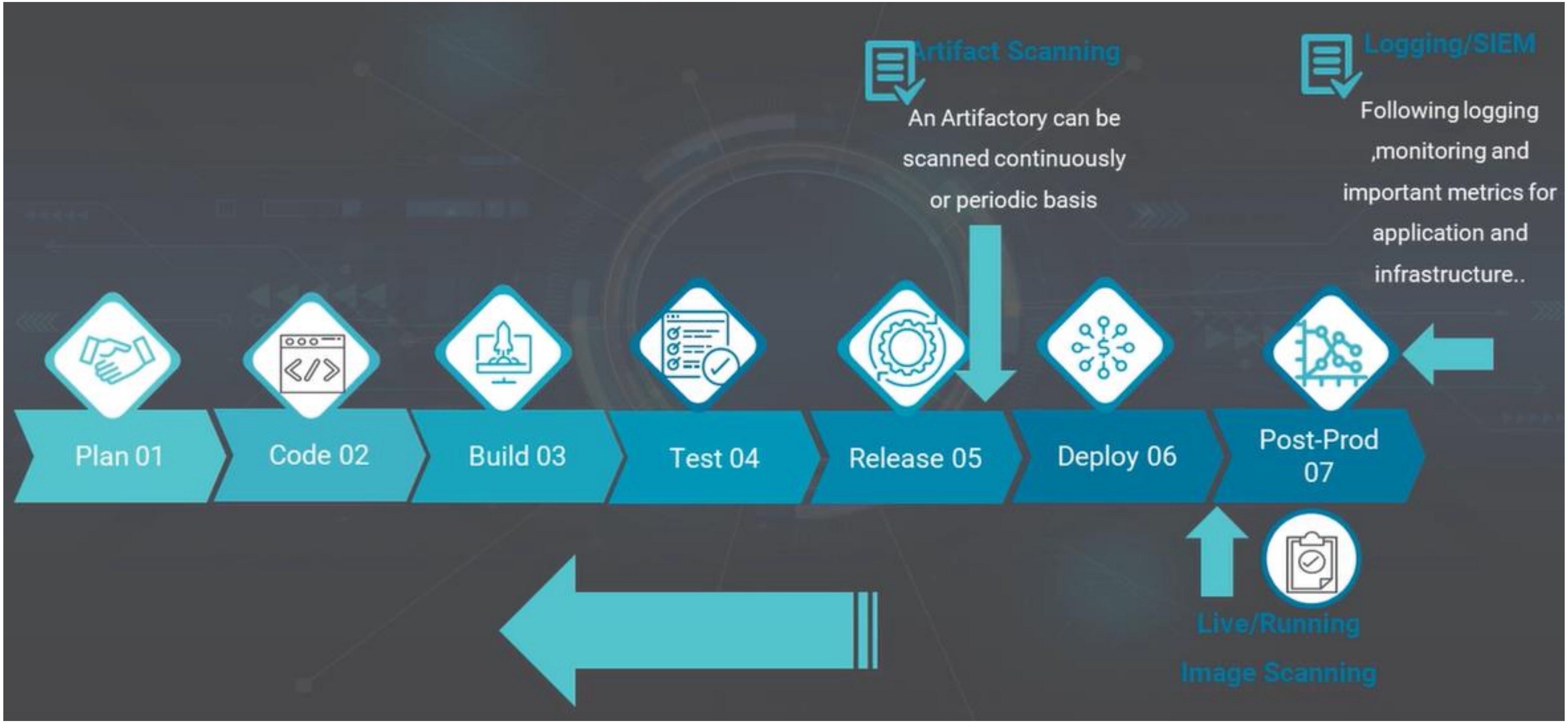
Too Expensive
Mitigating a security problem at last stages will cost more

Live/Running Image Scanning
If container images are coming from valid sources and vulnerabilities are above than expected

1. A

H.1.5. Security Challenges in CI/CD 2/2

Minimal Security Insertions (during operations)

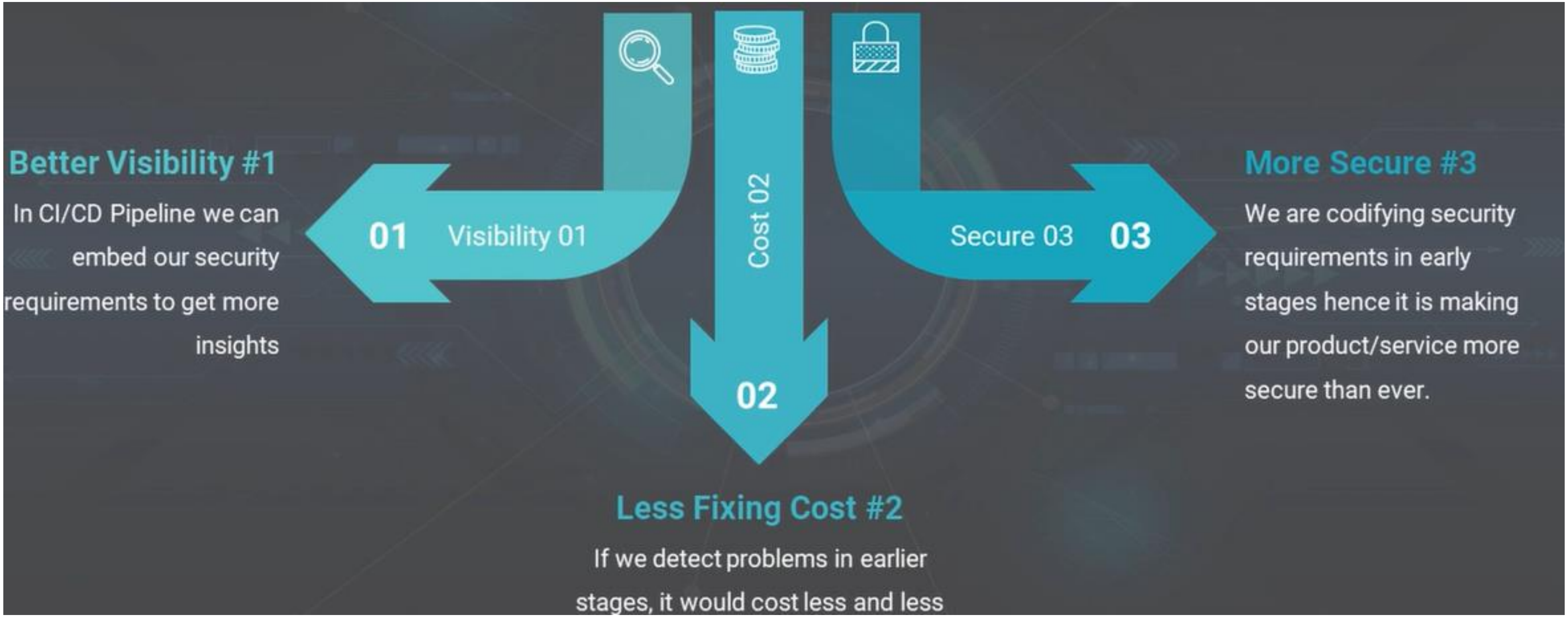


An Artifactory can be scanned continuously or periodic basis

Following logging ,monitoring and important metrics for application and infrastructure..

H.1.6. Shift Left Paradigm 1/4

Security is not a dedicated team’s responsibility



Better Visibility #1

In CI/CD Pipeline we can embed our security requirements to get more insights

More Secure #3

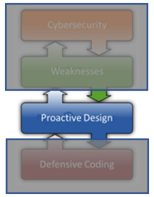
We are codifying security requirements in early stages hence it is making our product/service more secure than ever.

Less Fixing Cost #2

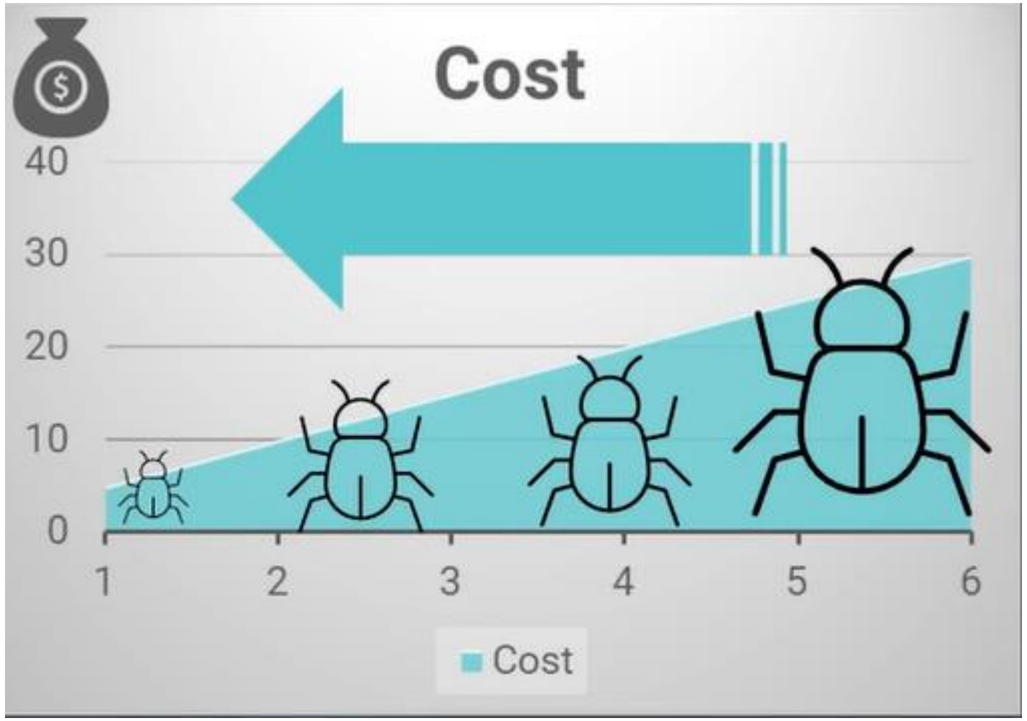
If we detect problems in earlier stages, it would cost less and less

H.1.6. Shift Left Paradigm 2/4

DevOps Cycle Security Review

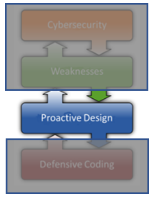


1. Codifying Security Requirements
2. Finding a place in CI/CD for embedding security
3. Security is a product/practice: more secure with proper actions in the SDLC phases (having different stakeholders)
4. Establishing scalable and repeatable security gates
5. Saving money on fixing bugs and problems when these are not large
6. Automating security actions (prevention, detection, mitigation)



H.1.6. Shift Left Paradigm 3/3

DevOps Cycle Security View

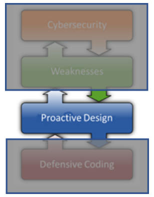


Not only Penetration Test at the End of Release: Security Requirements would be spreaded and injected into different phases:

1. Code Analysis
2. Compliance Checks
3. Vulnerability Detections
4. Secure Control Loops over all SDLC
5. Continous Monitoring (SIEM: Security Information and Event Management)
6. Continous Pattern Evaluation (IDS/IPS: Intrusion Prevention/Detection Systems)

H.1.7 Security and DevOps: Summary

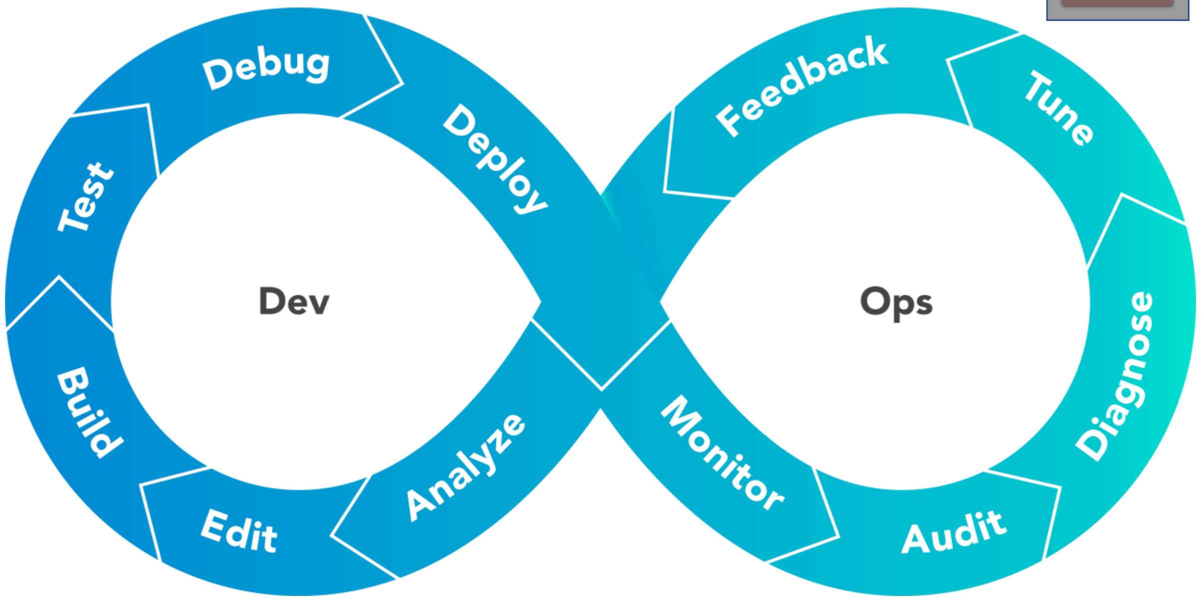
DevOps Cycle Security View

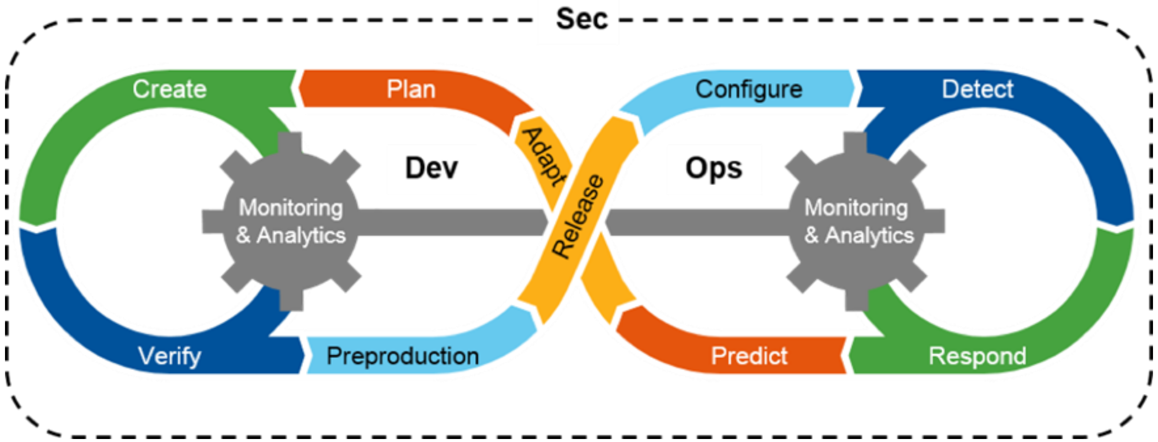


1. **DevOps is leaning the IT processes**: bringing smoother flow in IT processes

2. **CI/CD is way to go**: increasing quality of delivery and efficiency from idealization to a releasing product

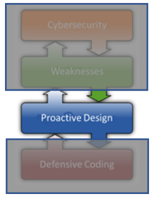
3. **Enbedding Security**: the requirements introduced and codified into different places of CI/CD Pipelines, it is better than acting in last minute deliveries





H.2. DevSecOps: Introduction

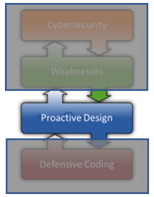
Automate Security into CI/CD Pipelines with Jenkins- Introduction to DevSecOps

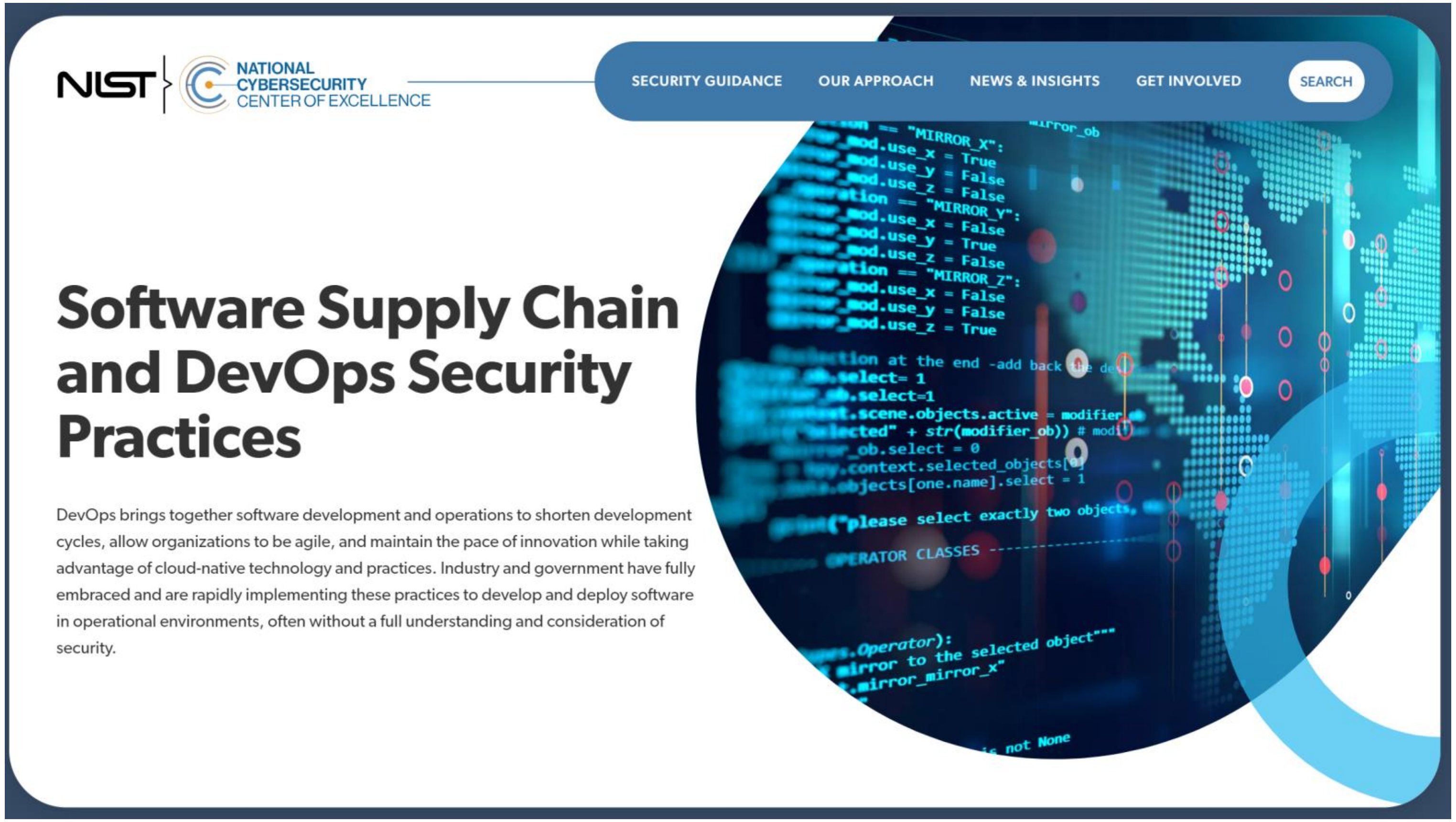


1. **DevSecOps Concepts**: what is DevSecOps? What are we trying to solve?
2. **DevSecOps Manifesto**: what can we get from manifesto?
3. **DevSecOps approach**: Which Problems can be solved with?
4. **Security in the CI/CD**: Placing/Positioning in Pipeline Phases
5. **Maturity Model**: adopting the approach and find a way to adopt
6. **Implementation**: tool selection and strategy

H.2.1 DevSecOps: what is it?

DevSecOps NIST





Software Supply Chain and DevOps Security Practices

DevOps brings together software development and operations to shorten development cycles, allow organizations to be agile, and maintain the pace of innovation while taking advantage of cloud-native technology and practices. Industry and government have fully embraced and are rapidly implementing these practices to develop and deploy software in operational environments, often without a full understanding and consideration of security.

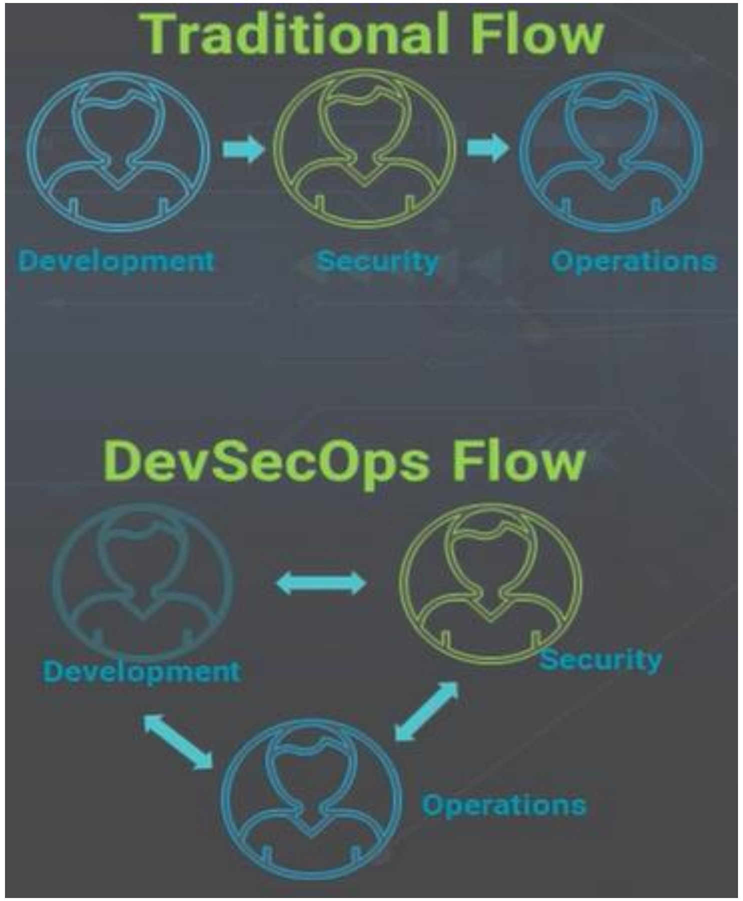
NCCoE DevSecOps project:

[Software Supply Chain and DevOps Security Practices | NCCoE (nist.gov)]

https://codered.eccouncil.org/courseVideo/04bcb9a8-a3c3-45f3-9860-913cbdb2c40b
https://codered.eccouncil.org/courseVideo/04bcb9a8-a3c3-45f3-9860-913cbdb2c40b?lessonId=65f4f3de-214b-47cc-be59-50452cea99cd

H.2.1a DevSecOps: what is it?

DevSecOps Concept: Approach



DevSecOps: set of practices and mindset
to apply security
in all stages and level
of application life cycle management
within the DevOps process.

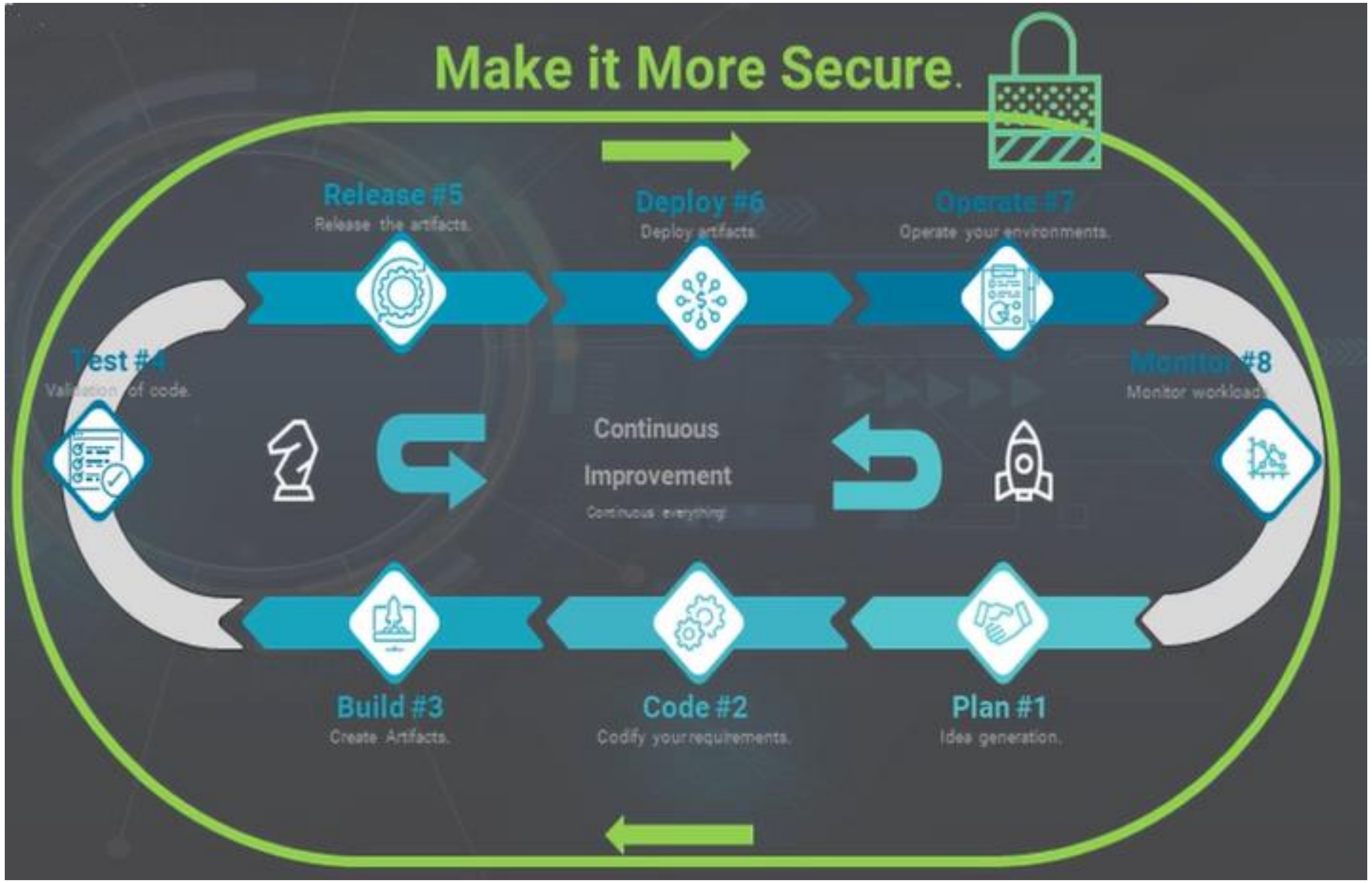
https://codered.eccouncil.org/courseVideo/04bcb9a8-a3c3-45f3-9860-913cbdb2c40b
https://codered.eccouncil.org/courseVideo/04bcb9a8-a3c3-45f3-9860-913cbdb2c40b?lessonId=65f4f3de-214b-47cc-be59-50452cea99cd

H.2.1b DevSecOps: what is it?

DevSecOps Concept: Activities

DevSecOps: other than #4 Test and #8 Monitor

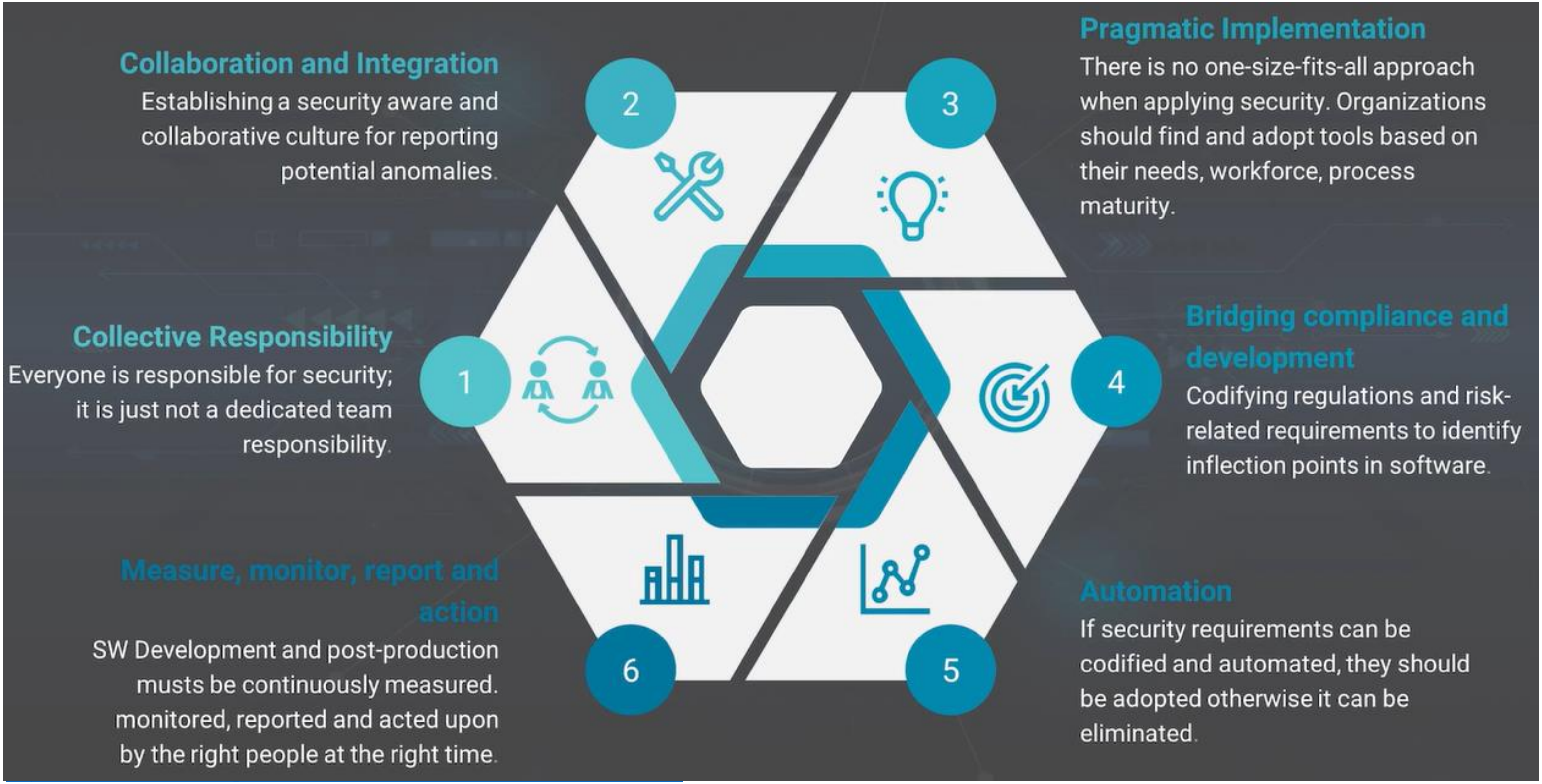
- #1 Plan: more secure AuthN and AuthZ
- #2 Code: addiction of security requirements (es. Logging)
- #3 Build: best libraries
- # 5 Release: protective integration
- #6 Deploy: secure environment
- #7 Operate: detection integration



https://codered.eccouncil.org/courseVideo/04bcb9a8-a3c3-45f3-9860-913cbdb2c40b
https://codered.eccouncil.org/courseVideo/04bcb9a8-a3c3-45f3-9860-913cbdb2c40b?lessonId=65f4f3de-214b-47cc-be59-50452cea99cd

H.2.1c DevSecOps: what is it?

DevSecOps 6 Pillarsby CSA (Cloud Security Alliance)



Collective Responsibility
Everyone is responsible for security; it is just not a dedicated team responsibility.

Collaboration and Integration
Establishing a security aware and collaborative culture for reporting potential anomalies.

Pragmatic Implementation
There is no one-size-fits-all approach when applying security. Organizations should find and adopt tools based on their needs, workforce, process maturity.

Bridging compliance and development
Codifying regulations and risk-related requirements to identify inflection points in software.

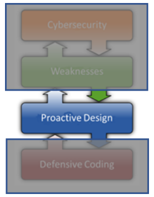
Automation
If security requirements can be codified and automated, they should be adopted otherwise it can be eliminated.

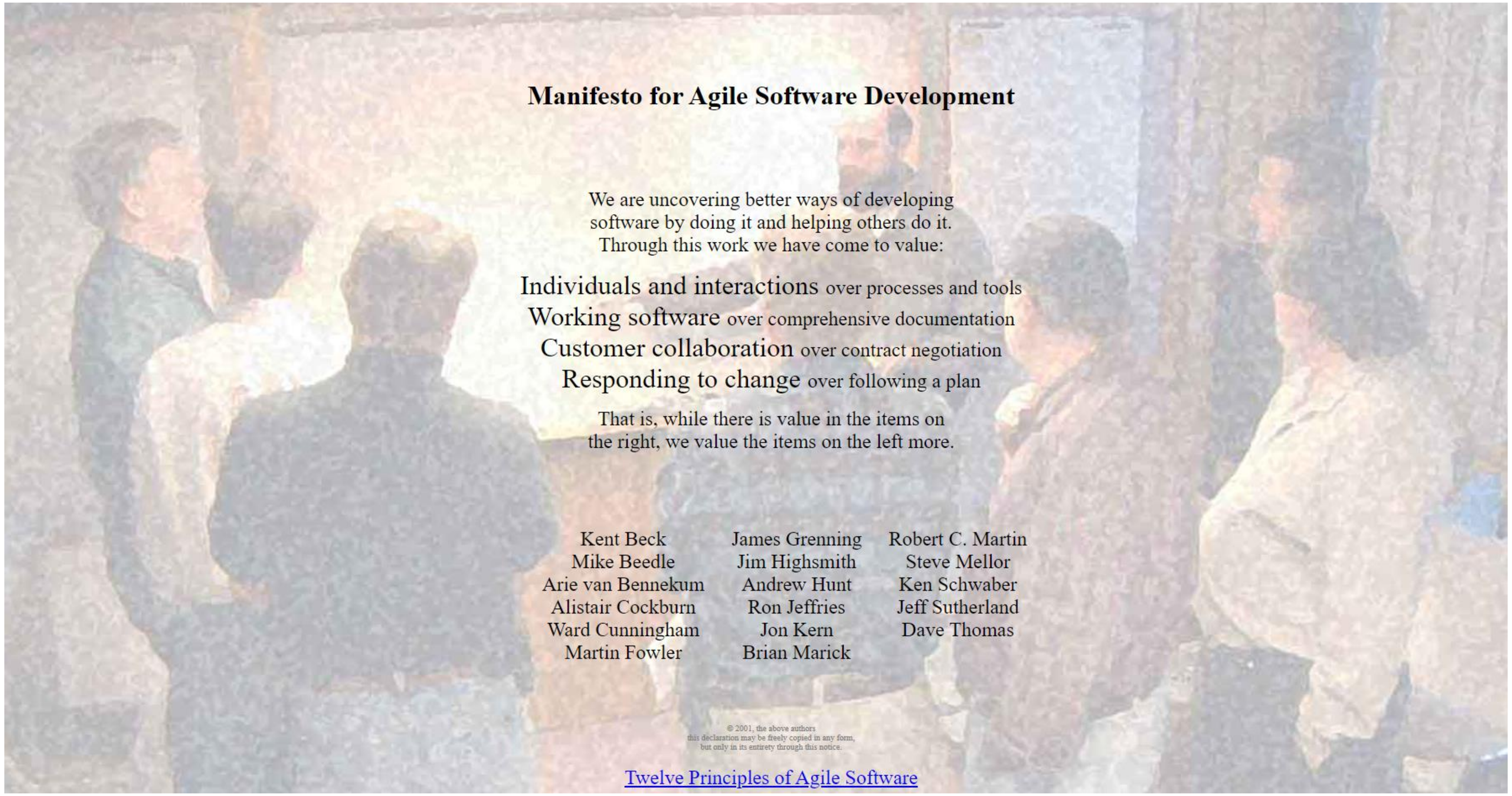
Measure, monitor, report and action
SW Development and post-production musts be continuously measured. monitored, reported and acted upon by the right people at the right time.

https://codered.eccouncil.org/courseVideo/04bcb9a8-a3c3-45f3-9860-913cbdb2c40b?lessonId=65f4f3de-214b-47cc-be59-50452cea99cd

H.2.2a DevSecOps: Manifesto

SW development: Agile manifesto





Manifesto for Agile Software Development

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.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

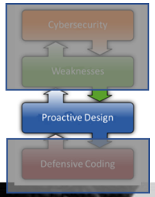
Twelve Principles of Agile Software

1. Satisfy Customer
2. Welcome Changing
3. Deliver SW frequently
4. Biz & Dev together
5. Motivate People
6. Conversate Face2Face
7. Progress 🡐 Working SW
8. Develop Sustainably
9. Design Tech Excellence
10. Maximize Work not Done
11. Self-Organize Teams
12. Improve Effectiveness Periodically

https://agilemanifesto.org/

H.2.2b DevSecOps: Manifesto

Security Manifesto like SW dev Agile manifesto

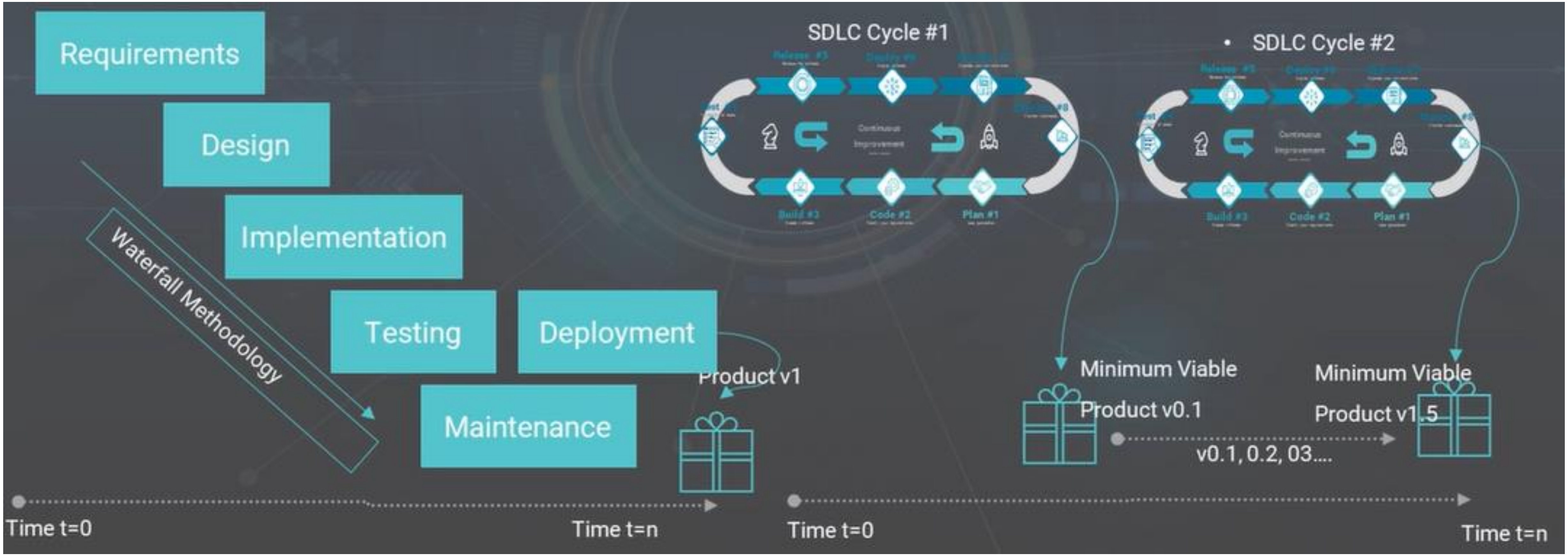


#	Name	over	Description
1	Leaning in	Always Saying "No"	**More Collaboration**: between Sec and Dev
2	Data & Security Science	Fear, Uncertainty and Doubt (FUD)	**Easily Misurable**: against each phase of CI/CD also about security
3	Open Contribution & Collaboration	Security-Only Requirements	**Embed Security**: Best Fit for the Company
4	Consumable Security Services with APIs	Mandated Security Controls & Paperwork	**Codified Security Services** (by API) and Measure whether the identified Security Services fit the application needs by evaluating the API consumption
5	Business Driven Security Scores	Rubber Stamp Security	**Evolving Security**: the measures reflect the continously changing business needs
6	Red & Blue Team Exploit Testing	Relying on Scans & Theoretical Vulnerabilities	**Real World Emulation**: using Red Team (outside attacker) and Blue Team (inside defender)- Continously.
7	24x7 Proactive Security Monitoring	Reacting after being Informed of an Incident	**Continous Defense**: security goes beyond one incident response, it should be put into the enterprise fabric, as daily operations.
8	Shared Threat Intelligence	Keeping Info to Ourselves	**Security Acumen Raising**: each member of the software development team is a contributing resource for a more secure computing environment.
9	Compliance Operations	Clipboards & Checklists	**Reasoning behind the Rules**: ongoing awareness, ongoing awareness of the rules, regulations and best practices around corporate IT security, by learn and adapt.

https://www.devsecops.org/

H.2.2c DevSecOps: Manifesto

Replacing Waterfall, Manual, Siloes



1. Agile Methodology repacing Waterfall
2. QA/Testing automatic and implemented in chunks
3. Security to be merged ad QA/Testing

https://www.devsecops.org/

H.2.3a DevSecOps: Approach

Security Sign-Off: PenTest/Audit of developed SW (usually in staging environment, yet)

Waterfall: SW PenTest/Audit

Time: enough (only 1 reiteration)

Skills: possible to find

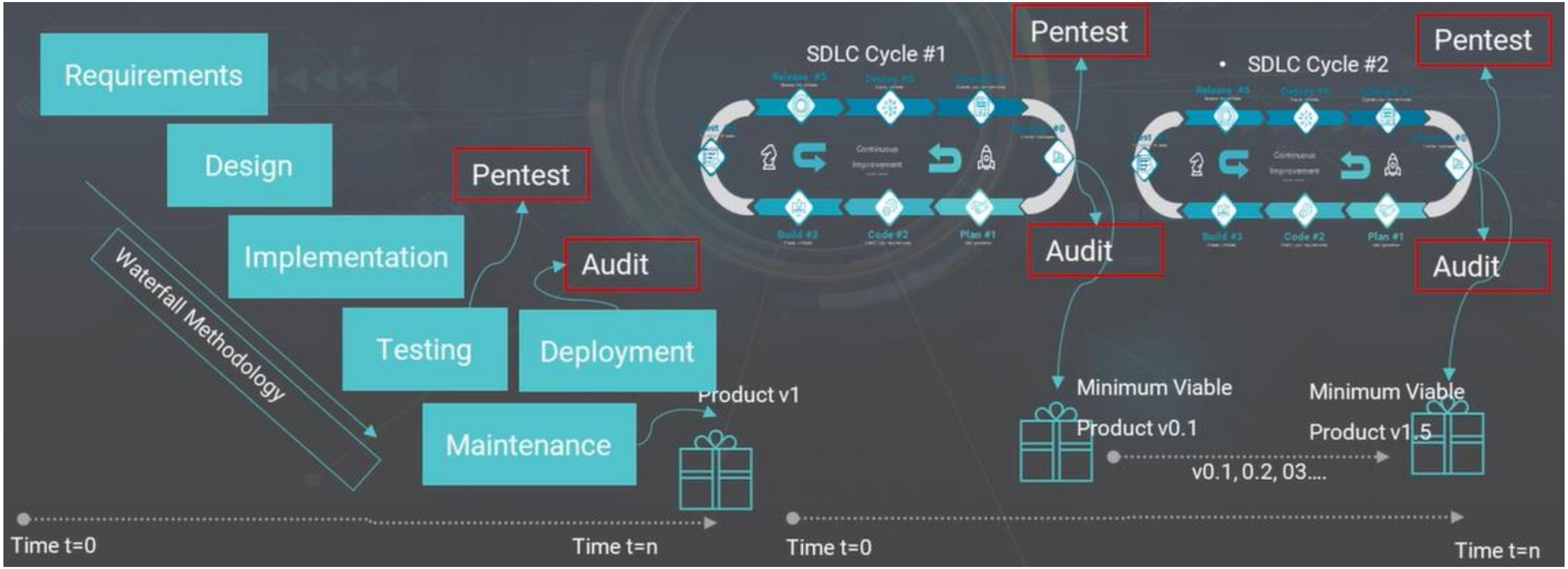
Reiteration: too expensive, in money and time

Agile Methodology: SW PenTest/Audit

Time: poor (many reiterations)

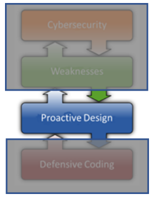
Skills: difficult to find (needs fast-testers)

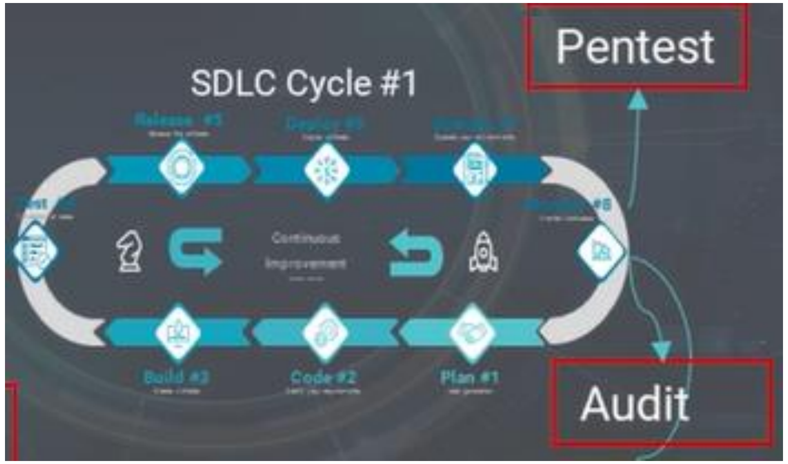
Reiteration: needs for speeding-up the process

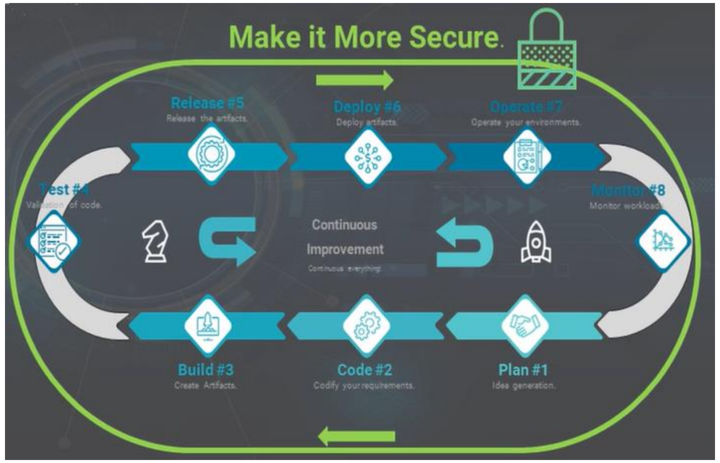


H.2.3b DevSecOps: Approach

Security Sign-Off: Problems & Resolutions





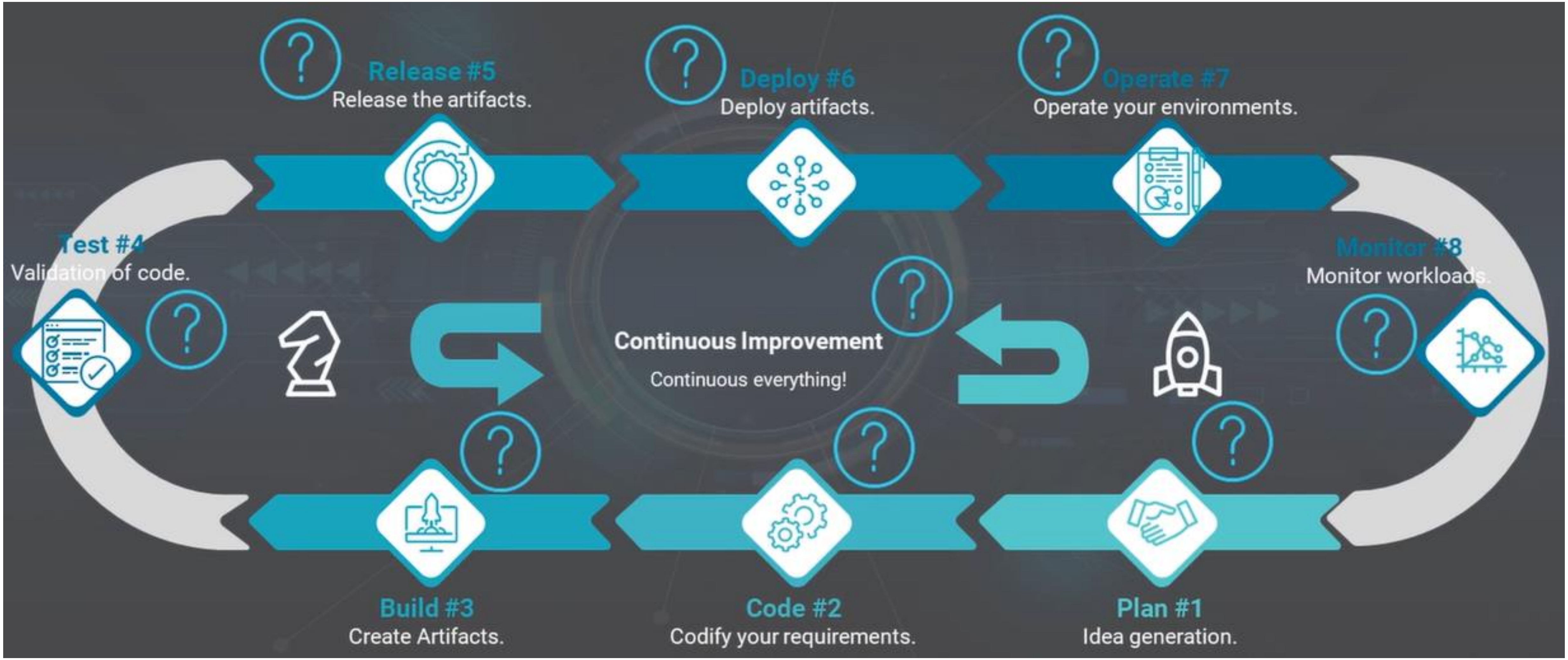


DevSecOps: other than #4 Test and #8 Monitor

- **Early Identification of Vulnerabilities/Issues** 🡐 #6 Deploy: secure environment
- **Early Fix** (Cost Reduction) 🡐 #2 Code: addiction of security requirements (es. Logging)
- **Improved Overall Security** 🡐 #1 Plan: more secure AuthN and AuthZ
- **Shared Responsibility** (everyone is responsible, cooperating) 🡐 #3 Build: best libraries
- **Secure by Design** (empowering developers with automation) 🡐 # 5 Release: protective integration
- **Not «Secured by PenTest»** 🡐 #7 Operate: detection integration

H.2.4a DevSecOps: CI/CD Pipeline

Application LifeCycle Management



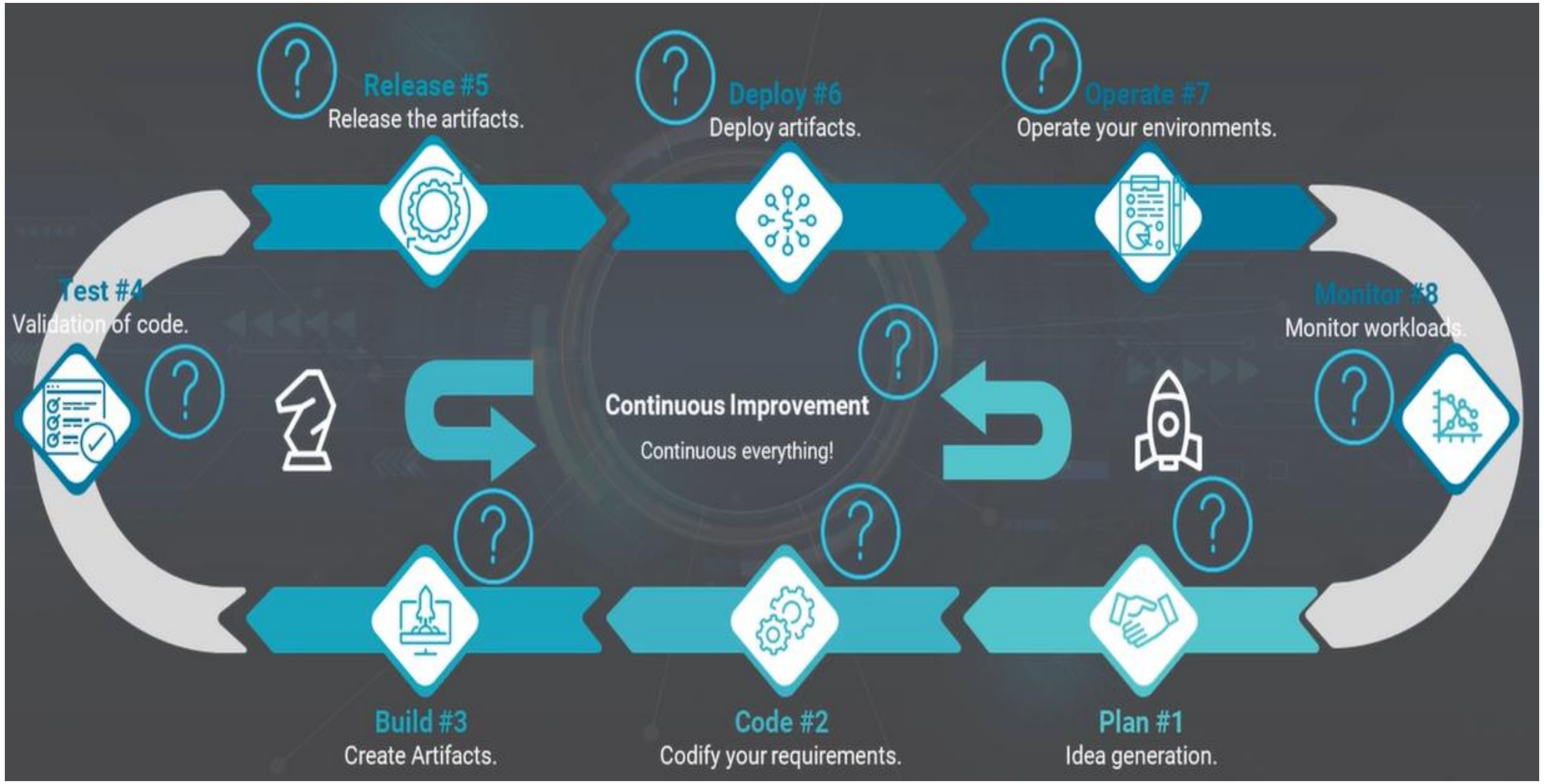
DevSecOps: in the CI/CD pipeline

H.2.4b DevSecOps: CI/CD Pipeline

Application LifeCycle Management

DevSecOps: in the CI/CD pipeline

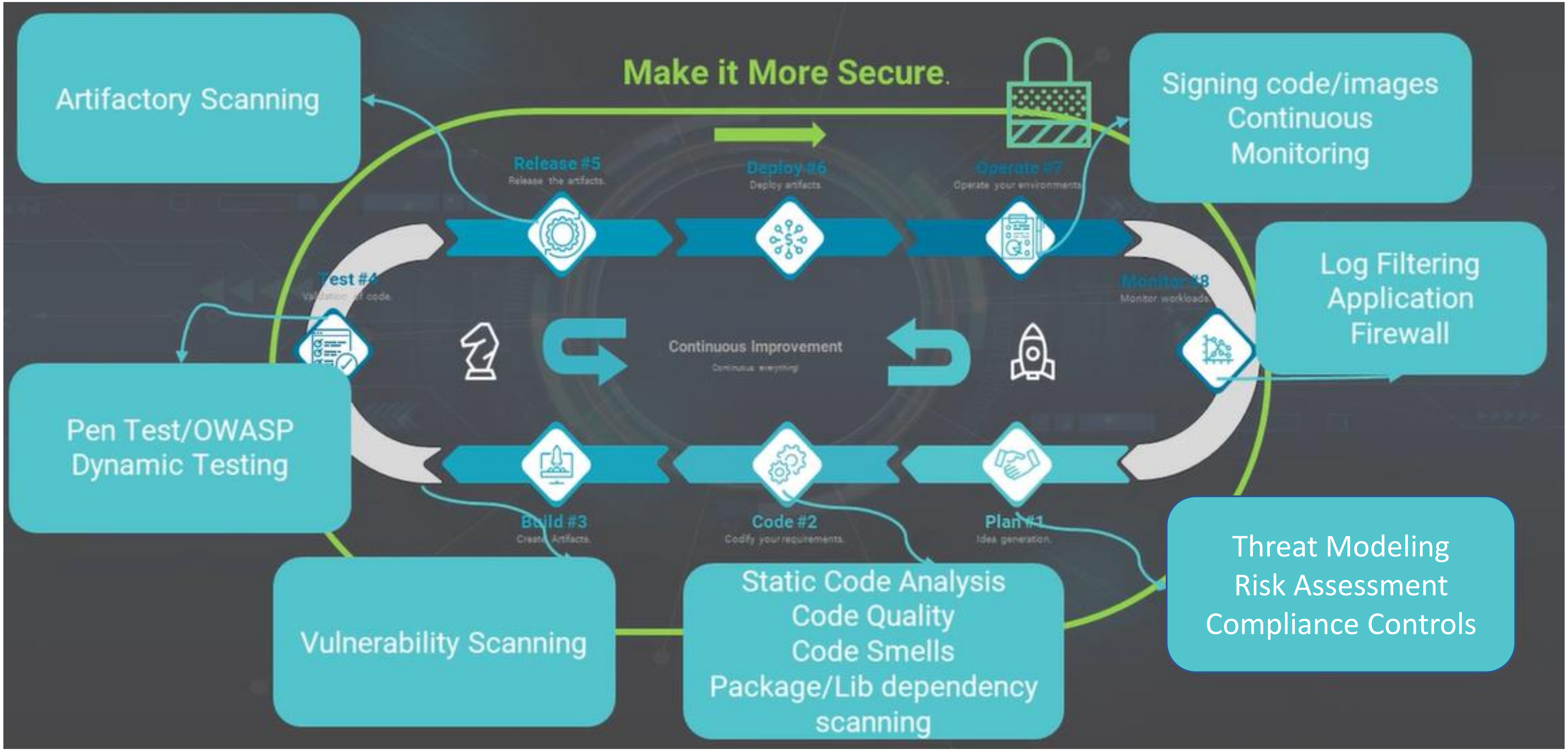
- #1 Plan: more secure AuthN and AuthZ
- #2 Code: addiction of security requirements (es. Logging)
- #3 Build: best libraries
- #4 Test: validation of code
- #5 Release: protective integration
- #6 Deploy: secure environment
- #7 Operate: detection integration
- #8 Monitor: monitor workloads



H.2.4c DevSecOps: CI/CD Pipeline

Merge DevOps and Security

Codify Security Requirements and streamline security controls all over the DevOps cycle



Artifactory Scanning

Signing code/images
Continuous
Monitoring

Log Filtering
Application
Firewall

Pen Test/OWASP
Dynamic Testing

Threat Modeling
Risk Assessment
Compliance Controls

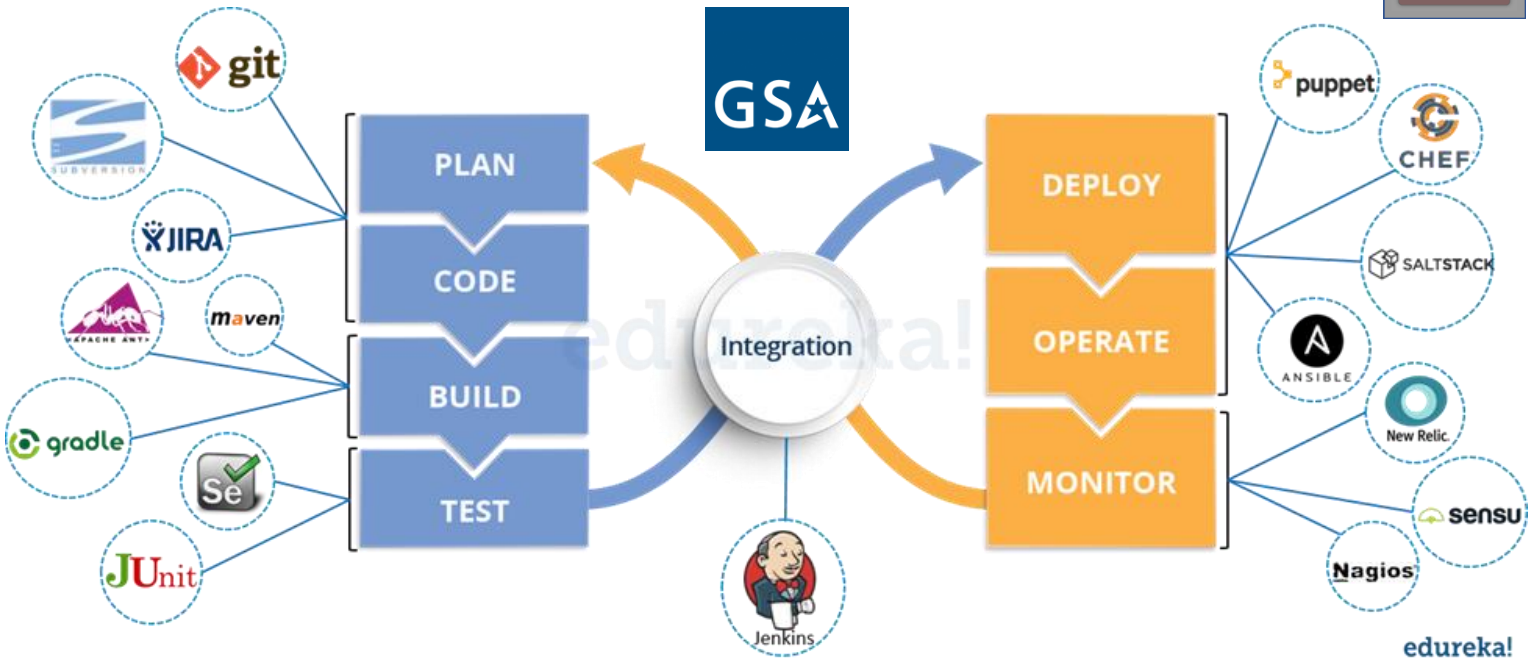
Vulnerability Scanning

Static Code Analysis
Code Quality
Code Smells
Package/Lib dependency scanning

H.2.6a DevSecOps: Tooling Selection and Implementation Strategy

Dimensions of DevSecOps Maturity Model

1. **DevSecOps Security Maturity Model**: identify one to be applied and executed by the company
   - No exotic: common in the market, usable and maintained (e.g. OWASP one)
   - Agree in the company aboout adoption
2. **Tools & 3° Party Products**
   1. For each level in levels in Maturity Model
   2. For each phase in CI/CD
   3. Already in use by Dev, QA, Operation
   4. Getting suggestions (tools already known, used) by Dev, QA, Operation
3. **Put it All Together**: Model + Tools best suites to the company
   - Usefulness
   - (possible) Already in Use
   - Required Knowledge
   - Resources (systems and time)



DevSecOps Tools at U.S. General Service Administration (https://tech.gsa.gov/guides/building_devsecops_culture/)

- **Consensus per Tools** → ensure usage
- Deploy **one Tool at once** → time to learn, adapt and get familiar with processes
- Implement **each Maturity Level at once** → digested by stakeholders

H.2.6b DevSecOps: Tooling Selection and Implementation Strategy

Merge DevOps and Security - #1 Plan

1. **Best Coding Standards**: for the chosen programming language
2. **Strategy for using Open-Source**: due to library vulnerabilities
3. **Strategy for using 3° Party**: due to library vulnerabilities
4. **Threat Model**: identify and define an execution plan suitable for the company
5. **Check Compliance**: compare regulations and SLA of services, vendors and public cloud

H.2.6c DevSecOps: Tooling Selection and Implementation Strategy

Merge DevOps and Security - #2 Code

1. **SAST**: Static Application Security Testing, checking source code against existing vulnerabilities
2. **Code Quality**: checking against the metrics defined in the #1 Plan
3. **Bad Coding**: identifying security weak coding
4. **Training**: encourage the adoption of clean code standards

H.2.6d DevSecOps: Tooling Selection and Implementation Strategy

Merge DevOps and Security - #3 Build

1. **SCA**: SW Composition Analysis, checking open-source library dependencies against existing vulns
2. **Chain Resolution**: scanning for dependencies (Lib-A → Lib-B → Lib-C → …)
3. **Scanning**: Artifact Repository, Container Images, Code Quality, Smells

H.2.6e DevSecOps: Tooling Selection and Implementation Strategy

Merge DevOps and Security - #4 Test

1. **PenTest**:
2. **Load Testing**
3. **Fuzzing**: Fuzz Testing (Black Box)
4. **BDD**: Behaviour Driven Development testing (extension of TDD)
5. **Integration Testing**: internal or 3° Party API

H.2.6f DevSecOps: Tooling Selection and Implementation Strategy

Merge DevOps and Security - #5 Release

1. **Artifactory Management**: select and secure
   - Tenancy
   - Creating projects/tenants
   - RBAC
   - RACI matrix
2. **DAST**: Dynamic Application Security Testing
   - OWASP ZAP
   - Arachni Scanner

H.2.6g DevSecOps: Tooling Selection and Implementation Strategy

Merge DevOps and Security - #6 Deploy #7 Production

1. **Infrastructure**: select and secure
   - VMs
   - Containers
   - Networks
   - Storage
   - RACI matrix
2. **Compliance**: check adherence to requirements

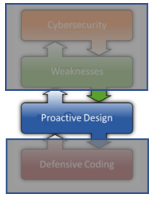
H.2.6h DevSecOps: Tooling Selection and Implementation Strategy

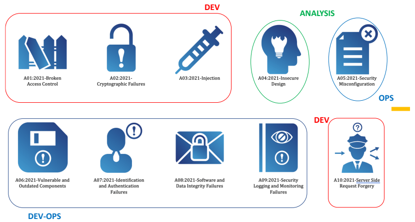
Merge DevOps and Security - #8 Monitor

1. **Centralized Logging and Monitoring**:
   - Log Analytics
   - Log Security Pattern Detection
2. **Security Threats**: Monitoring
   - Network and System level
   - API
3. **Alerting**:
   1. SIEM
   2. SOAR

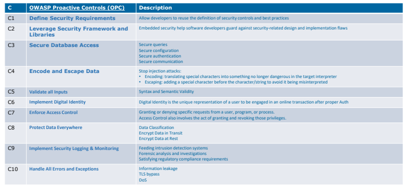
H.2.7 DevSecOps: Approach

Main Issues



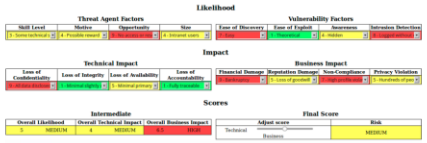

DEV
ANALYSIS
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
A04:2021-Insecure Design
A05:2021-Security Misconfiguration
OPS
DEV
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures
A10:2021-Server Side Request Forgery
DEV-OPS

Responsibility Border: No Boundaries among IT elements (O.S., Middleware, Libraries, Custom Application, etc (Atos Development is not in charge of elements provided by other companies/functions)



C	OWASP Proactive Controls (OPC)	Description
C1	Define Security Requirements	Allow developers to reuse the definition of security controls and best practices
C2	Leverage Security Framework and Libraries	Embedded security help software developers guard against security-related design and implementation flaws
C3	Secure Database Access	Secure queries Secure configuration Secure authentication Secure communication
C4	Encode and Escape Data	Stop injection attacks: • Encoding: translating special characters into something no longer dangerous in the target interpreter • Escaping: adding a special character before the character/string to avoid it being misinterpreted
C5	Validate all Inputs	Syntax and Semantic Validity
C6	Implement Digital Identity	Digital Identity is the unique representation of a user to be engaged in an online transaction after proper Auth
C7	Enforce Access Control	Granting or denying specific requests from a user, program, or process. Access Control also involves the act of granting and revoking those privileges.
C8	Protect Data Everywhere	Data Classification Encrypt Data in Transit Encrypt Data at Rest
C9	Implement Security Logging & Monitoring	Feeding intrusion detection systems Forensic analysis and investigations Satisfying regulatory compliance requirements
C10	Handle All Errors and Exceptions	Information leakage TLS bypass DoS

Exhaustive Integration: Insufficient identification of Security Requirements about thoroughly meshing applications, since the analysis phase, into the customer environment in order to get compliance, ergonomics, easy security, quick&dirty manageability, etc



Likelihood

Threat Agent Factors				Vulnerability Factors			
Skill Level	Motive	Opportunity	Size	Ease of Discovery	Ease of Exploit	Awareness	Intrusion Detection
3 - Some technical s	4 - Possible reward	9 - No access or res	4 - Intranet users	7 - Easy	1 - Theoretical	4 - Hidden	8 - Logged without

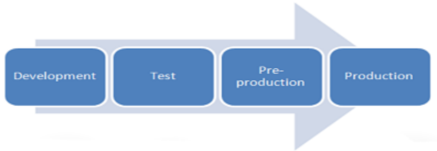
Impact

Technical Impact				Business Impact			
Loss of Confidentiality	Loss of Integrity	Loss of Availability	Loss of Accountability	Financial Damage	Reputation Damage	Non-Compliance	Privacy Violation
9 - All data disclosed	1 - Minimal slightly	5 - Minimal primary	1 - Fully traceable	9 - Bankruptcy	5 - Loss of goodwill	7 - High profile viola	5 - Hundreds of peo

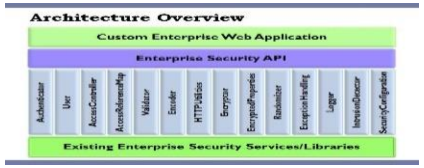
Scores

Intermediate			Final Score	
Overall Likelihood	Overall Technical Impact	Overall Business Impact	Adjust score	Risk
5 MEDIUM	4 MEDIUM	6.5 HIGH	Technical Business	MEDIUM

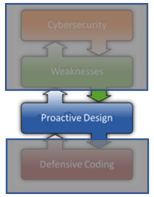
Security Risk: only vulnerability exposures are encompassed. IT Risk is also about lacking Proactive Controls, Using Know Vulnerable Components (A9), Security Misconfiguration (A5)



Testing Environment: no complete security test before installation in customer environment. Security requirement for underpinning elements.

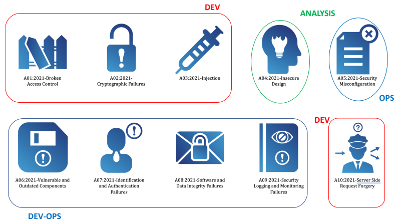


SW Library: Not clear utilization of standard security library (e.g. ESAPI) as default behaviour

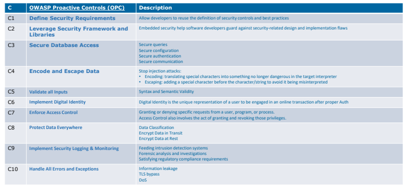


H.2.7 DevSecOps: Approach

Main Issues

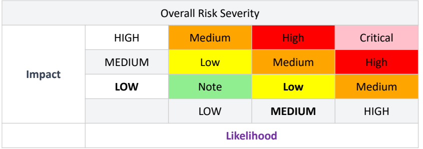

DEV
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
ANALYSIS
A04:2021-Insecure Design
A05:2021-Security Misconfiguration
OPS
DEV
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures
A10:2021-Server Side Request Forgery
DEV-OPS

Responsibility Border: ➔ OWASP Top Ten (Top10) A1, A2, A3, A4, A6, A7, A8, A10. **NO**: A5, A9.



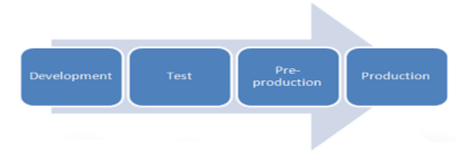
C	OWASP Proactive Controls (OPC)	Description
C1	Define Security Requirements	Allow developers to reuse the definition of security controls and best practices
C2	Leverage Security Framework and Libraries	Embedded security help software developers guard against security-related design and implementation flaws
C3	Secure Database Access	Secure queries Secure configuration Secure authentication Secure communication
C4	Encode and Escape Data	Stop injection attacks: • Encoding: translating special characters into something no longer dangerous in the target interpreter • Escaping: adding a special character before the character/string to avoid it being misinterpreted
C5	Validate all Inputs	Syntax and Semantic Validity
C6	Implement Digital Identity	Digital Identity is the unique representation of a user to be engaged in an online transaction after proper Auth
C7	Enforce Access Control	Granting or denying specific requests from a user, program, or process. Access Control also involves the act of granting and revoking those privileges.
C8	Protect Data Everywhere	Data Classification Encrypt Data in Transit Encrypt Data at Rest
C9	Implement Security Logging & Monitoring	Feeding intrusion detection systems Forensic analysis and investigations Satisfying regulatory compliance requirements
C10	Handle All Errors and Exceptions	Information leakage TLS bypass DoS

Exhaustive Integration: ➔ OWASP Proactive Controls (OPC)

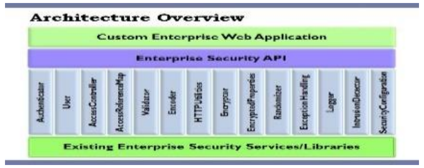


Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Security Risk: ➔ OWASP Risk Rating Methodology



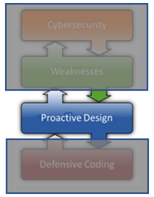
Testing Environment: ➔ DAST on specific Testing Environment OWASP Testing Merthodology



SW Library: ➔ OWASP ESAPI

H.2.7a DevSecOps: Approach

Main Issues

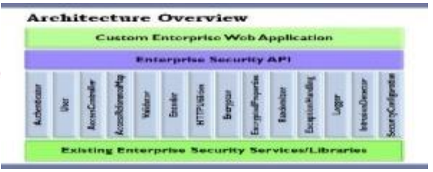


OWASP Technical Controls

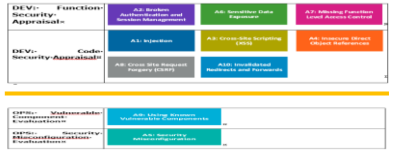
Proactive Controls

- C1: Define Security Requirements
- C2: Leverage Security Frameworks and Libraries
- C3: Secure Database Access
- C4: Encode and Escape Data
- C5: Validate All Inputs
- C6: Implement Digital Identity
- C7: Enforce Access Controls
- C8: Protect Data Everywhere
- C9: Implement Security Logging and Monitorin
- C10: Handle All Errors and Exceptions

ESAPI

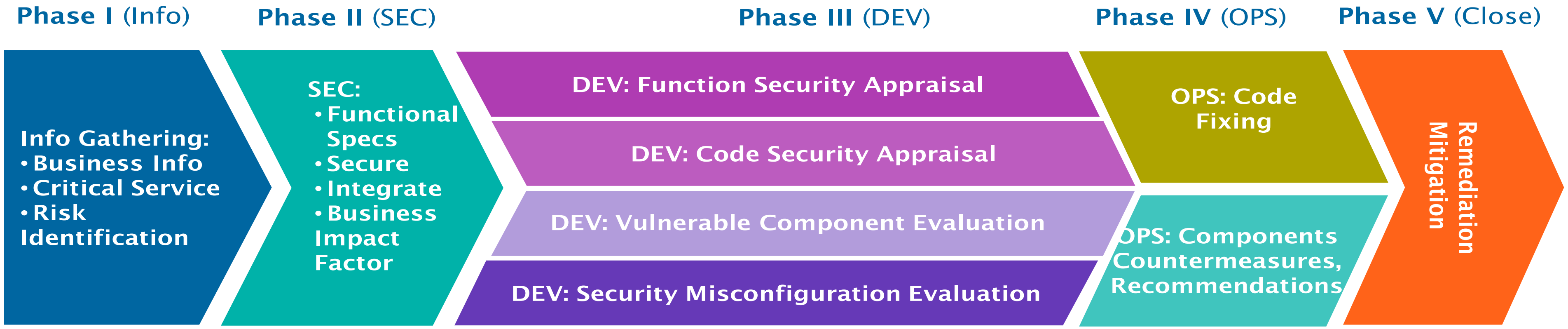


Top Ten



DEV: Function Security Appraisal	A2: Broken Authentication and Session Management	A6: Sensitive Data Exposure	A7: Missing Function Level Access Control
DEV: Code Security Appraisal	A1: Injection	A3: Cross-Site Scripting (XSS)	A4: Insecure Direct Object References
	A8: Cross Site Request Forgery (CSRF)	A10: Invalidated Redirects and Forwards	

OPS: Vulnerable Component Evaluation	A9: Using Known Vulnerable Components
OPS: Security Misconfiguration Evaluation	A5: Security Misconfiguration



Phase I (Info)

Info Gathering:
- Business Info
- Critical Service
- Risk Identification

Phase II (SEC)

SEC:
- Functional Specs
- Secure
- Integrate
- Business Impact Factor

Phase III (DEV)

DEV: Function Security Appraisal

DEV: Code Security Appraisal

DEV: Vulnerable Component Evaluation

DEV: Security Misconfiguration Evaluation

Phase IV (OPS)

OPS: Code Fixing

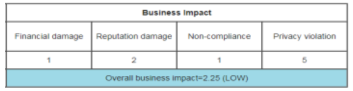
OPS: Components Countermeasures, Recommendations

Phase V (Close)

Remediation Mitigation

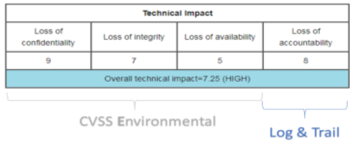
OWASP Risk Rating Methodology

Impact



Business Impact			
Financial damage	Reputation damage	Non-compliance	Privacy violation
1	2	1	5
Overall business impact=2.25 (LOW)			

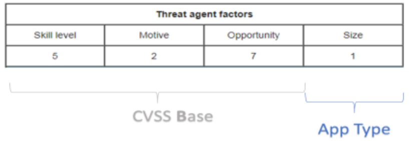
Likelihood



Technical Impact			
Loss of confidentiality	Loss of integrity	Loss of availability	Loss of accountability
9	7	5	8
Overall technical impact=7.25 (HIGH)			

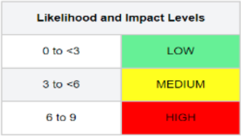
CVSS Environmental
Log & Trail

Scores



Threat agent factors			
Skill level	Motive	Opportunity	Size
5	2	7	1

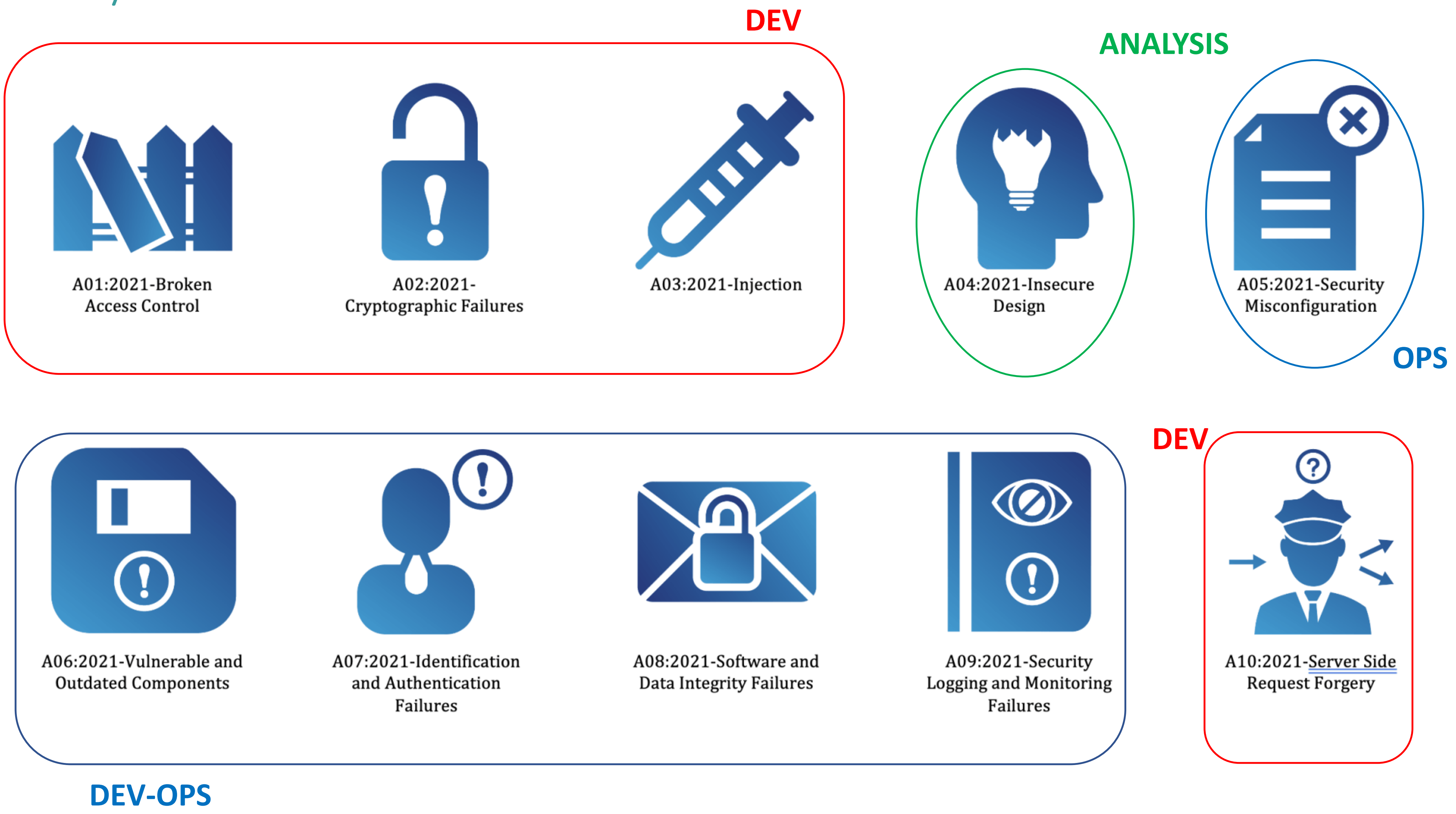
CVSS Base
App Type



Likelihood and Impact Levels	
0 to <3	LOW
3 to <6	MEDIUM
6 to 9	HIGH

H.2.7b DevSecOps: Approach

Responsibility Border



DEV

A01:2021-Broken Access Control

A02:2021-Cryptographic Failures

A03:2021-Injection

ANALYSIS

A04:2021-Insecure Design

A05:2021-Security Misconfiguration

OPS

A06:2021-Vulnerable and Outdated Components

A07:2021-Identification and Authentication Failures

A08:2021-Software and Data Integrity Failures

A09:2021-Security Logging and Monitoring Failures

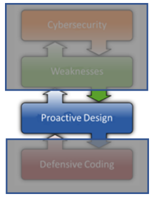
DEV-OPS

DEV

A10:2021-Server Side Request Forgery

H.2.7c DevSecOps: Approach

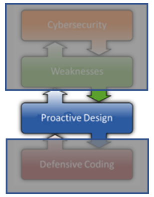
OWASP Proactive Controls (OPC)



C	OWASP Proactive Controls (OPC)	Description
C1	**Define Security Requirements**	Allow developers to reuse the definition of security controls and best practices
C2	**Leverage Security Framework and Libraries**	Embedded security help software developers guard against security-related design and implementation flaws
C3	**Secure Database Access**	Secure queries Secure configuration Secure authentication Secure communication
C4	**Encode and Escape Data**	Stop injection attacks: • Encoding: translating special characters into something no longer dangerous in the target interpreter • Escaping: adding a special character before the character/string to avoid it being misinterpreted
C5	**Validate all Inputs**	Syntax and Semantic Validity
C6	**Implement Digital Identity**	Digital Identity is the unique representation of a user to be engaged in an online transaction after proper Auth
C7	**Enforce Access Control**	Granting or denying specific requests from a user, program, or process. Access Control also involves the act of granting and revoking those privileges.
C8	**Protect Data Everywhere**	Data Classification Encrypt Data in Transit Encrypt Data at Rest
C9	**Implement Security Logging & Monitoring**	Feeding intrusion detection systems Forensic analysis and investigations Satisfying regulatory compliance requirements
C10	**Handle All Errors and Exceptions**	Information leakage TLS bypass DoS

H.2.7c1 DevSecOps: Approach

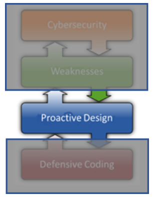
Cheat Sheets – Reference Tools 1/2



C	OWASP Proactive Controls (OPC)	Cheat Sheet	Reference/Tool
C1	**Define Security Requirements**	• Abuse Case Cheat Sheet • Attack Surface Analysis Cheat Sheet • Threat Modeling Cheat Sheet	• ASVS
C2	**Leverage Security Framework and Libraries**	• Vulnerable Dependency Management Cheat Sheet	• OWASP Dependency Check 4 Maven; • Retire.js.
C3	**Secure Database Access**	• Database CheatSheet • Query Parameterization Cheat Sheet • SQL Injection Prevention Cheat Sheet	
C4	**Encode and Escape Data**	• Cross Site Scripting Prevention Cheat Sheet; • OWASP Injection Prevention Cheat Sheet in Java.	• OWASP Java Encoder;
C5	**Validate all Inputs**	• Input Validation Cheat Sheet.	
C6	**Implement Digital Identity**	• OWASP Authentication Cheat Sheet; • OWASP Password Storage Cheat Sheet; • OWASP Forgot Password Cheat Sheet; • OWASP Choosing and Using Security Question Cheat Sheet; • OWASP Session Manager Cheat Sheet; • OWASP IoS Developer Cheat Sheet;	• OWASP Mobile Security Testing Guide; • OWASP Testing for Authentication Guide; • NIST Special Publication 800-63 Revision 3 - Digital Identity Guidelines.

H.2.7c2 DevSecOps: Approach

Cheat Sheets – Reference Tools 2/2



C	OWASP Proactive Controls (OPC)	Cheat Sheet	Reference/Tool
C7	Enforce Access Control	• OWASP Access Control Cheat Sheet; • OWASP iOS Developer - Poor Authorization and Authentication Cheat Sheet; • OWASP Testing for Authorization Guide.	• OWASP ZAP • Access Control Testing
C8	Protect Data Everywhere	• OWASP TLP Cheat Sheet • OWASP HSTS Cheat Sheet • OWASP Cryptographic Storage Cheat Sheet • OWASP Password Storage Cheat Sheet • OWASP IOS Developer - Insecure Data Storage Cheat Sheet	• Ivan Ristic: SSL/TLS Deployment Best Practices • OWASP Testing Guide: Testing for TLS • SSLyze - scanning library and CLI tool • SSLLabs - scan and checkTLS/SSL conf • OWASP O-Saft TLS Tool - TLS test tool • GitRob - find sensitive info on GitHub • TruffleHog - Searches for secrets accidentally committed • KeyWhiz - Secrets manager • Hashicorp Vault - Secrets manager • Amazon KM - Manage keys on Amazon AWS
C9	Implement Security Logging & Monitoring	• OWASP Logging Cheat Sheet • OWASP Application Logging Vocabulary Cheat Sheet	• OWASP Log injection • Apache Logging Services
C10	Handle All Errors and Exceptions	• OWASP REST Security Cheat Sheet (Error Handling) • OWASP Error Handling Cheat Sheet	• OWASP Code Review Guide: Error Handling • OWASP Testing Guide: Testing for Error Handling • OWASP Improper Error Handling • CWE 209: Information Exposure Through an Error Message • CWE 391: Unchecked Error Condition

H.2.7d DevSecOps: Approach

Security Risk

E4g Security Risk: Rating

OWASP Risk Rating Methodology - Estimation

Step 4: Determining the Severity of the Risk

The likelihood estimate and the impact estimate are put together to calculate an overall severity for this risk.

Likelihood and Impact Levels	
0 to <3	LOW
3 to <6	MEDIUM
6 to 9	HIGH

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	**Low**	Medium
		LOW	**MEDIUM**	HIGH
	Likelihood			

Determining Severity

The tester can now combine the likelihood and impact estimates to get a final severity rating for this risk.

If there is good business impact information, it is better to use that instead of the technical impact information

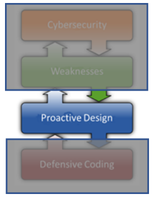
In the example:

Overall **Likelihood** = 4.375 (MEDIUM)

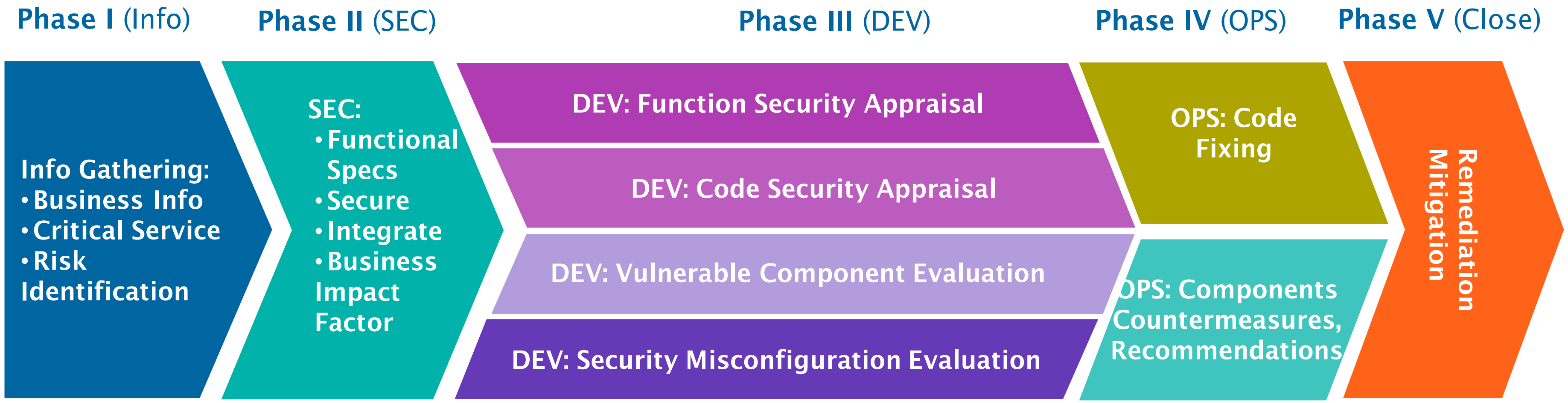
Business **Impact** = 2.250 (LOW)

H.2.7e DevSecOps: Approach

Risk among phases



OWASP Risk Rating Methodology



Phase I (Info)

Info Gathering:
- **Business Info**
- **Critical Service**
- **Risk Identification**

Phase II (SEC)

SEC:
- **Functional Specs**
- **Secure**
- **Integrate**
- **Business Impact Factor**

Phase III (DEV)

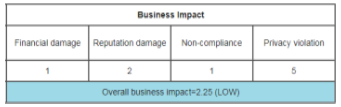
- **DEV: Function Security Appraisal**
- **DEV: Code Security Appraisal**
- **DEV: Vulnerable Component Evaluation**
- **DEV: Security Misconfiguration Evaluation**

Phase IV (OPS)

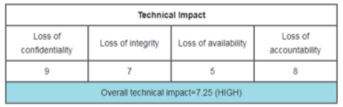
- **OPS: Code Fixing**
- **OPS: Components Countermeasures, Recommendations**

Phase V (Close)

Remediation Mitigation



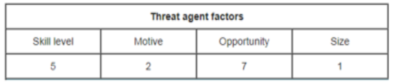
Business Impact			
Financial damage	Reputation damage	Non-compliance	Privacy violation
1	2	1	5
Overall business impact=2.25 (LOW)			



Technical Impact			
Loss of confidentiality	Loss of integrity	Loss of availability	Loss of accountability
9	7	5	8
Overall technical impact=7.25 (HIGH)			

CVSS Environmental

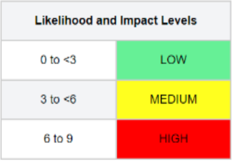
Log & Trail



Threat agent factors			
Skill level	Motive	Opportunity	Size
5	2	7	1

CVSS Base

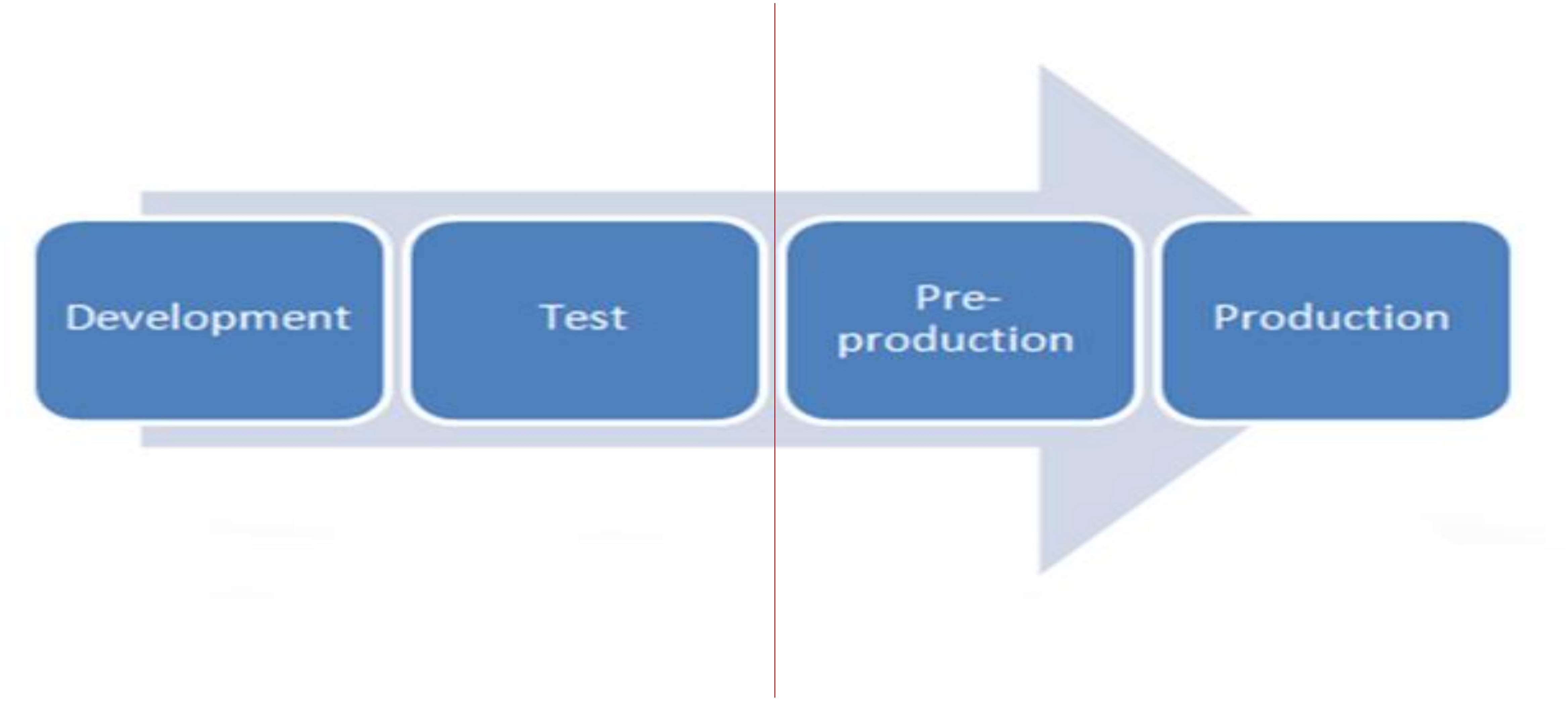
App Type



Likelihood and Impact Levels	
0 to <3	LOW
3 to <6	MEDIUM
6 to 9	HIGH

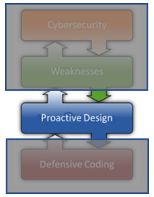
H.2.7e DevSecOps: Approach

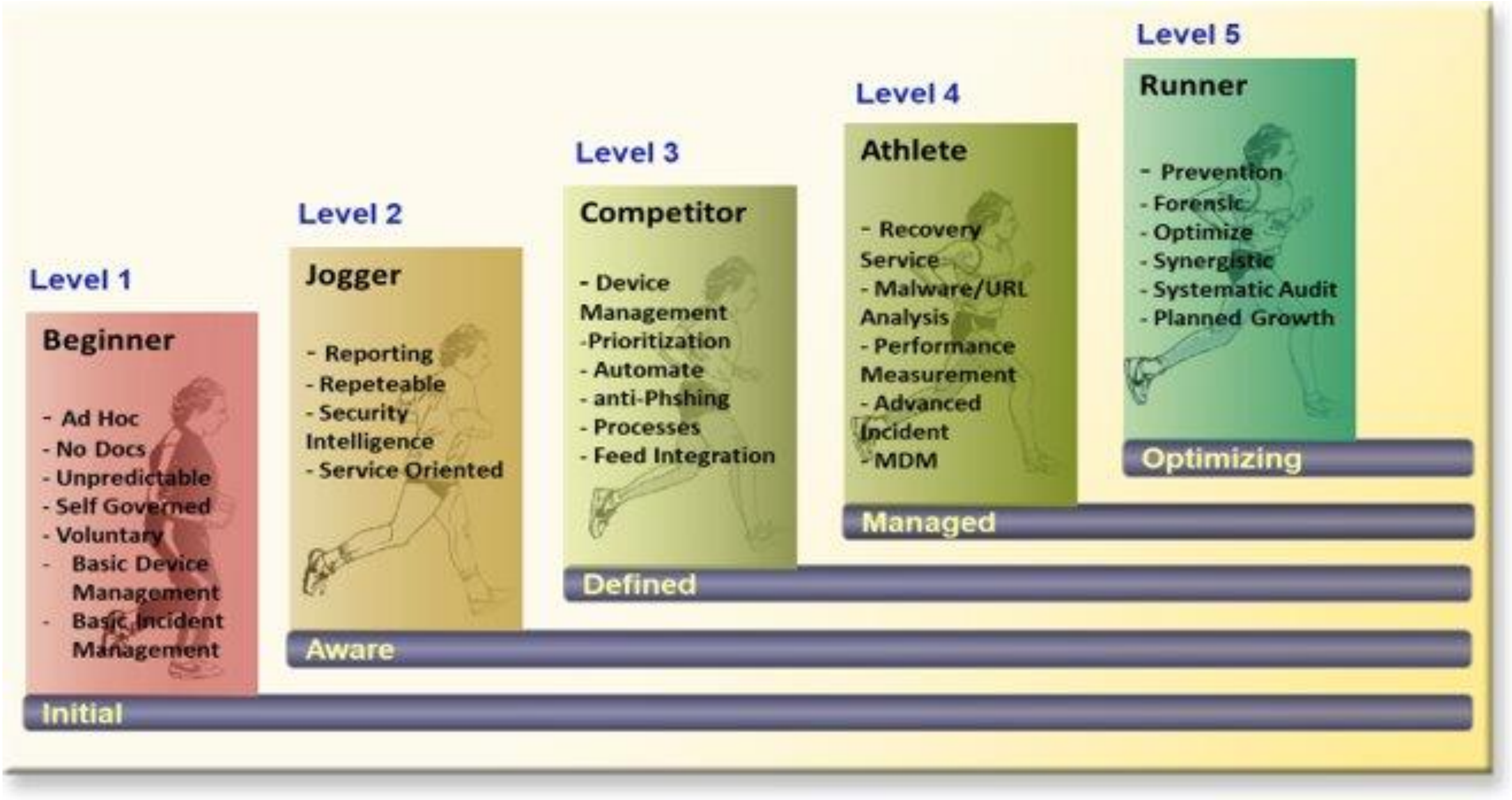
DAST on specific testing environment



H.3 DevSecOps: Framework

Benefit of Maturity Models





- **Strategy**: placing
  - Abstration Chart for Higher Management
  - Factual Requirements for Security Manager
  - Technical Guidance for employee
- **Path**: drawing a path for further improvements
  - Clear goals for everyone in the organization → target
  - Allowing continous improvements → milestones
- **Evaluation**: checkpoints for meeting objectives
  - Metrics for objectiveness
  - Proactive (continously improving) not more Reactive (acting on security issues)
- **Visibility**: risk determination
  - Prioritization of the weak spots to be strenghtened
- **Savings**: remove redundancies
  - Replacements needs
  - Processes (people skills, technologies)

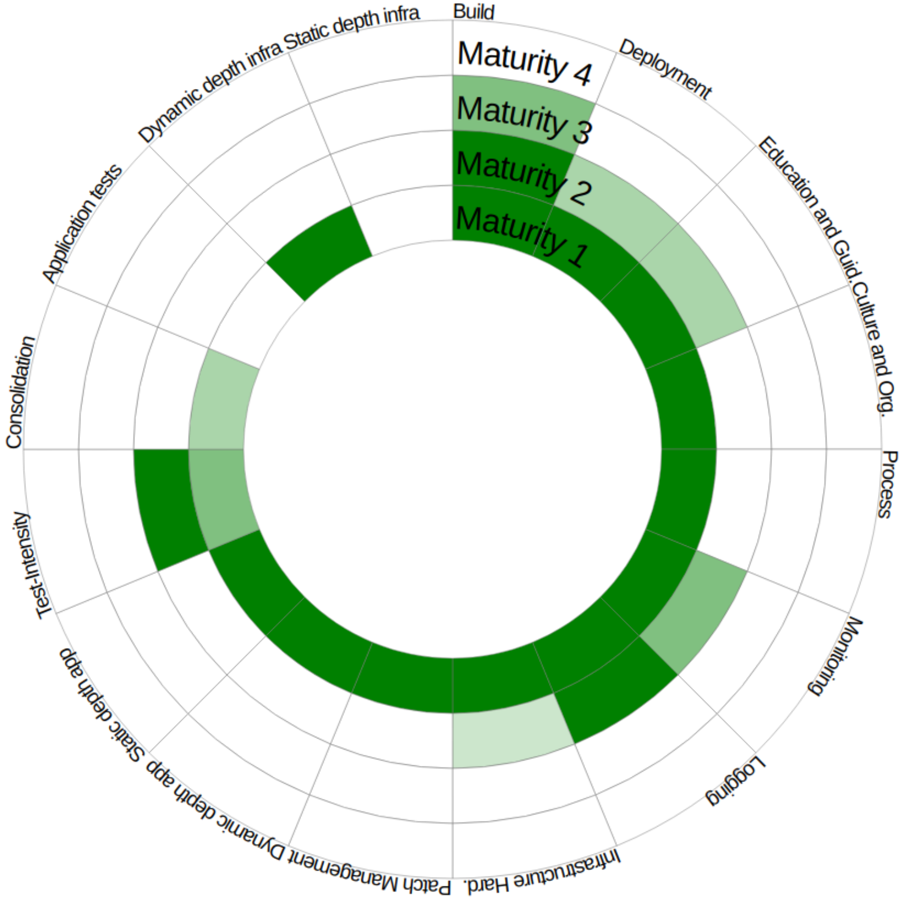
H.3a DevSecOps: OWASP DSOMM

Levels of DevSecOps Maturity Model

OWASP: https://owasp.org/www-project-devsecops-maturity-model/

DSOMM: Dev Sec Ops Maturity Model

Identification of the degree of the implementation



4 Levels:

1. **Basic Understanding of Security Practices**: basic GRC, some controls in development environment
2. **Adoption of Basic Security Practices**: login audit, static check, hardening
3. **High Adoption of Security Practices**: Infrastructure as Code (IaC), Dashboard (advanced metrics), Code Signing
4. **Advanced Deployment of Security Practices at Scale**: improving Advanced Threat Model and History, Defense Metrics

H.3a1 DevSecOps: OWASP DSOMM

Dimensions of DevSecOps Maturity Model

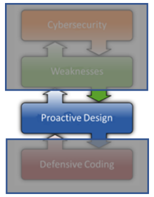
	Dimension	Sub-Dimension
	Build and Deployment	Build Deployment Patch Management
	Culture and Organization	Design Education and Guidance Process
	Implementation	Application Hardening Development and Source Control Infrastructure Hardening
	Information Gathering	Logging Monitoring
	Test and Verification	Application Tests Consolidation Dynamic/Static Depth for Applications/Infrastrucure Test-Intensity

DSOMM

https://dsomm.timo-pagel.de/

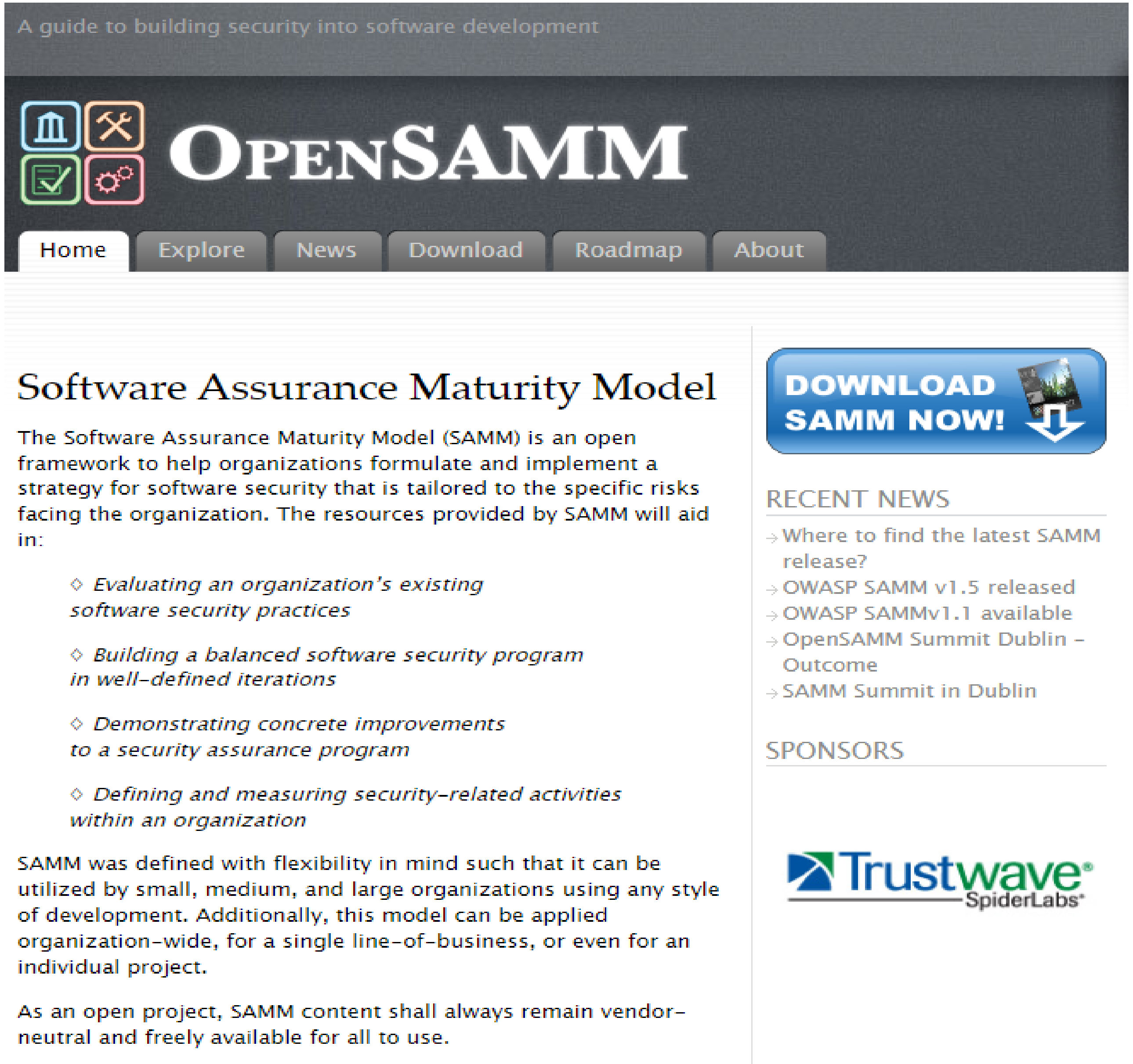
H.3b OpenSAMM and its descendants

Software Assurance Maturity Model



OpenSAMM: https://www.opensamm.org/

OpenSAMM was created by Pravir Chandra and sponsored by Fortify. Fortify has then donated OpenSAMM to the OWASP community. Both BSIMM and SAMM originate from OpenSAMM. Both models still contain some similarities, but follow different approaches to application security.



Software Assurance Maturity Model

The Software Assurance Maturity Model (SAMM) is an open framework to help organizations formulate and implement a strategy for software security that is tailored to the specific risks facing the organization. The resources provided by SAMM will aid in:

- *Evaluating an organization's existing software security practices*
- *Building a balanced software security program in well-defined iterations*
- *Demonstrating concrete improvements to a security assurance program*
- *Defining and measuring security-related activities within an organization*

SAMM was defined with flexibility in mind such that it can be utilized by small, medium, and large organizations using any style of development. Additionally, this model can be applied organization-wide, for a single line-of-business, or even for an individual project.

As an open project, SAMM content shall always remain vendor-neutral and freely available for all to use.

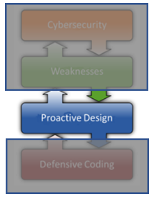
RECENT NEWS

- Where to find the latest SAMM release?
- OWASP SAMM v1.5 released
- OWASP SAMMv1.1 available
- OpenSAMM Summit Dublin – Outcome
- SAMM Summit in Dublin

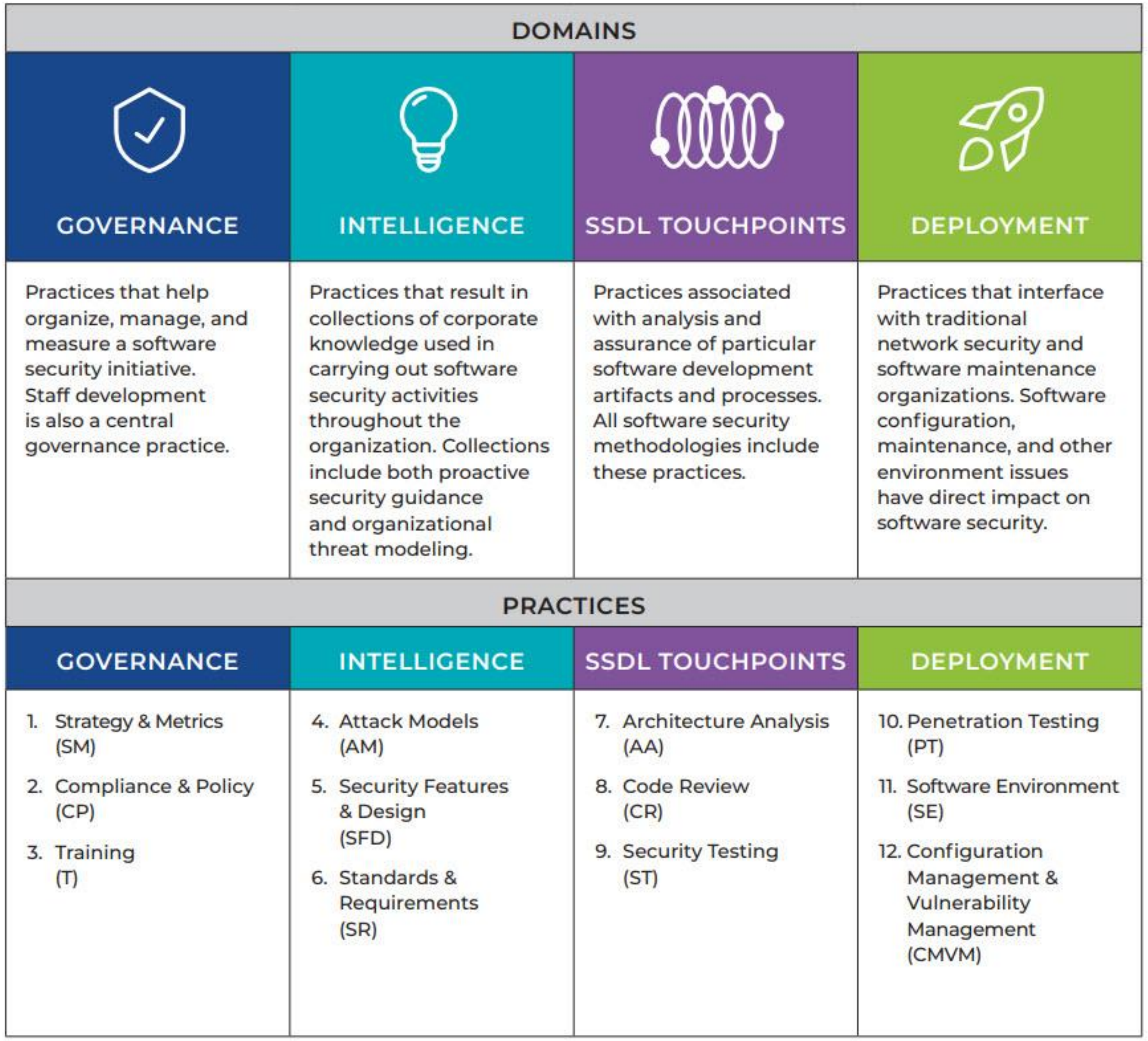
SPONSORS

H.3b1 OpenSAMM and its descendants

Building Security In Maturity Model (BSIMM)



BSIMM: https://www.synopsys.com/software-integrity/software-security-services/bsimm-maturity-model.html



DOMAINS			
GOVERNANCE	INTELLIGENCE	SSDL TOUCHPOINTS	DEPLOYMENT
Practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.	Practices that result in collections of corporate knowledge used in carrying out software security activities throughout the organization. Collections include both proactive security guidance and organizational threat modeling.	Practices associated with analysis and assurance of particular software development artifacts and processes. All software security methodologies include these practices.	Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance, and other environment issues have direct impact on software security.
PRACTICES			
GOVERNANCE	INTELLIGENCE	SSDL TOUCHPOINTS	DEPLOYMENT
1. Strategy & Metrics (SM) 2. Compliance & Policy (CP) 3. Training (T)	4. Attack Models (AM) 5. Security Features & Design (SFD) 6. Standards & Requirements (SR)	7. Architecture Analysis (AA) 8. Code Review (CR) 9. Security Testing (ST)	10. Penetration Testing (PT) 11. Software Environment (SE) 12. Configuration Management & Vulnerability Management (CMVM)

BSIMM is a maturity model that helps organizations plan, implement and measure their software security assurance programme. BSIMM consists of 4 domains split in 12 practices and containing a total of 125 security activities. So think of pen testing, patching, monitoring tools and threat modeling as some of these 125 activities you could (but not always should) do in your security assurance programme. Here is a structural overview of the BSIMM13 domains and practices.

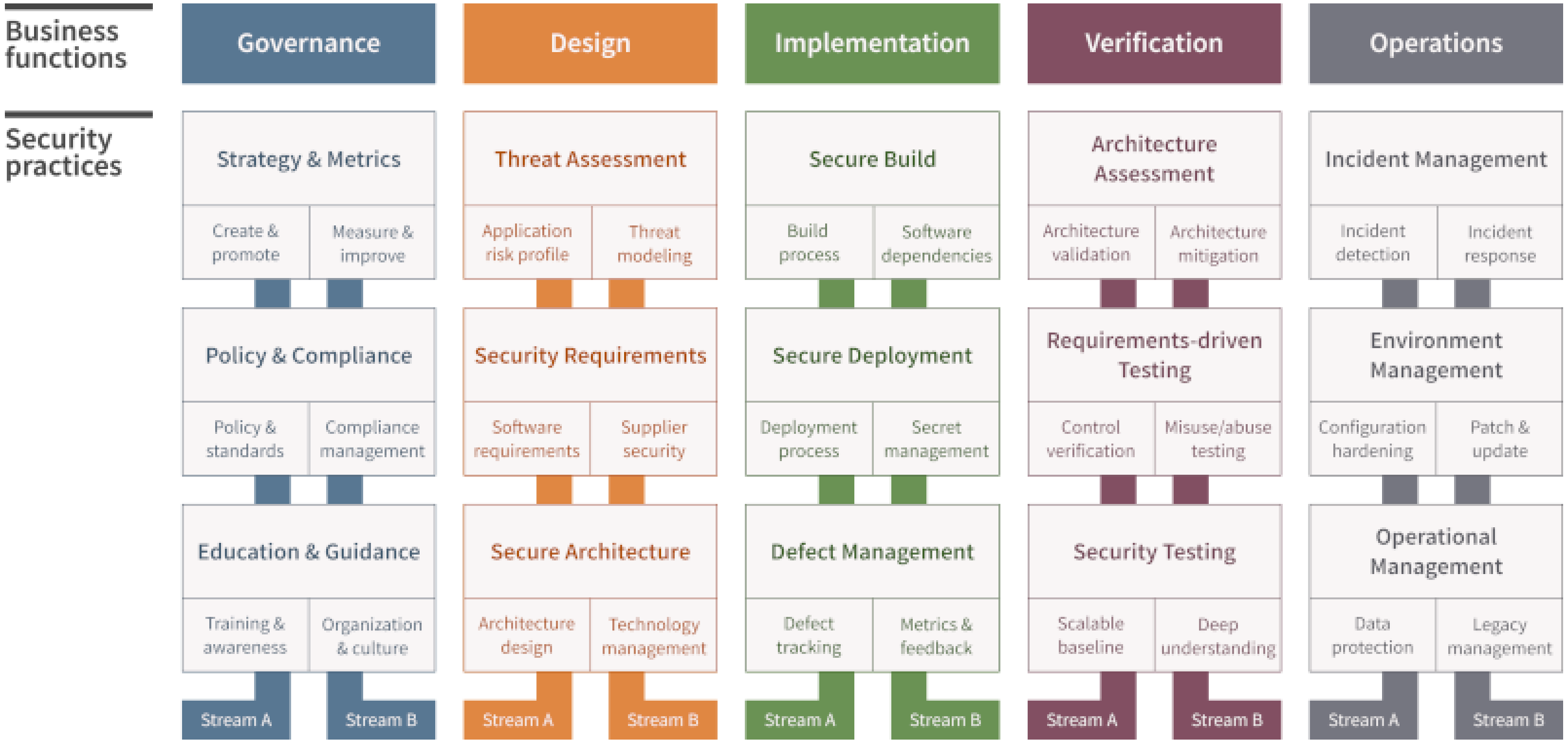
BSIMM is not only the framework, but is also a measuring stick in the industry. BSIMM comes with an objective assessment of the different activities in 130 organizations from 8 industry verticals (financial services, independent software vendors, technology, healthcare, cloud, Internet of Things, insurance, and retail).

H.3b2 OpenSAMM and its descendants

Software Assurance Maturity Model (SAMM)

BSIMM: https://owasp.org/www-project-samm/

SAMM is a maturity model that provides an effective and measurable way for all types of organizations to analyze and improve their software security posture. SAMM consists of 5 business functions split over 15 security practices and containing a total of 90 security activities. Here is a structural overview of SAMM functions and practices.

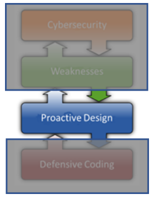


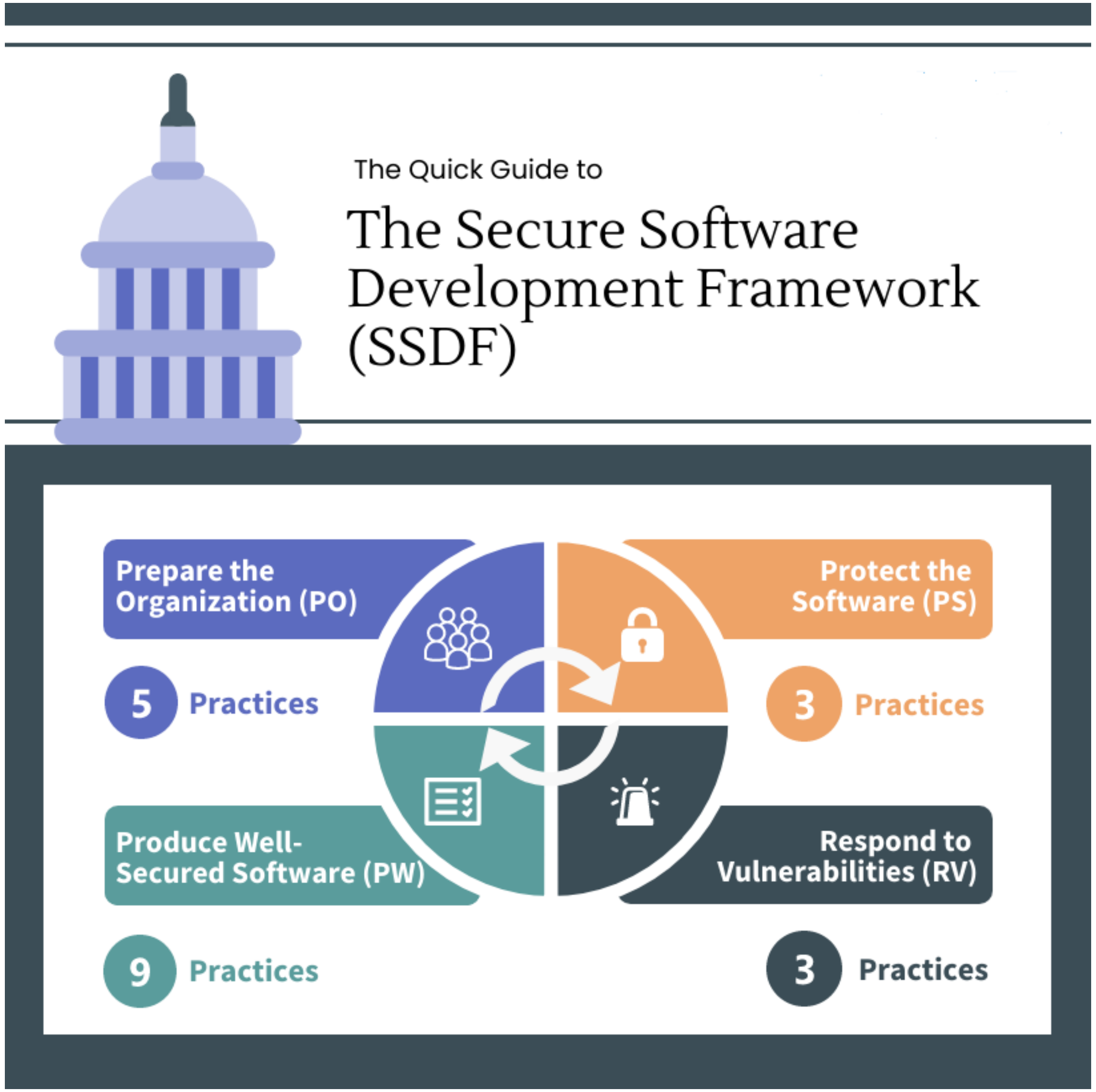
Business functions	Governance	Design	Implementation	Verification	Operations
Security practices	Strategy & Metrics	Threat Assessment	Secure Build	Architecture Assessment	Incident Management
	Create & promote Measure & improve	Application risk profile Threat modeling	Build process Software dependencies	Architecture validation Architecture mitigation	Incident detection Incident response
	Policy & Compliance	Security Requirements	Secure Deployment	Requirements-driven Testing	Environment Management
	Policy & standards Compliance management	Software requirements Supplier security	Deployment process Secret management	Control verification Misuse/abuse testing	Configuration hardening Patch & update
	Education & Guidance	Secure Architecture	Defect Management	Security Testing	Operational Management
	Training & awareness Organization & culture	Architecture design Technology management	Defect tracking Metrics & feedback	Scalable baseline Deep understanding	Data protection Legacy management
	Stream A Stream B	Stream A Stream B	Stream A Stream B	Stream A Stream B	Stream A Stream B

SAMM is planned to include the upcoming Benchmarking project that will also allow one to compare your own security posture with the rest of the industry.

H.3c NIST SSDF (NIST SP800-218)

Secure Software Development Framework - https://csrc.nist.gov/publications/detail/sp/800-218/final





The Quick Guide to
The Secure Software Development Framework (SSDF)
Prepare the Organization (PO)
5 Practices
Protect the Software (PS)
3 Practices
Produce Well-Secured Software (PW)
9 Practices
Respond to Vulnerabilities (RV)
3 Practices

NIST SSDF is a security assurance programme to be integrated within the software development lifecycle (SDLC).

SSDF consists of 19 security practices divided across 42 tasks, covering 42 topics in software security to get the attestation required by the Feds. These are about security best practices. Examples include:

- mandatory and role-specific security trainings for the team,
- identifying and documenting all security requirements,
- running threat modeling and risk assessment exercises.

Security assurance frameworks are relatively abstract to remain applicable. But NIST SSDF has a complete mapping to OWASP SAMM. Inversely, SAMM has a complete mapping to SSDF.

➔ **implement SAMM** and automatically **check SSDF compliance**.