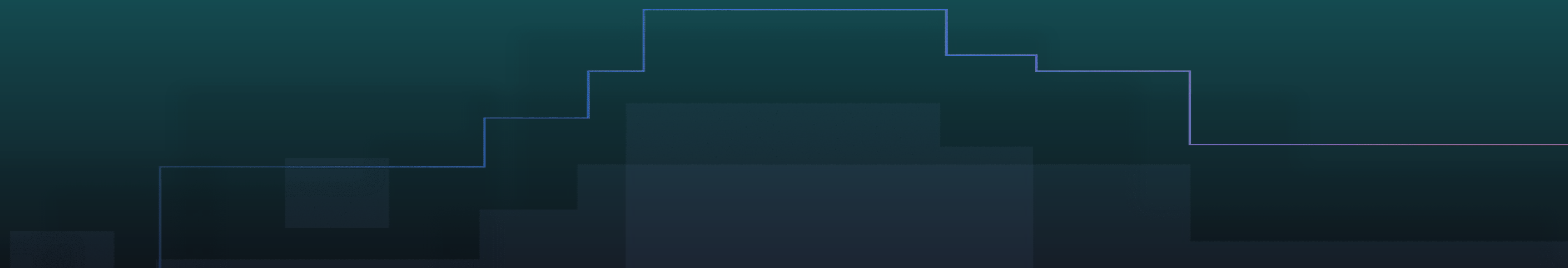
GameSS

EDUCATIONAL MATERIAL IN CYBER SECURITY



WHO IS THE MATERIAL FOR?

This material provides a generic introduction to the concepts of DevSecOps.
It is suitable for professionals such as

- software developers,
- system administrators,
- operations engineers,
- security professionals,
- IT managers,
- responsible for IT software or equipment acquisition.

WHO MADE THIS MATERIAL?

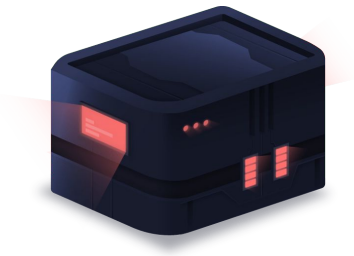
Jacopo Mauro

Professor at the Department of Mathematics and Computer Science at the University of Southern Denmark (SDU)

mauro@imada.sdu.dk

WHAT ARE THE MAIN TAKEAWAYS FOR THIS CONTENT?

- What DevSecOps and SRE are about
- Automation is key for security
- Guidelines and best practices to inject security in the software pipeline



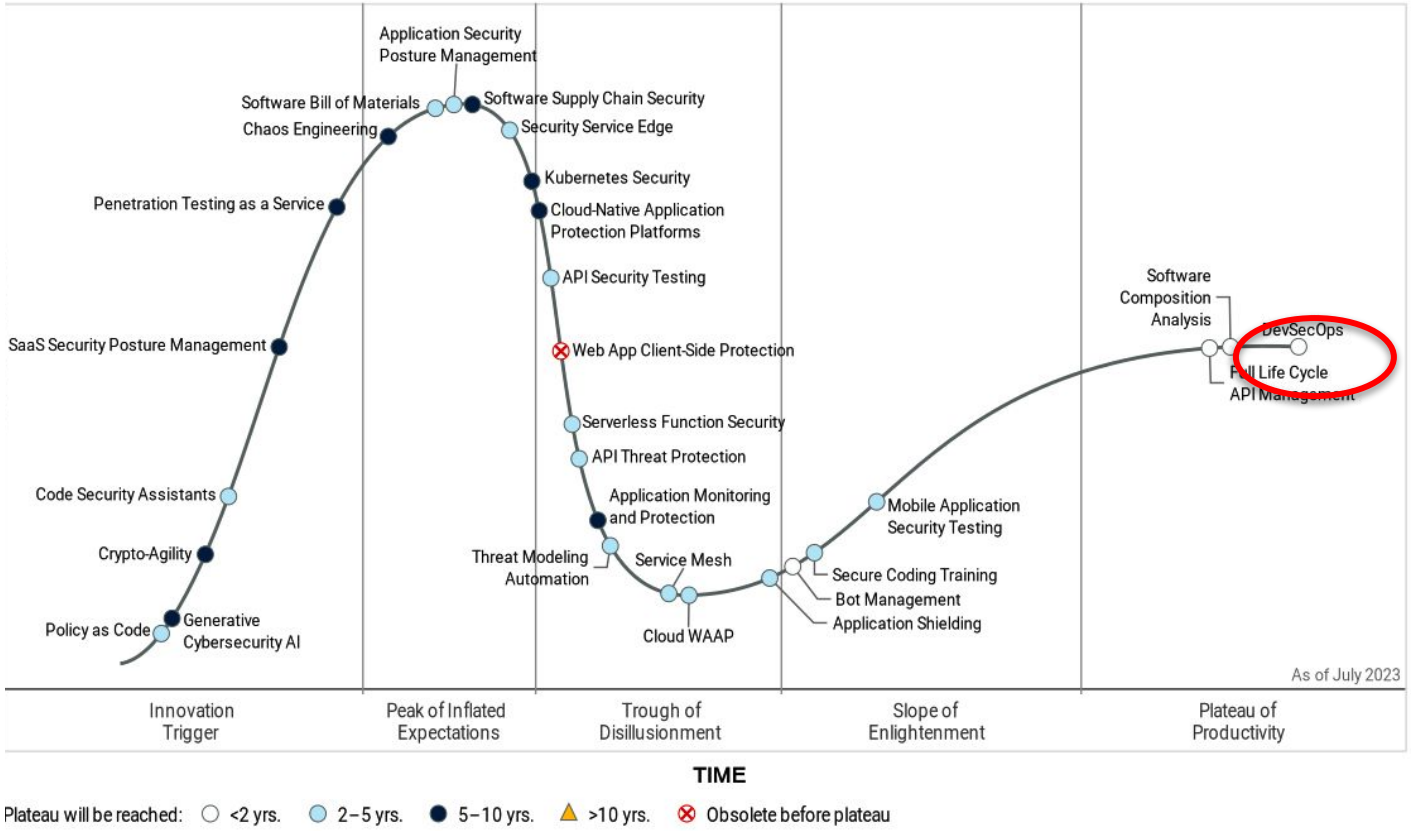
Outline

- DevSecOps
  - **Intro**
  - DevOps Processes
  - Pipeline
  - Some Data
- Site Reliability Engineering
- Best practices
- Standards and Laws

Why?

Performance level	Deployment frequency	Change lead time	Change failure rate	Failed deployment recovery time
Elite	On demand	Less than one day	5%	Less than one hour
High	Between once per day and once per week	Between one day and one week	10%	Less than one day
Medium	Between once per week and once per month	Between one week and one month	15%	Between one day and one week
Low	Between once per week and once per month	Between one week and one month	64%	Between one month and six months

Gartner Hype Cycle



What is DevOps?

- DevOps is a set of practices intended to reduce time between committing a change to a system and the change being placed into normal production, while ensuring high quality
- DevOps practices involve Developers and Operators' processes, architectures, and tools
- Started ~2008

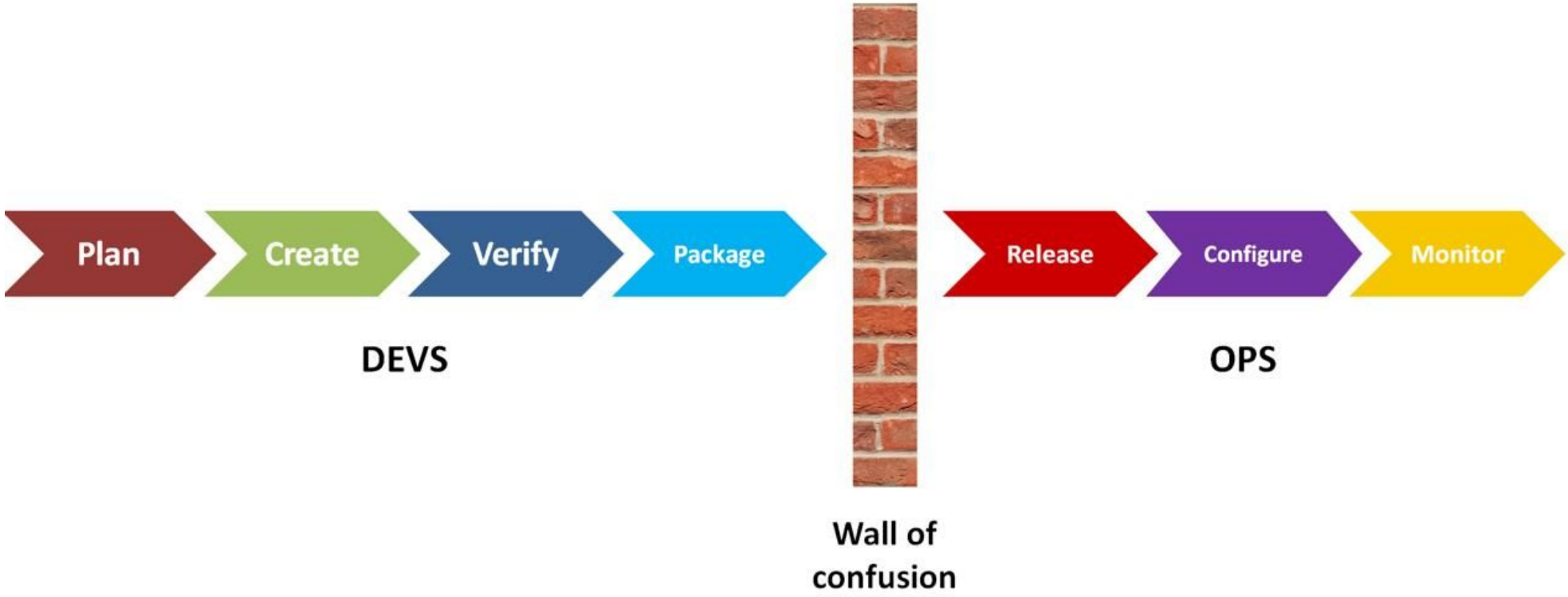
DevSecOps

- DevSecOps is a practice that rose from DevOps
- Includes information technology security as a fundamental aspect in all the stages of development
- Sec already in DevOps but used to emphasise security

Site Reliability Engineering (SRE)

- SRE is "what happens when a software engineer is tasked with what used to be called operations." [Ben Treynor, founder of Google's Site Reliability Team]
- SRE is a discipline that incorporates aspects of software engineering and applies them to infrastructure and operations problems. The main goals are to create scalable and highly reliable distributed applications

The Old Way


Plan
Create
Verify
Package
Release
Configure
Monitor
DEVS
OPS
Wall of
confusion

Code is not the end

- Code Complete & Tested ≠ Code in Production
- Between them: deployment !!!
- Deploying completed can be
  - Time consuming
  - Error prone (Security!)

Operations Tasks

- Integrate code developed by others
- Executable can be constructed?
- Test built system for correctness
- Test built system for performance and other qualities
- The built system is placed into production + test + mainantinance

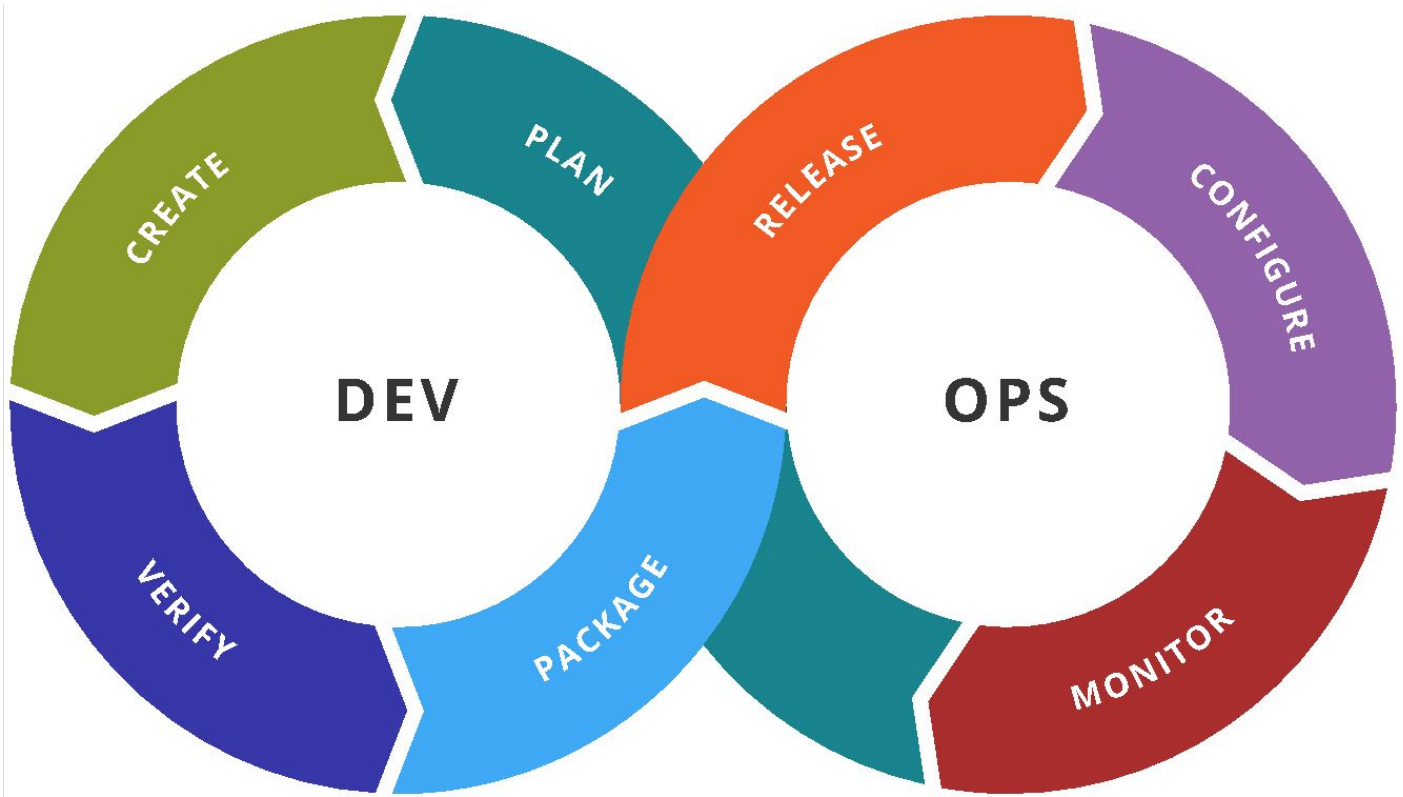
Time is passing

- Every error must either be corrected or prevented
  - Coordination takes time
  - Correcting errors takes time
- Preventing errors through combination of
  - Tooling & Processes
  - Coordination among teams

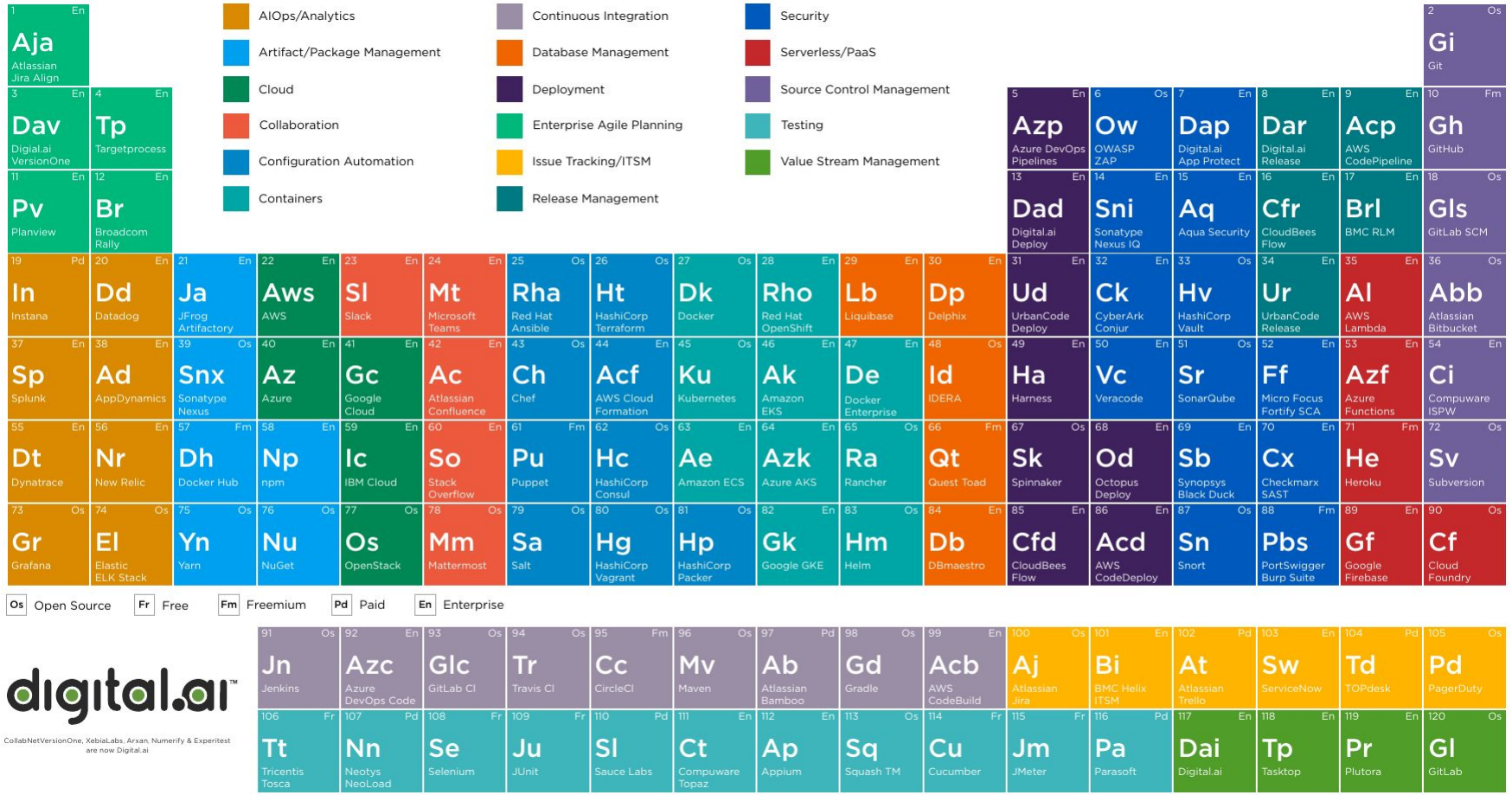
Velocity is important

- Internet companies deploy multiple times a day
- Amazon had a new release to production every 11.6 seconds in May of 2011
- 2015 → 50 million deployments annually (more than one a second!)
- Velocity → Fast in releasing features → Stay ahead of competition

New Solution



A large number of tools



Outline

- DevSecOps
  - Intro
  - **DevOps Processes**
  - Pipeline
  - Some Data
- Site Reliability Engineering
- Best practices
- Standards and Laws

DevOps Processes

- Treat Operators as first class citizens
- Make Dev more responsible for incident handling
- Enforce deployment practices uniformly across both dev and ops
- Use Continuous Deployment
- Infrastructure as a code

1 - Operators as first class citizens

- One problem today: developers do not necessarily have to care about good logs and error messages
- Operators spend long time troubleshooting
- Treating operators as a stakeholder → gathering input for requirements solving this problem

2 - Dev & Incident Handling

- Traditional incident handling sequence is
  - Operator informed of a problem
  - Operator decides whether it is an operational problem (their responsibility) or a bug (developers responsibility).
  - If it is a developers responsibility, the incident is escalated to the developers

2 - Dev & Incident Handling

- Problem
  - Time has passed
  - Developers have lost context for potential solutions
  - Developers do not see problems directly and may be insensitive to how frequently they cause problems
- Solution: Making Dev first point of contact for incident handling

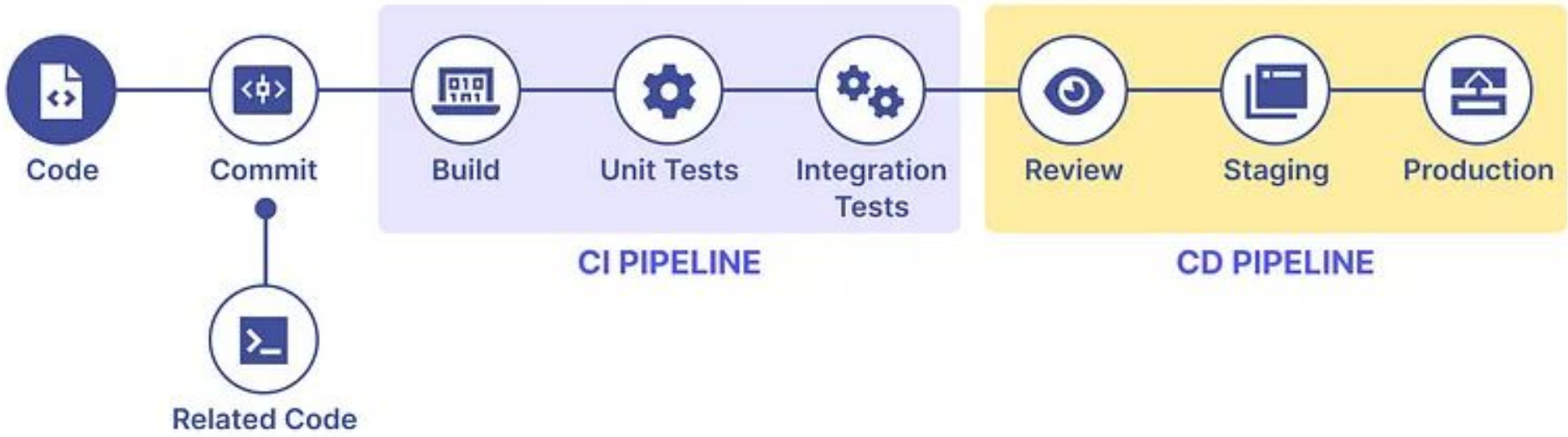
3 - Uniform depl practices

- Currently: Ops may have different deployment practices
- E.g., login as superuser to directly apply security patches without going through version control, automated testing, ...
- Goal: every executing image be traceable in terms of its constituent parts and its version history

4 - Continuous Deployment

Continuous deployment is a process and related practices that allow developers to place code into production without human intervention

Delivery Mechanism - Pipeline



5 - Infrastructure as a code

- Developers have a number of processes to ensure quality
  - Test driven development
  - Automated tests
  - Version control
- Operators, historically, have not used these processes
- Goal: reduce error rate for infrastructure code.

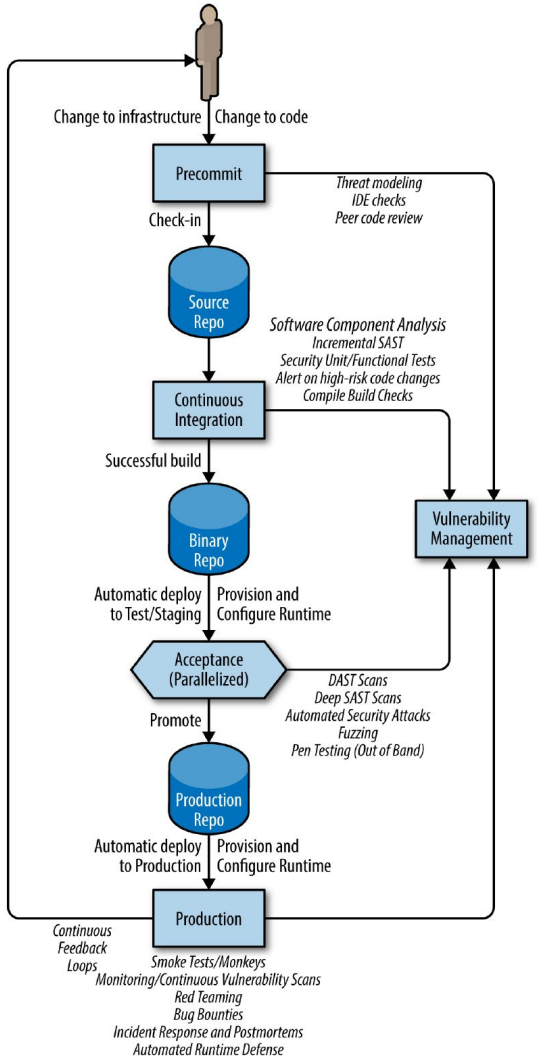
Outline

- DevSecOps
  - Intro
  - DevOps Processes
  - **Pipeline**
  - Some Data
- Site Reliability Engineering
- Best practices
- Standards and Laws

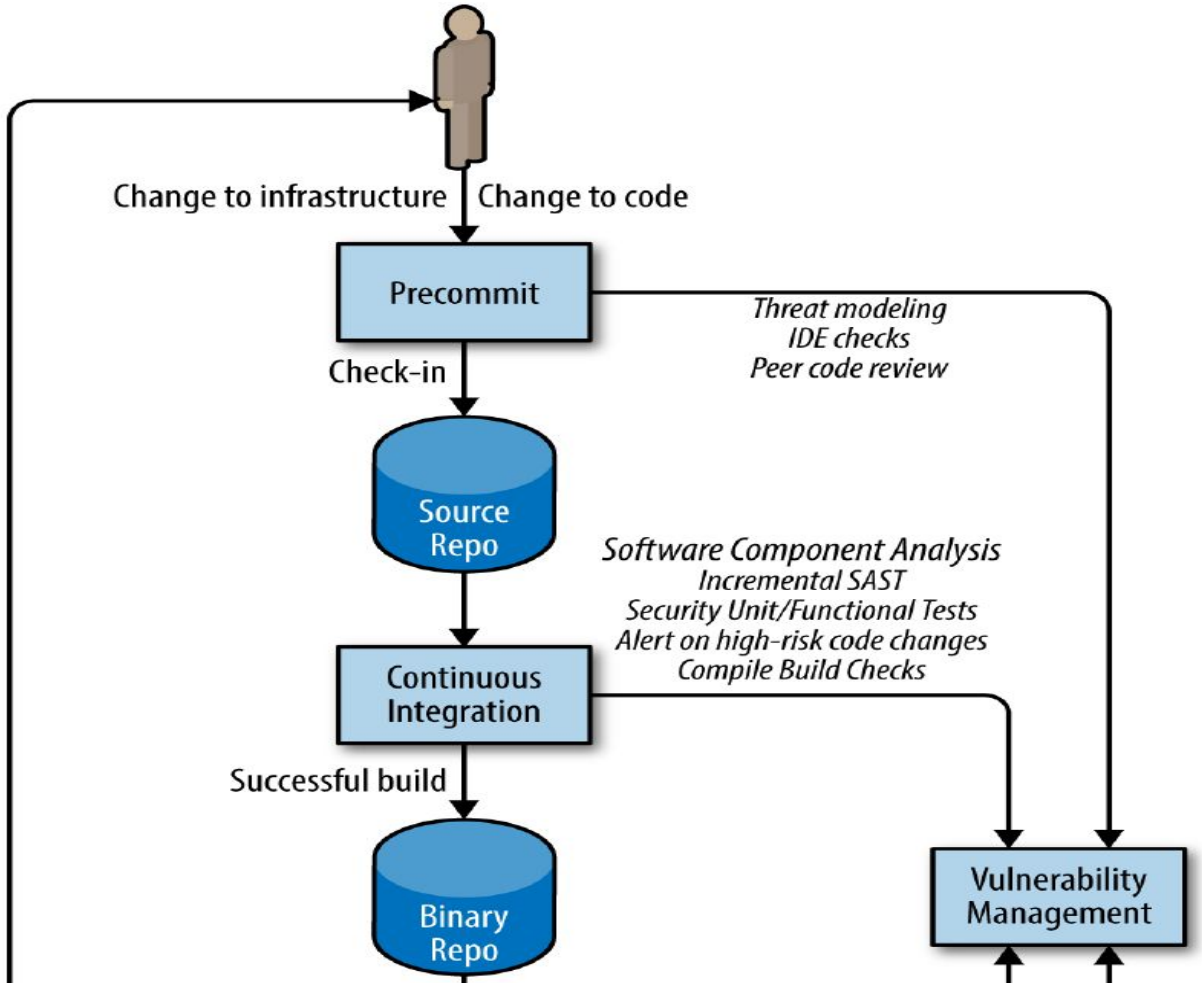
Delivery Mechanism - Pipeline

- Failures in the pipeline will add time to deployment
- Increase surface of attack
- More important than the app itself
  - if no pipeline ⟶ no app

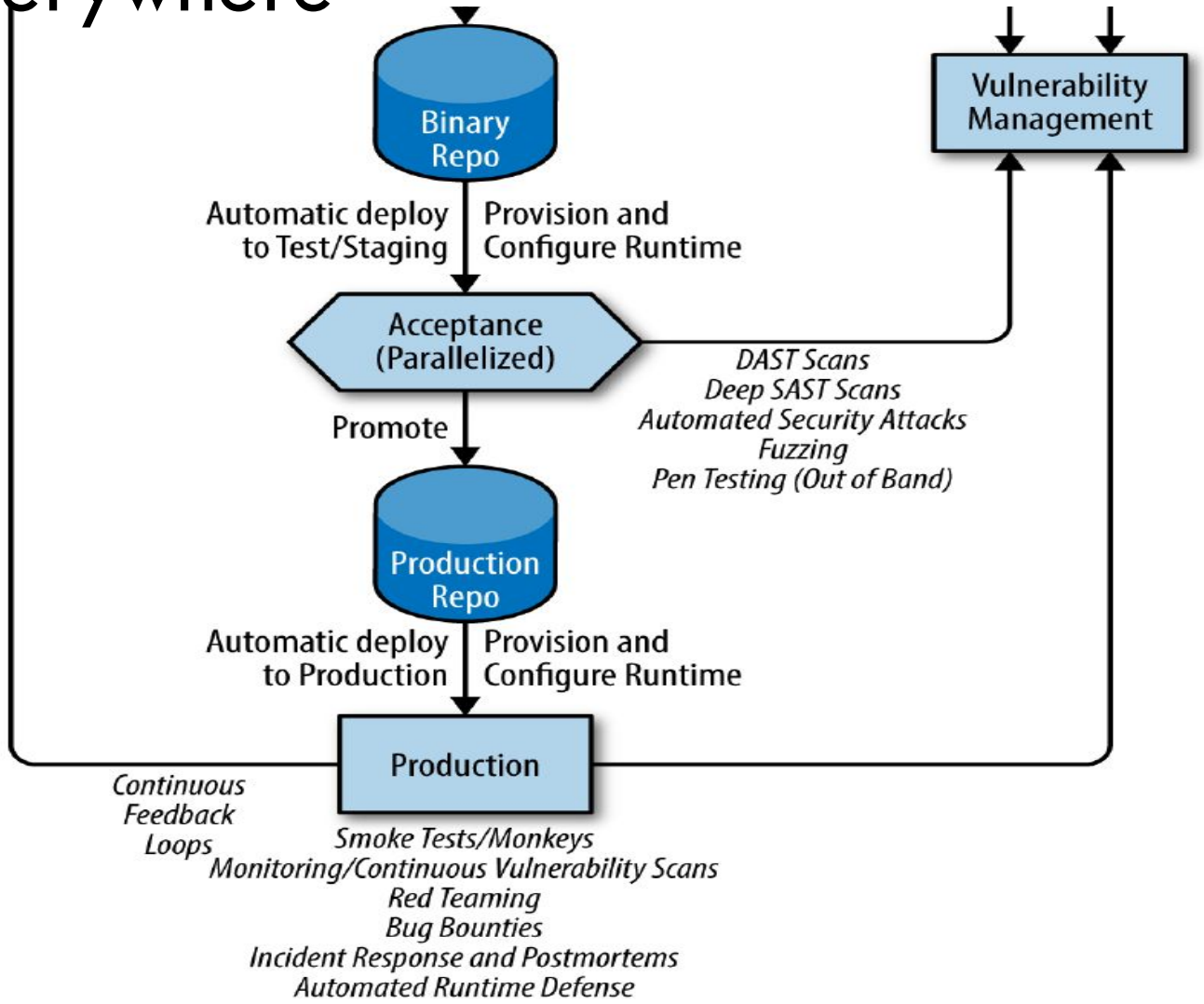
Test are everywhere



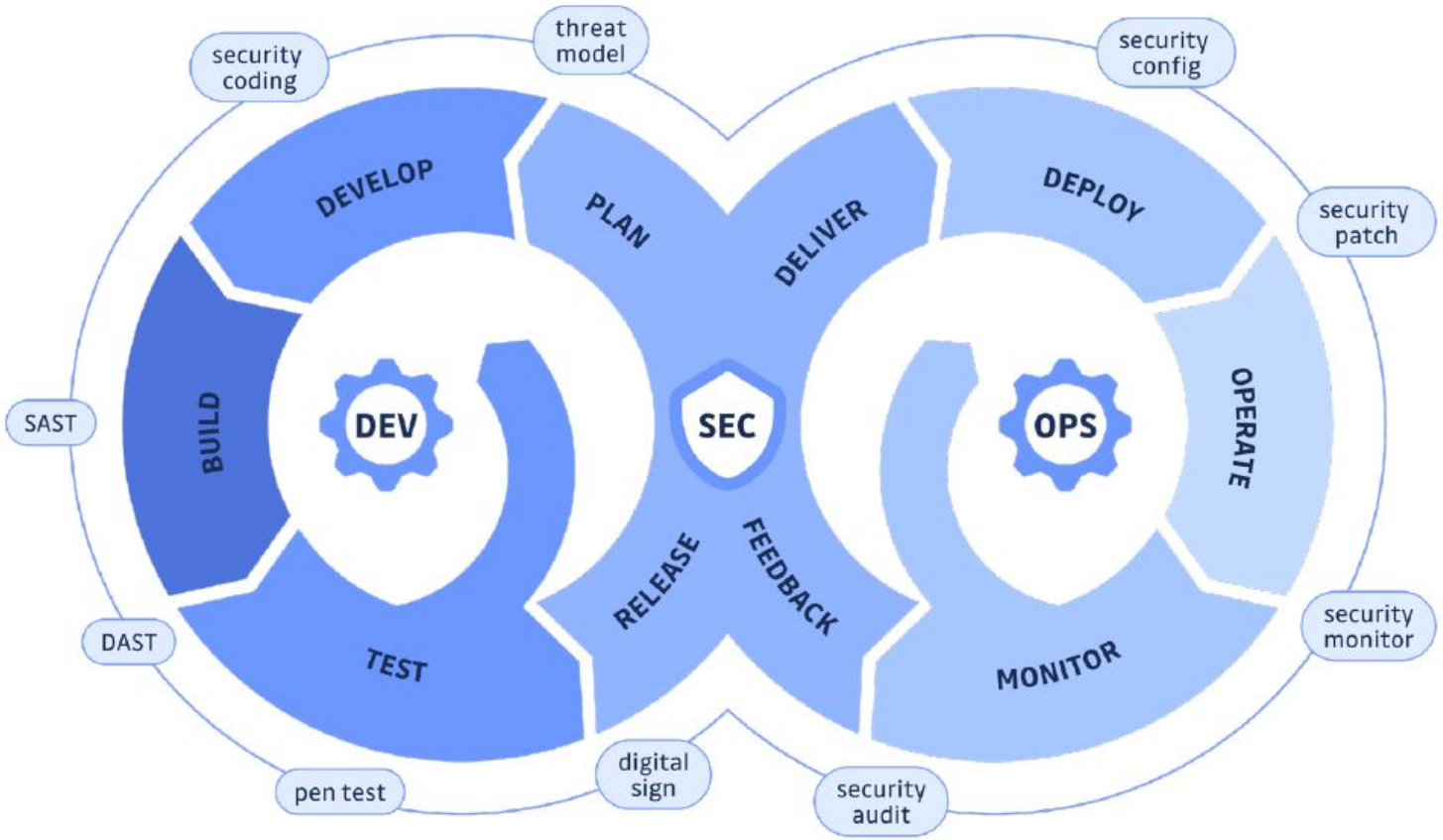
Test are everywhere



Test are everywhere



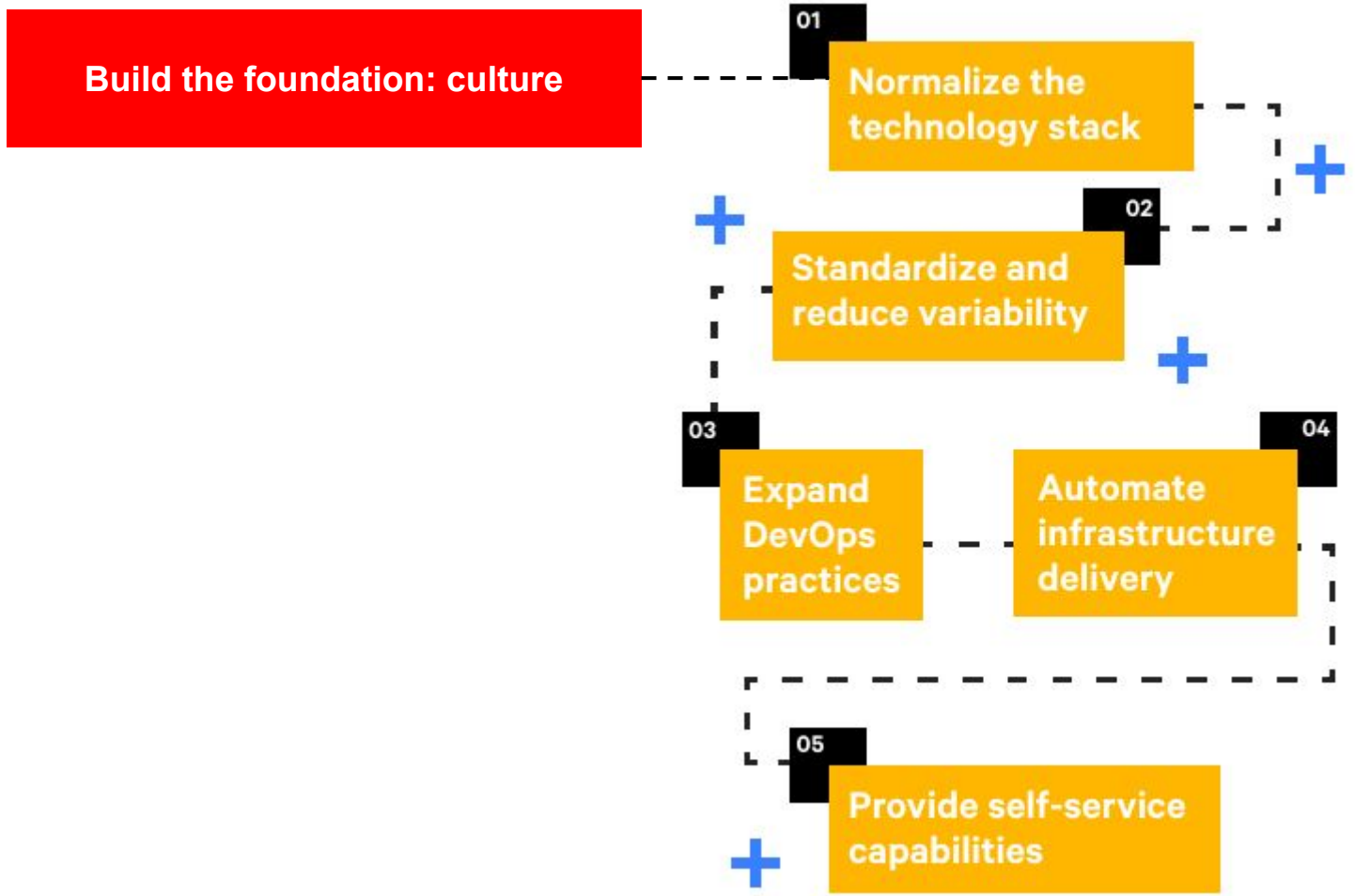
Security is everywhere



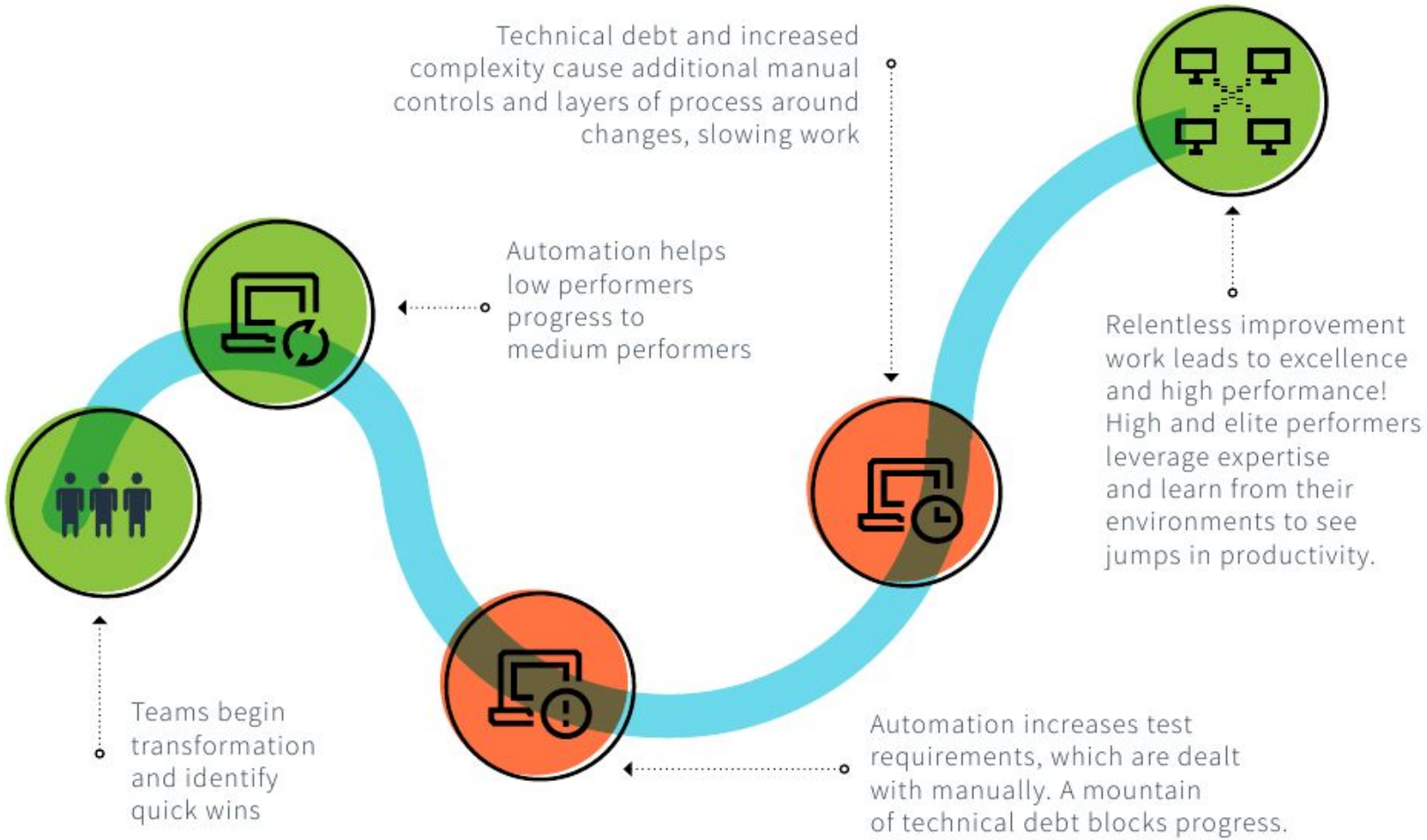
Culture

- Foster a learning mindset
- Look for continuous improvement
- Establishing psychological safety to enable truth telling
  - Blameless Postmortem reviews
    - Removing blame from a postmortem gives people the confidence to escalate issues
    - Important not to stigmatize frequent production of postmortems by a person or team
    - Atmosphere of blame risks creating a culture in which incidents and issues are swept under the rug → greater risk for the organization

Five Evolution Stages

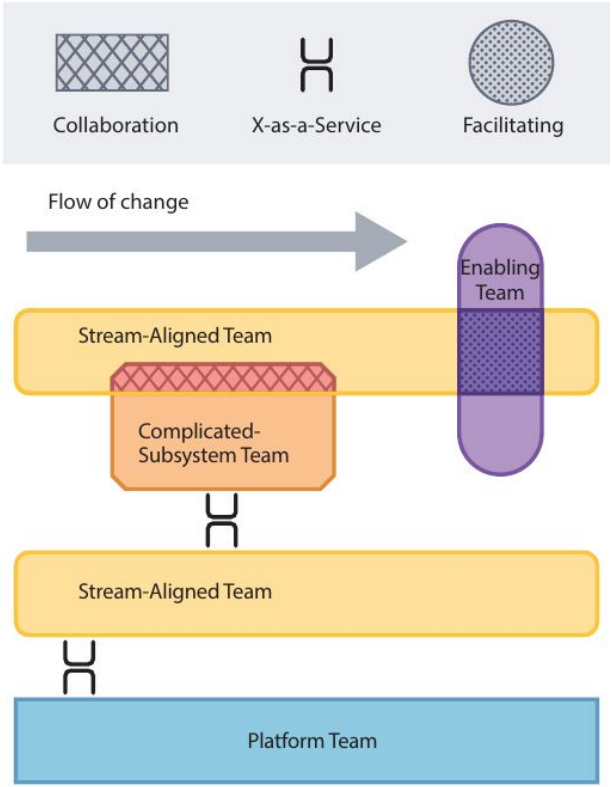


J-Curve of Transformation



Team topologies

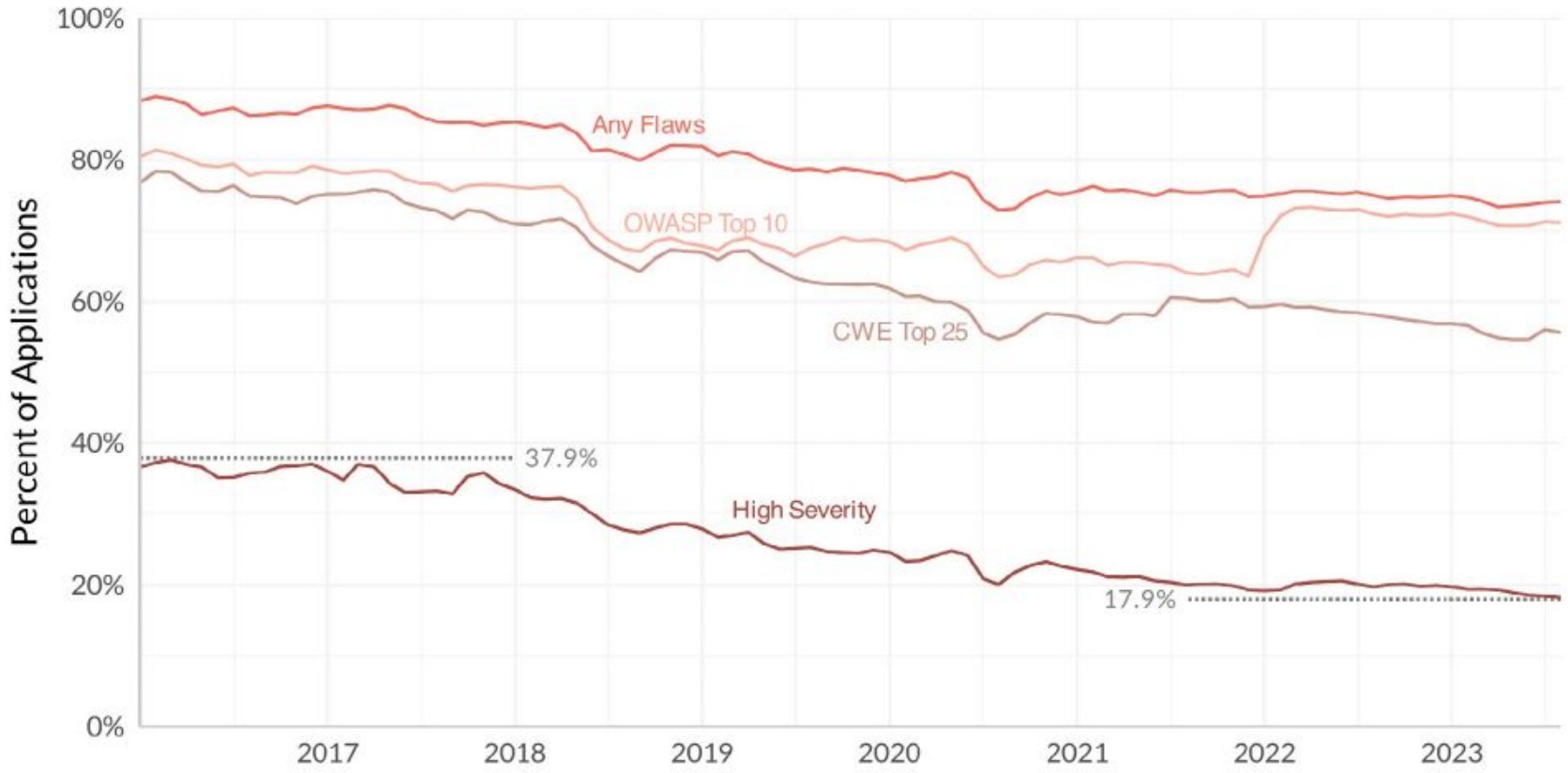
- Stream-aligned
  - aligned to a flow of work
- Enabling
  - helps a stream-aligned team to overcome obstacles + missing capabilities
- Complicated subsystem
  - where mathematics/calculation or hard-to-find niche technical expertise is needed
- Platform
  - provide a internal product to accelerate delivery by stream-aligned teams.



Outline

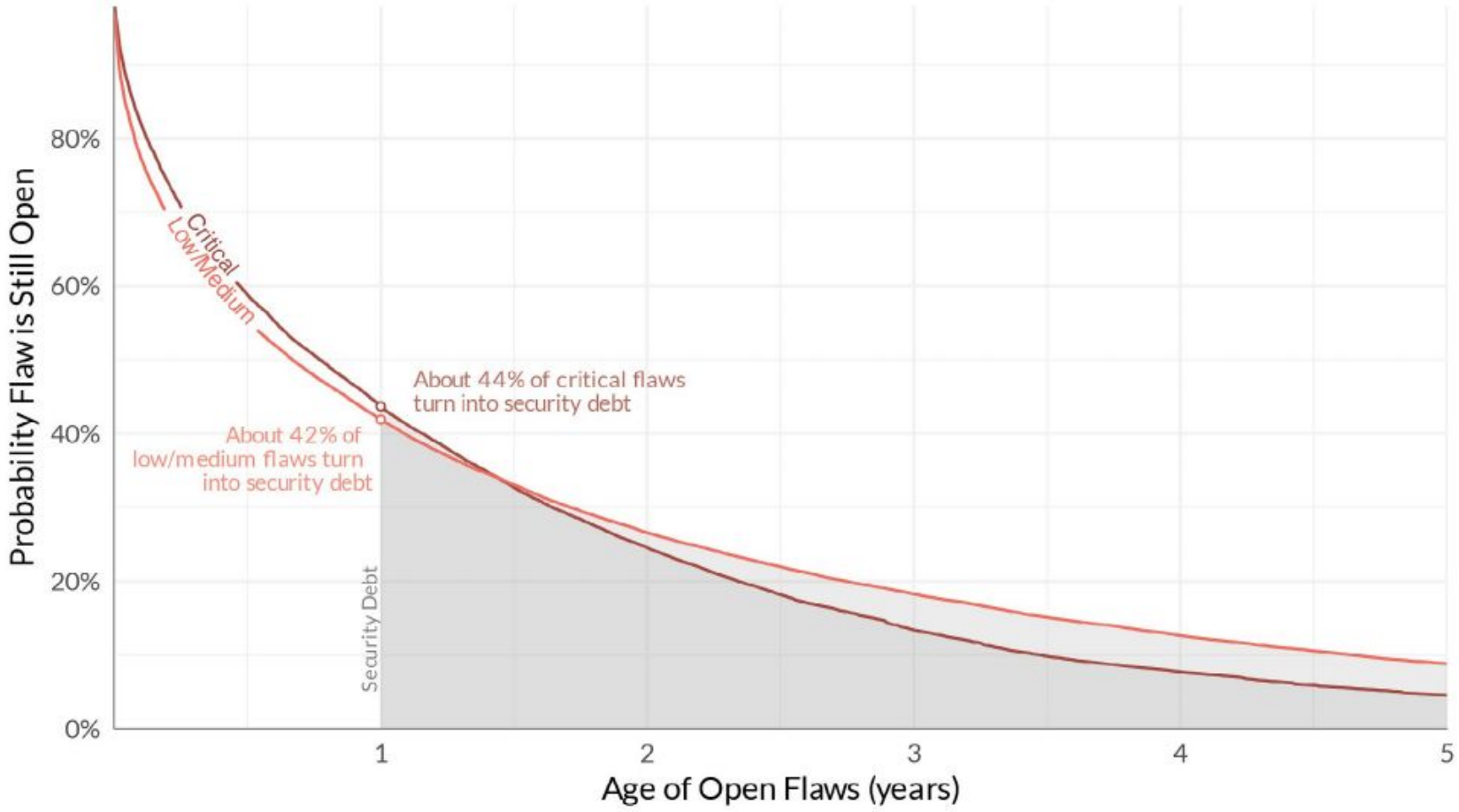
- DevSecOps
  - Intro
  - DevOps Processes
  - Pipeline
  - **Some Data**
- Site Reliability Engineering
- Best practices
- Standards and Laws

Security Status



State of Software Security 2024 - Veracode

Days to close a flaw



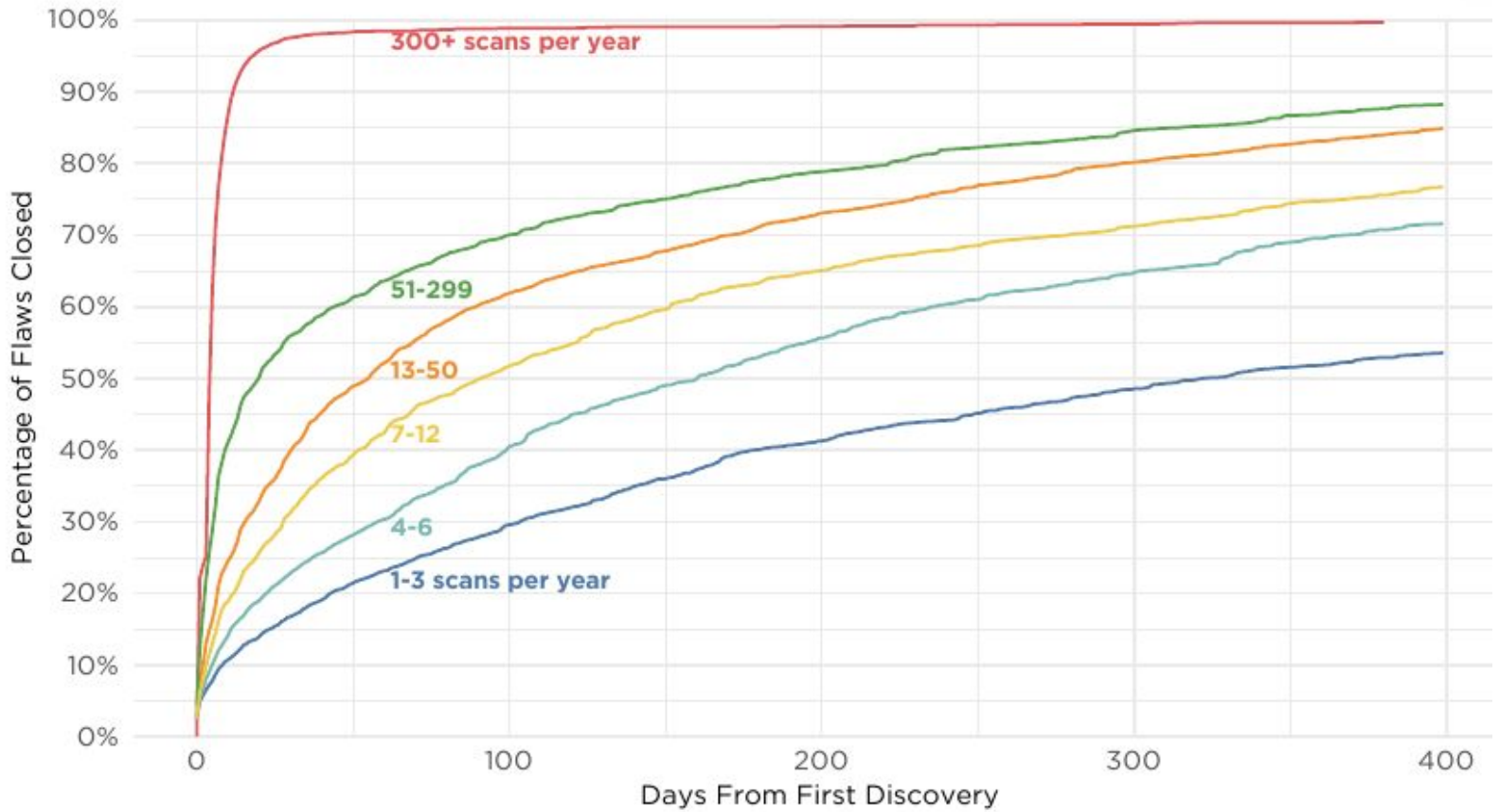
State of Software Security 2024 - Veracode

Automation Level

	Low	Medium	High	Elite
Automated build	64%	81%	91%	92%
Automated unit tests	57%	66%	84%	87%
Automated acceptance tests	28%	38%	48%	58%
Automated performance tests	18%	23%	18%	28%
Automated security tests	15%	28%	25%	31%
Automated provisioning and deployment to testing environments	39%	54%	68%	72%
Automated deployment to production	17%	38%	60%	69%
Integration with chatbots / Slack	29%	33%	24%	69%
Integration with production monitoring and observability tools	13%	23%	41%	57%
None of the above	9%	14%	5%	4%

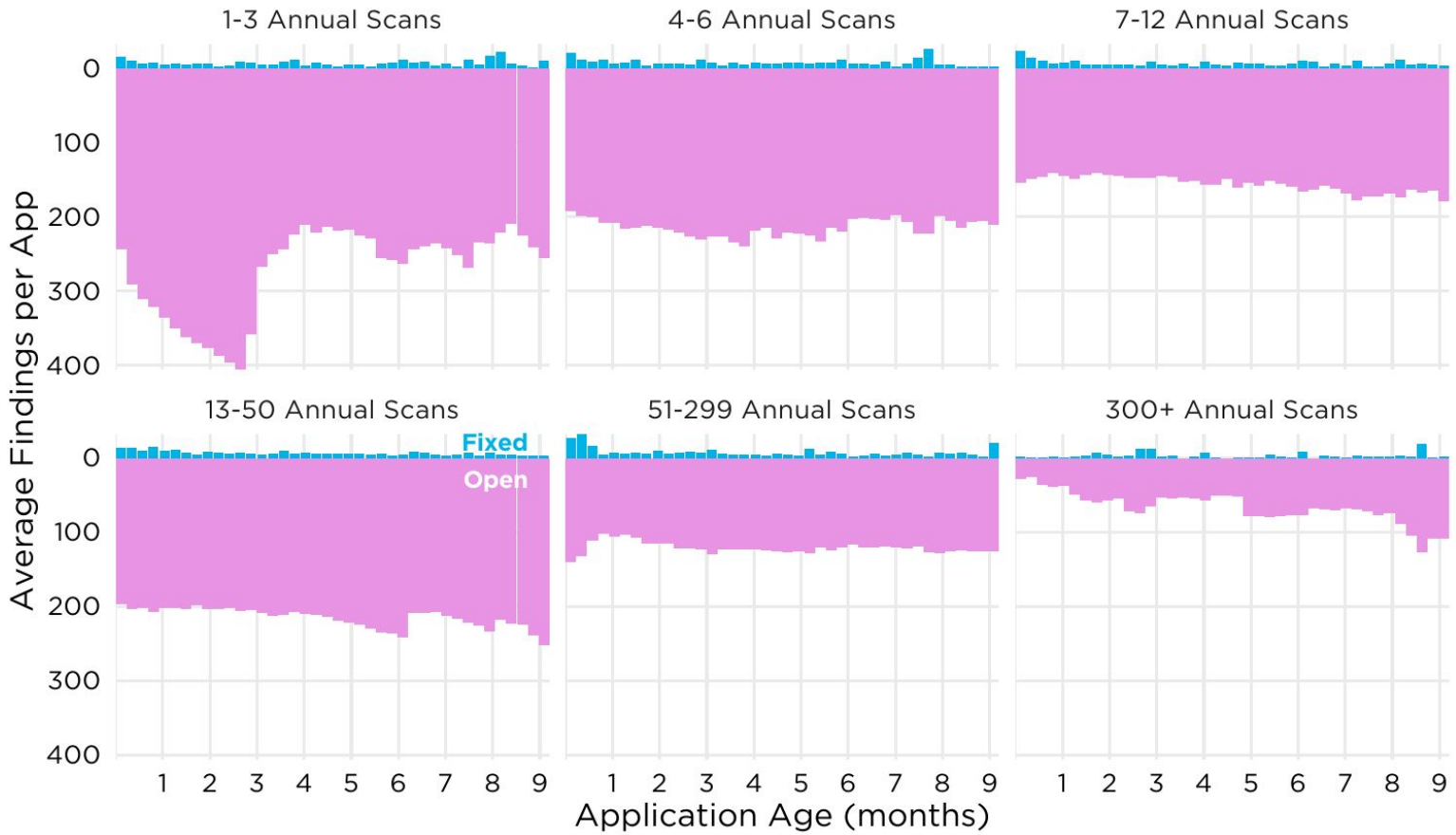
State of DevOps - DORA

Fix Velocity & Scan



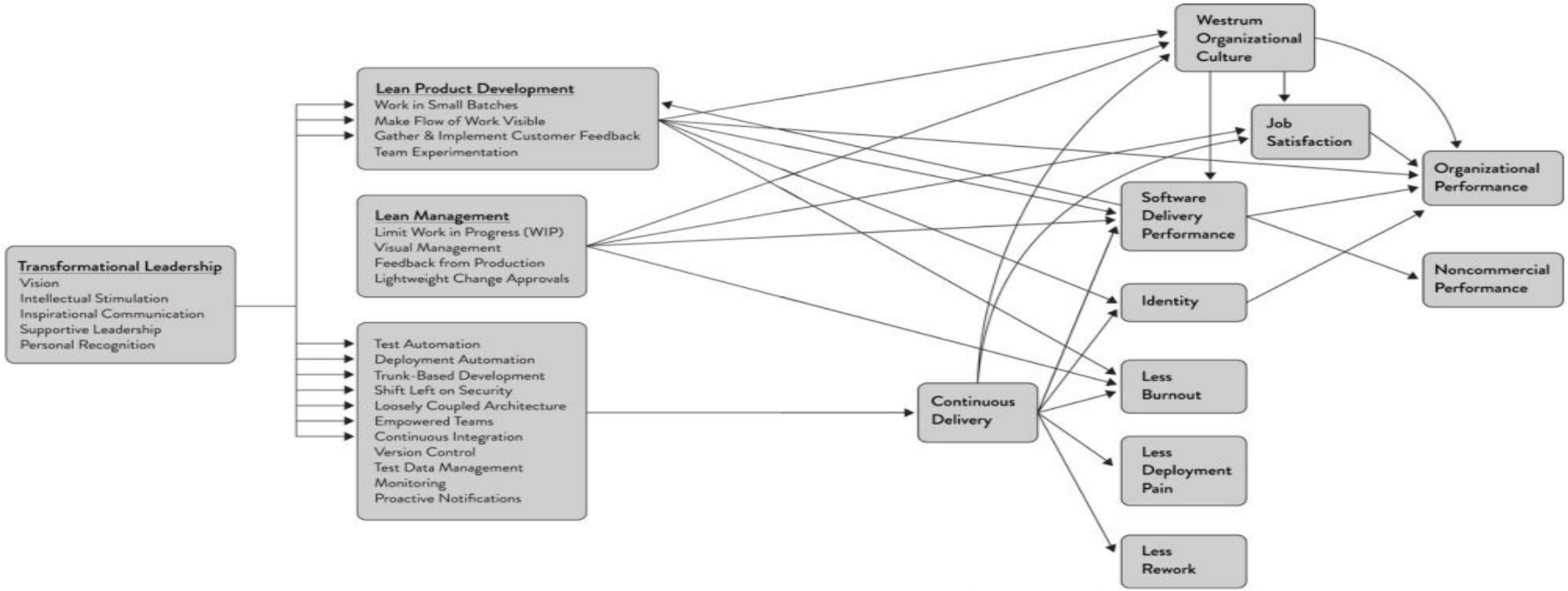
State of Software Security - Veracode

Time to remediation & Scan



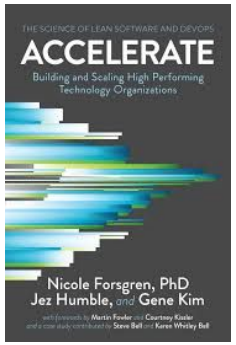
State of Software Security - Veracode

DORA Findings



Accelerate: Building and Scaling High Performance Technology Organizations

DORA Findings



- 4 State of DevOps performance unlocks competitive advantages (profitability, productivity, market share, customer satisfaction, ...)
- The cloud improves software delivery performance
  - Teams that leverage on cloud → 23 times more likely to be high performers
- Open source software improves performance
  - 1.75 times more likely to be extensively used by highest performers
- Outsourcing by function is rarely adopted by elite performers and hurts performance.

DORA Findings

- Key technical practices drive high performance:
  - monitoring and observability,
  - continuous testing
  - database change management
  - integrating security earlier (Shift Left)
- High performers in both non-regulated and highly regulated industries alike.

Outline

- DevSecOps
- **Site Reliability Engineering**
- Best practices
- Standards and Laws

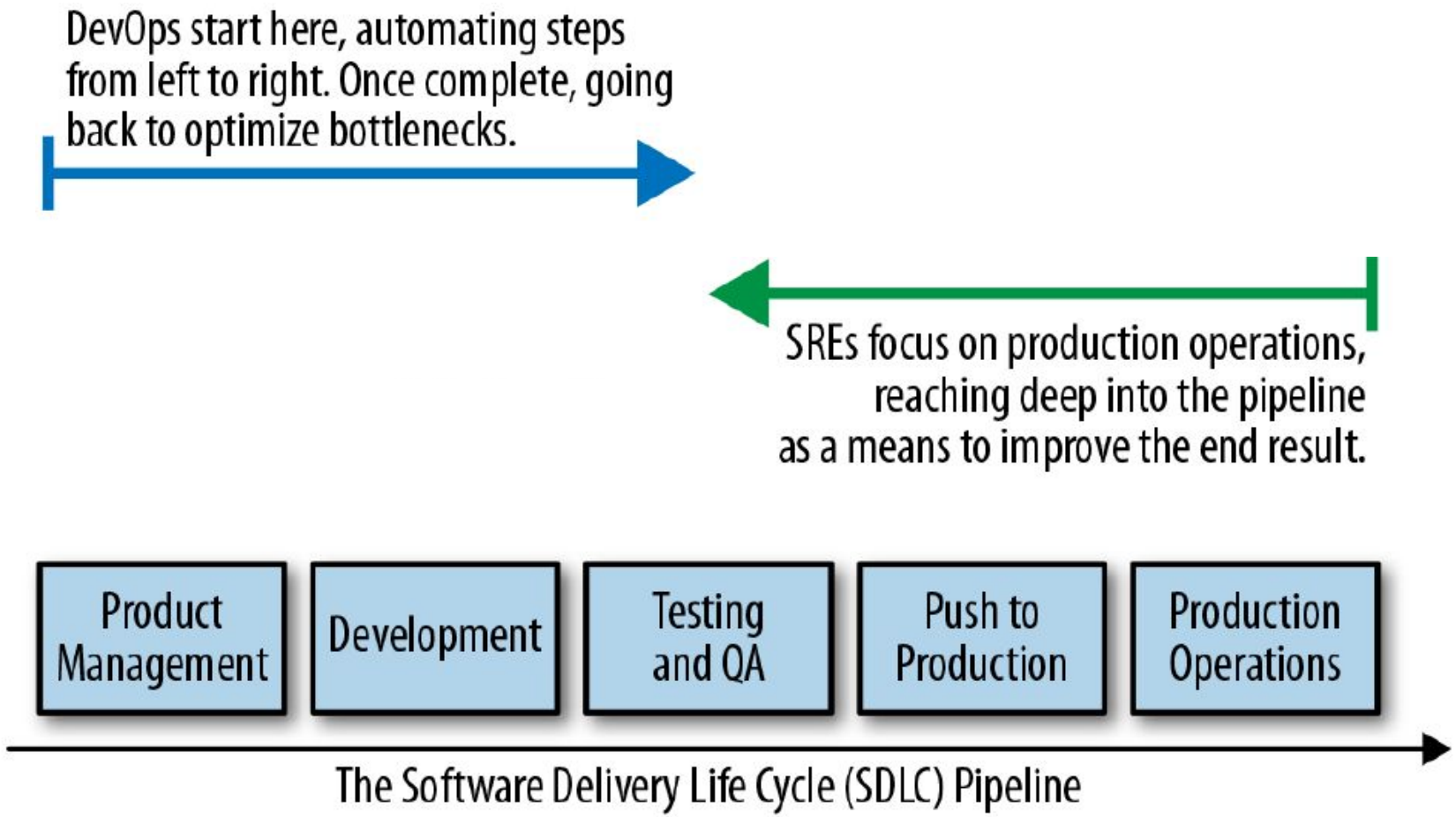
DevOps → SRE

- SRE is viewed as a "specific implementation of DevOps with some idiosyncratic extensions" [Google]
- The discipline of security engineering has gradually made its way into DevOps with DevSecOps, rugged DevOps, SRE and other name variants. Things are changing and new systems are designed and build with security in mind [Aaron Rinehart - Chaos engineer]

Toil

"If a human operator needs to touch your system during normal operations, you have a bug. The definition of normal changes as your systems grow" [Carla Geisser, Google SRE]

Site Reliability vs DevOps



Site Reliability Engineering (SRE)

SRE & DevOps

- SRE shares ownership with developers to create shared responsibility
- Use the same tools that developers use
- Accept failure as normal & embrace risk
- Blameless post mortems, gradual changes, tooling and automation
- Measure everything

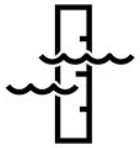
Metrics

- Metrics needed for compliance and risk-management
  - Understand where you need to prioritize your testing and training efforts
  - Assess your application security program
- Metrics should answer
  - How many vulnerabilities have you found?
  - How were they found?
  - What tools or testing approaches are giving you the best returns?
  - What are the most serious vulnerabilities?
  - How long are they taking to get fixed?
  - Is this getting better or worse over time?

SRE Main Concepts

- Quantifies failure and availability in a prescriptive manner
  - **Service Level Indicators** (SLIs)
    - request latency, error rate, system throughput, availability
  - **Service Level Objectives** (SLOs) (lower bound ≤ SLI ≤ upper bound)
    - **Error Budget** Policy + **Policy** on what to do when service runs out of budget
- Reduce **toil** (charter to automate menial tasks)

Service Level Indicators



- Recommend treating SLI as the ratio of two numbers (good vs total)
- Examples:
  - Number of successful HTTP requests / total HTTP requests (success rate)
  - The fraction of time a service is usable
  - Number of search results / total number of search results

Service Level Objectives

- SLOs help determine what engineering work to prioritize
  - E.g., automating rollbacks or moving to a replicated data store? Which one should you choose?
- Things needed to use SLO
  - Stakeholders have approved that some SLOs are fit for the product
  - The people responsible for ensuring that the service meets its SLO have agreed that it is possible to meet this SLO under normal circumstances
  - The organization has committed to using the error budget for decision making and prioritizing (error budget policy)
  - There is a process in place for refining the SLO

SLO Examples

- Over four weeks, the API metrics show
  - Total requests: 3,663,253
  - Total successful requests: 3,557,865 (97.123%)
  - 90th percentile latency: 432 ms
  - 99th percentile latency: 891 ms

SLO type	Objective
Availability	97%
Latency	90% of requests < 450 ms
Latency	99% of requests < 900 ms

Error Budget

SLO	Allowed failures
97% availability	109,897
90% of requests faster than 450 ms	366,325
99% of requests faster than 900 ms	36,632

SLO: VALET

- Volume (traffic): How much business volume can my service handle?
- Availability: Is the service up when I need it?
- Latency: Does the service respond fast when I use it?
- Errors: Does the service throw an error when I use it?
- Tickets: Does the service require manual intervention?

Service Level Objectives

- 100% reliability is the wrong target
  - From 99% to 99.9% to 99.99% reliability, each extra nine comes at an increased cost, but marginal utility approaches zero
  - It means that you only have time to be reactive
  - 4 nines of availability in a given quarter (99.99%) means
    - allowed quarterly downtime is around 13 minutes
    - reaction time of on-call engineers has to be in the order of minutes!!!
- First attempt at an SLI and SLO doesn't have to be correct → the most important goal is to get something in place

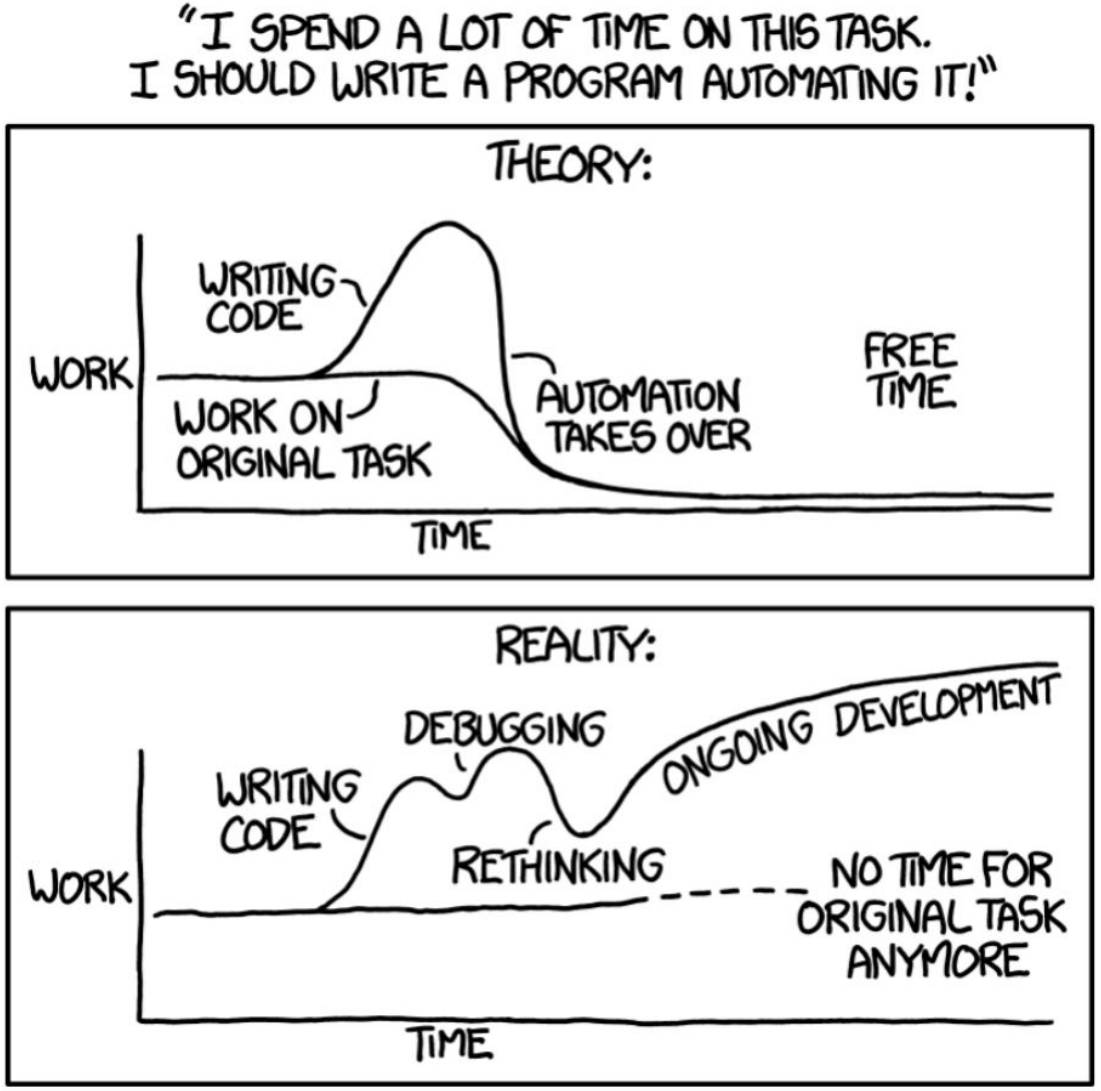
Time Windows

- Rolling windows → aligned with user experience
  - E.g., 30-day window
- Calendar windows are more closely aligned with business planning and project work
- Shorter time windows → decisions more quickly
- Longer time periods → better for more strategic decisions

Toil

- Toil is work tied to running a production service that tends to be manual, repetitive, automatable, reactive, lacks enduring value, and that scales linearly as a service grows
- Goal: keep toil (+ overhead such as admin work, peer reviews, meetings, …) below 50%
- Toil bad for: career stagnation, low morale, slows progress, promotes attrition

Toil


"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"
THEORY:
WORK
WRITING CODE
WORK ON ORIGINAL TASK
AUTOMATION TAKES OVER
FREE TIME
TIME
REALITY:
WORK
WRITING CODE
DEBUGGING
RETHINKING
ONGOING DEVELOPMENT
NO TIME FOR ORIGINAL TASK ANYMORE
TIME

Outline

- DevSecOps
- Site Reliability Engineering
- **Best practices**
- Standards and Laws

Situational Awareness and Attack-Driven Defense

- Involving entire team (not only Security team)
  - Metrics available in the context of the running system
  - Graphing
- Recognize that your system is, or will be, under constant attack
- Feed data back into testing and reviews
  - Prioritize based on what you are see in production
- Watch for runtime errors, exceptions, and attack signatures
  - Shows where you are being probed and tested
  - What kind of attacks you are seeing
  - Where they are attacking
  - Where they are being successful
  - What parts of the code need to be protected more

Code reviews

- Useful for:
  - Accountability
  - Internal threads
  - Guidelines
  - Better code
  - …
- Best practice (e.g., Etsy company)
  - as soon as high-risk code is identified through reviews or scanning → hash it and create a unit test that automatically alerts the security team when the code hash value has been changed
- Pair programming miss important bugs, including security vulnerabilities
  - pair programming is more about joint problem solving, navigating
- Do separate security-focused code reviews for high-risk code

Securing the Infrastructure

- In CD you also need to protect the pipeline
  - From insider attacks
    - Ensuring that all changes are fully transparent and traceable from end to end
    - Malicious and informed insider cannot make a change
  - Periodically review logs
    - Ensure that they are complete
    - Ensure you can trace a change through from start to finish
    - Ensure that the logs are immutable (cannot be erased or forged)

Security in Production

- Randomly injecting failures into the production runtime
  - Runtime Checks, Chaos Monkey, Chaos Gorilla, and Chaos Kong
  - Check that the system is set up and designed correctly to handle failures
- Captures details about changes to policies over time
  - Used as analysis and reporting tool
  - Forensics purposes, letting you search for changes across time
  - Security Monkey, Amazon's AWS Inspector

Penetration Testing and Bug Bounties

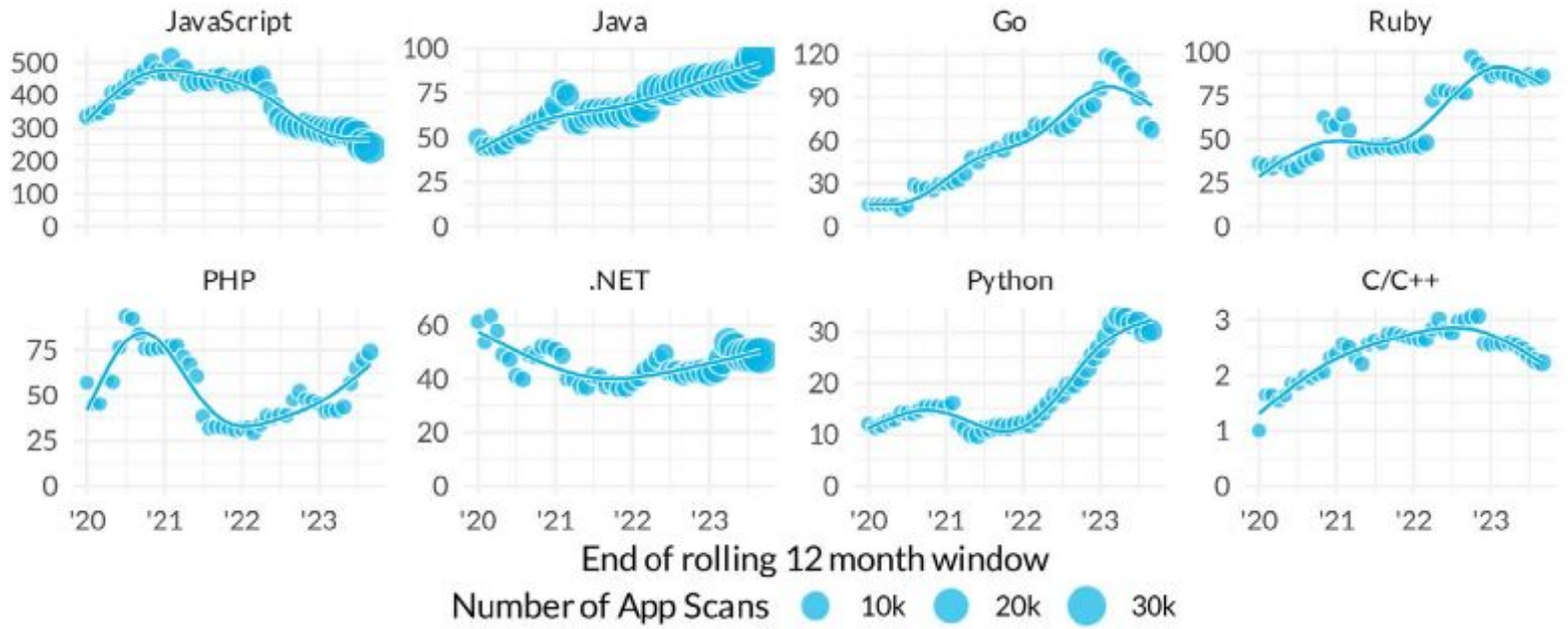
- Manual penetration testing not effective in CD
  - the velocity of delivery is too fast
  - penetration tests take too long to set up, run, and review
- The real value in these tests is not in the bugs but
  - In the information that the bugs provide you if you look deep enough
  - Where did the bug come from?
  - Why did you miss finding it your workflow

Securing supply chain

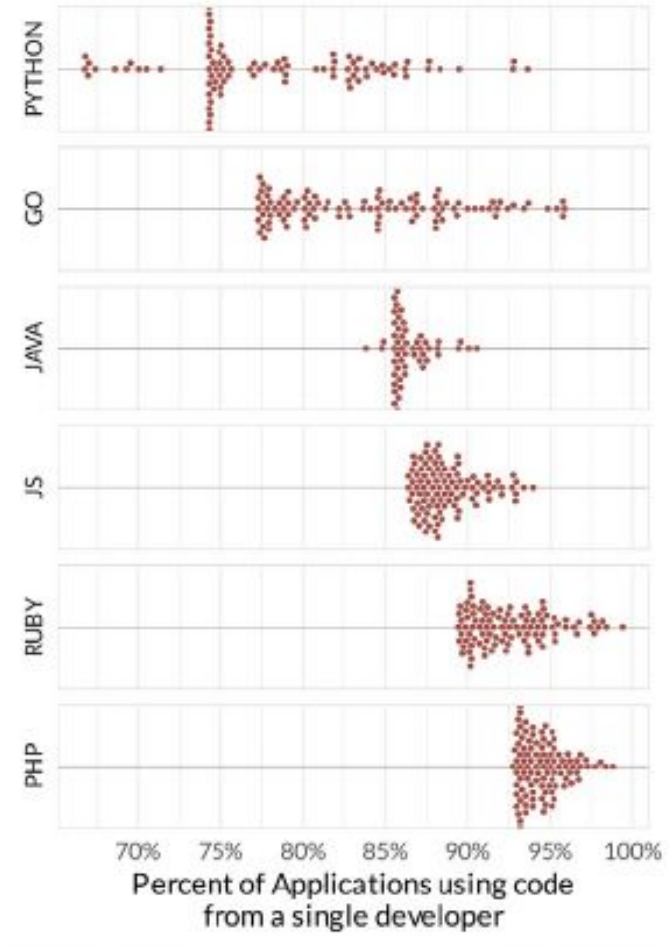
- 80 percent of the code in today's applications comes from libraries and frameworks [Sonatype → runs world's largest repository for open source software]
- 50 new critical vulnerabilities in open source software are reported every day.
- Check and secure the software you use:
  - Software Component Analysis (SCA) tools
    - OWASP's Dependency Check project
    - Sonatype's Nexus Lifecycle
    - SourceClear
    - …

Libraries per application

“In many respects, development teams have shifted from writing software to assembling software.”
CHRIS WYSOPAL, CTO VERACODE

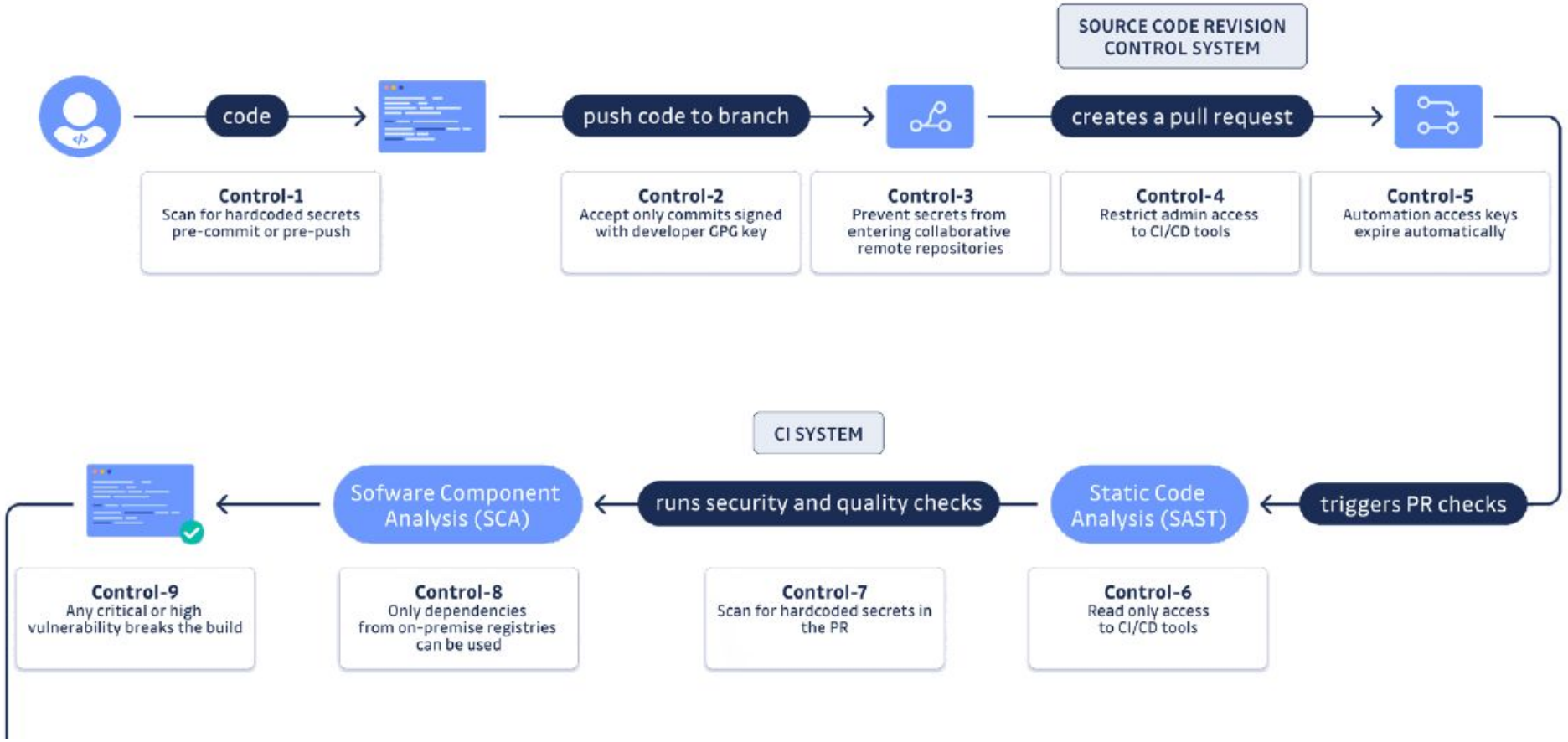


One developer library code is everywhere

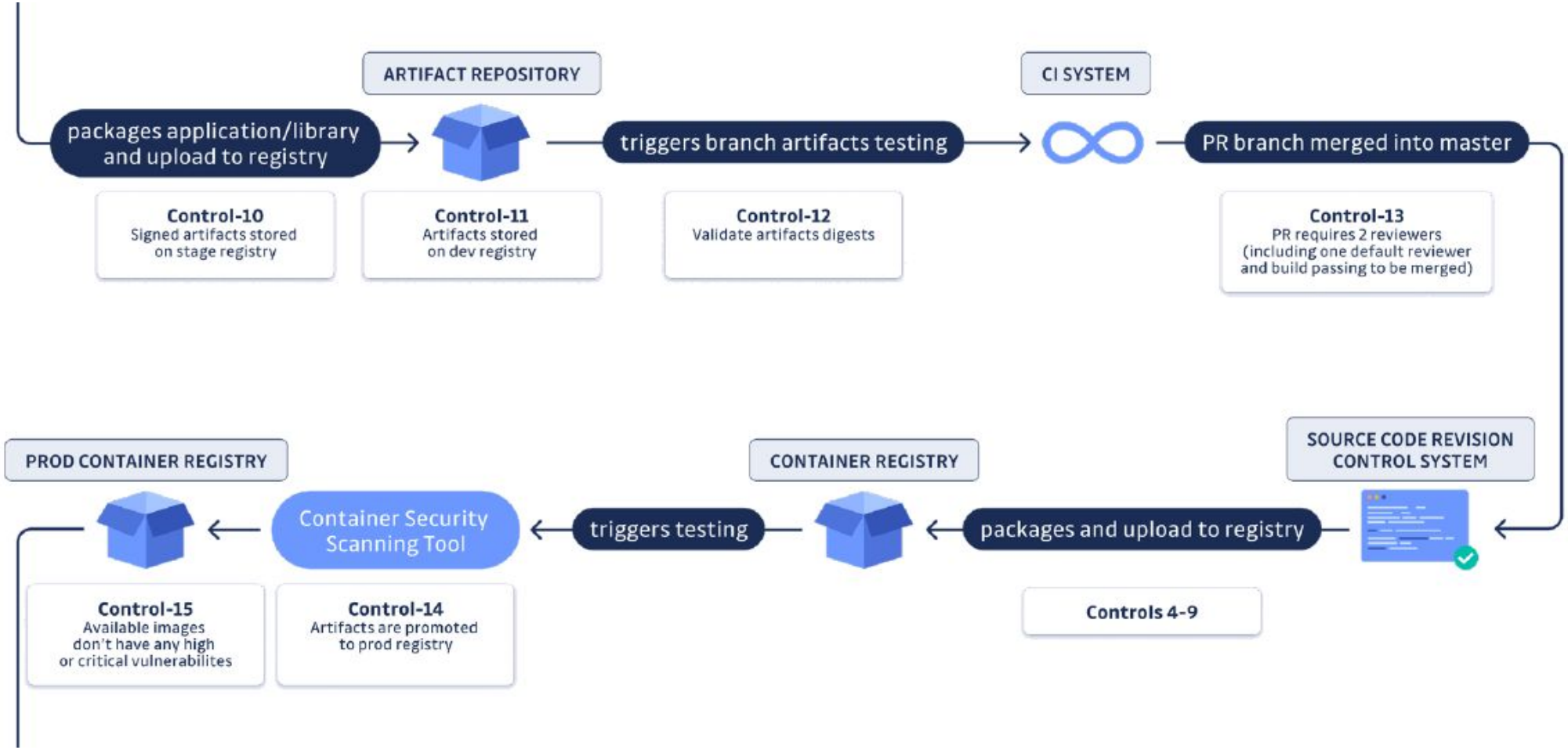


All languages have multiple open-source developers that contribute code that is included in 90% of applications. These represent potential weak links in the software supply chain

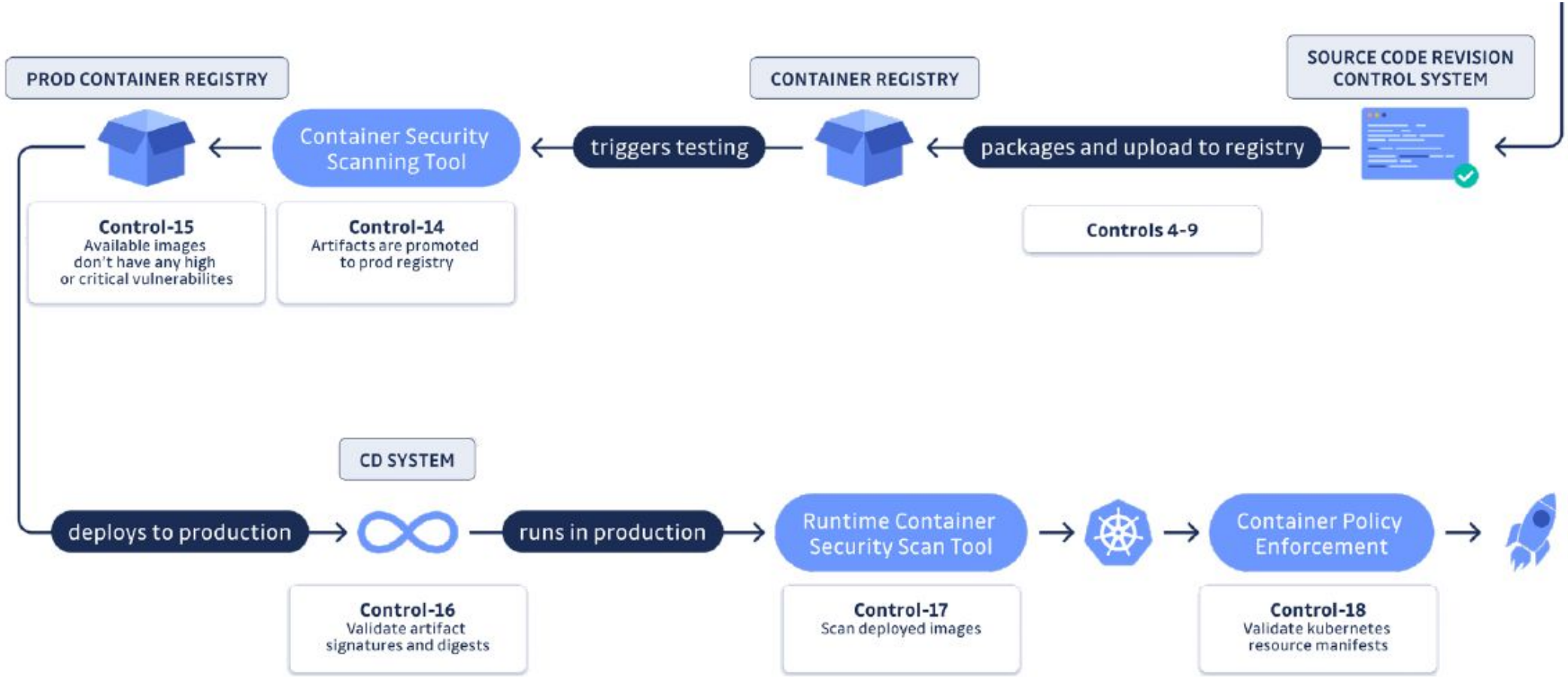
Securing supply chain



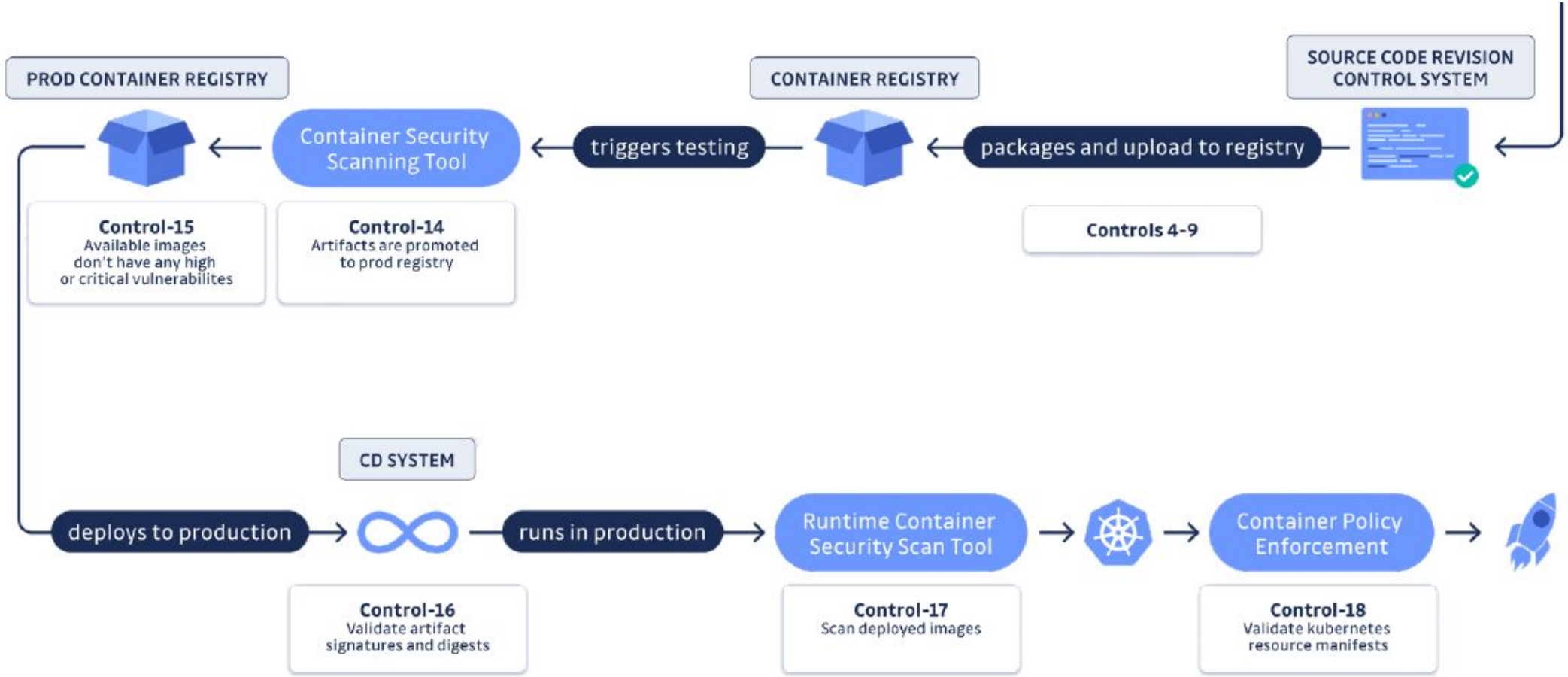
Securing supply chain



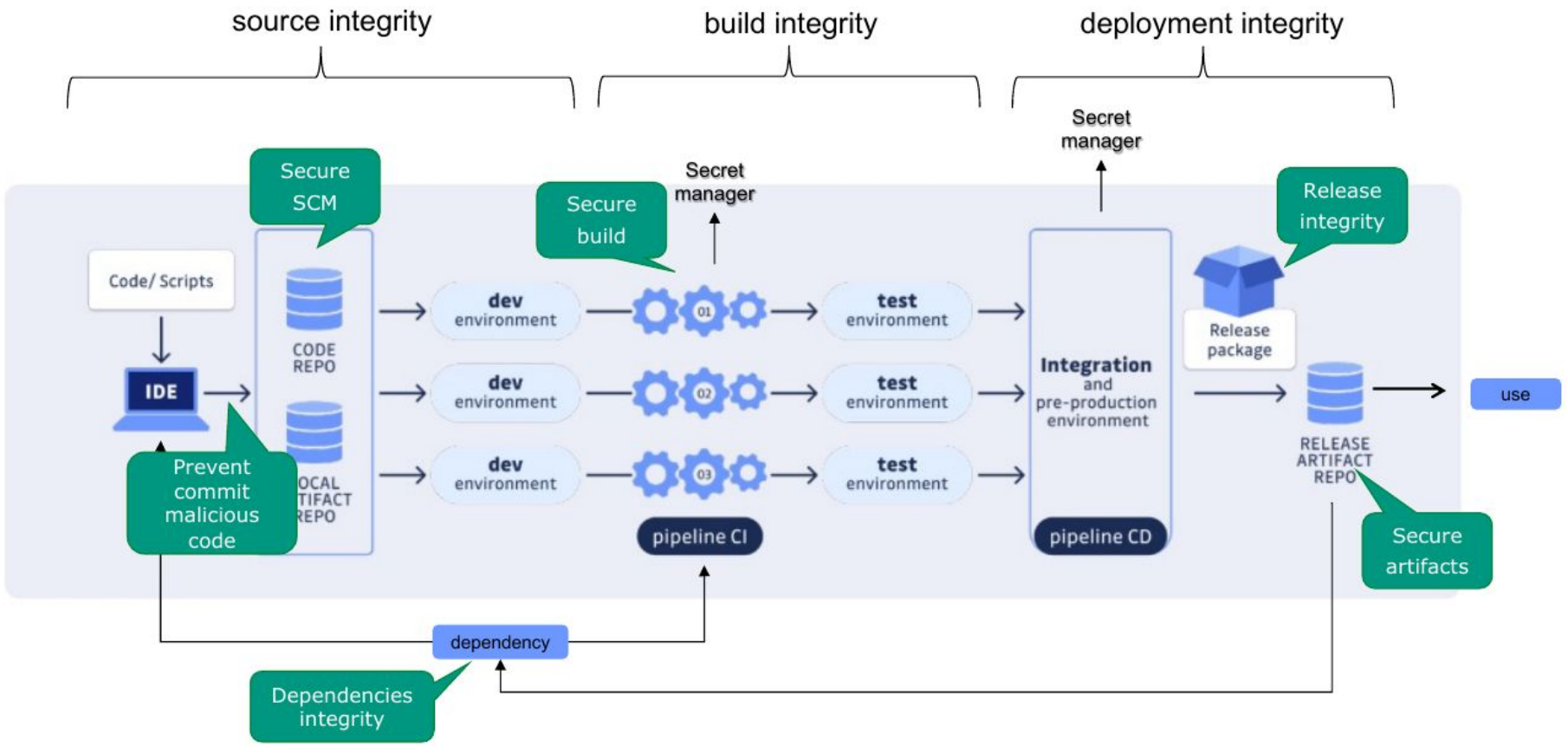
Securing supply chain



Securing supply chain



Securing supply chain



Examples of Attacks

- IDE - Octopus Scanner Malware (2020)
  - Attacked NetBeans IDE
- Dependencies - Log4Shell attack (2021)
  - Vulnerability in Apache Log4j logging framework
- Repos - Canonical Github account breach (2019)
  - Github account compromised
- Pipeline CI - Codecov (2022)
  - Script to share env variables from the CI of Codecov customers
- Release Package - SolarWinds (2020)
  - Access to SolarWinds development infrastructures and malware injection

More available at https://www.sonatype.com/resources/vulnerability-timeline

Guidelines

- Center for Internet Security Software-Supply-Chain-Security (pdf)
  - More than 100 recommendations across five core areas
    - Source code (e.g., Scan for vulnerabilities in code via SAST, SCA and SBOM to look for vulnerabilities in open-source software components)
    - Build pipelines (e.g., Make build workers single-use)
    - Dependencies (e.g., Require SBOM's from third party suppliers)
    - Artifacts (e.g., Ensure artifacts are signed by the pipeline itself)
    - Deployment (e.g., Make deployments automated and reproducible)
- NIST's Software Supply Chain Security Guidance
- Emerging frameworks & Tools - Supply Chain Levels of Software Artifacts (SLSA)

SRE Lessons

- A new mission cannot always be achieved with old tools and methods
- If you have to wait for a human to detect an error, you've already lost
- If a well-intentioned human can "break" it, it was already broken
- On-call can't be off-loaded (assign responsibilities to the people creating the system)
- Elite warrior/hero culture is a trap → better prudent design and preventive planning
- Monitoring is about ensuring the steady flow of traffic, not a steady flow of alerts
- Prevention of all errors is impossible, costly, and annoying to anyone trying to get things done

SRE Lessons

- Build better tools and frameworks to reduce the toil of service launches
- SRE is a pull function, not a push function → tools should be attractive for your users
- Failure is inevitable. Get good at handling it, rather than betting everything on avoiding it (may not apply to all things!)
- Organizations produce the results that they value, not the results their components strive for (don't use SRE as only a buzzword)

Learning from Failure

- Game Days → running real-life, large-scale failure tests
  - E.g., shutting down a data center
- Red Team exercises
  - Red Team identifies weaknesses in the system that they believe can be exploited
  - Work as ethical hackers to attack the live system
  - Blue Team is are the people running, supporting, and monitoring the system
  - Better if done in production

Outline

- DevSecOps
- Site Reliability Engineering
- Best practices
- **Standards and Laws**

Standards

FKOBST 358-1

ISO 2700XX

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

SP 800 – 37
SP 800 – 115
SP 800-171A
SP 800-171

Common Criteria

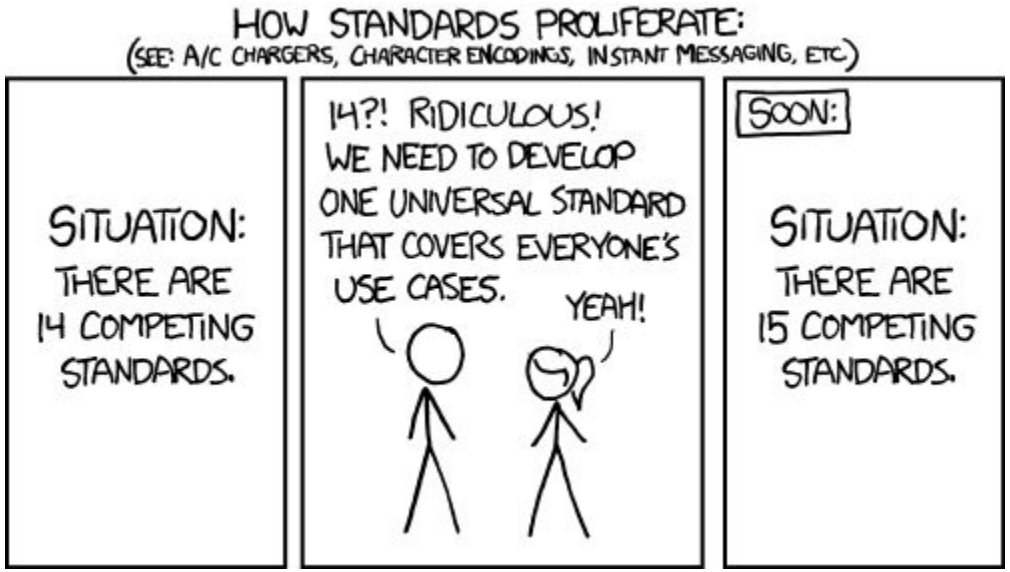
ISA International Society of Automation

ISA/IEC 62443 Standards Industrial Process Control

Standards

- ISO/IEC 27000 → family of Information Security Management Systems standards
  - Broad in scope, covering more than just privacy, confidentiality, IT/technical/cybersecurity issues
  - Not technical (e.g., not asked if API checked for OWASP breaches)
  - Look at the controls to mitigate risk
    - E.g., what controls do you have to mitigate data loss due to software vulnerabilities?
- NIST, … → good-practice framework
- More info at https://introductiontostandardization.ds.dk/

Standards


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

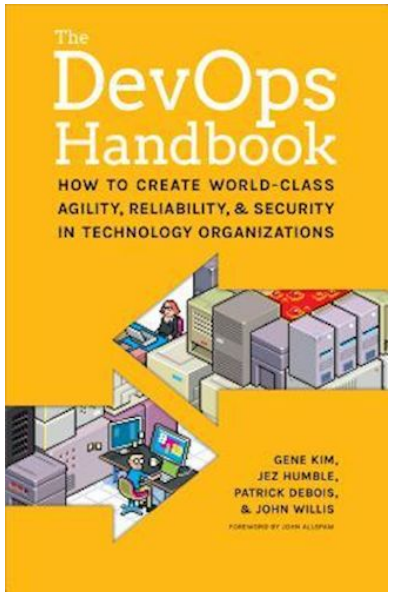
NIS 2 Directive

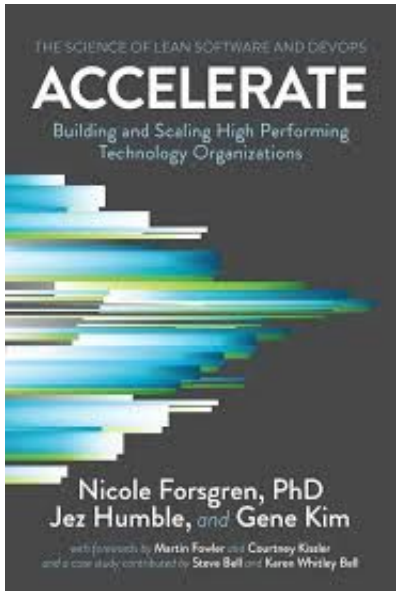
- Applies to companies > 50 people or > 10m euro revenue in
  - high critical sectors (energy, transport, banking, health, digital infrastructure, ICT service management, public administration, …)
  - other critical sectors (postal services, waste management, digital providers, desearch)
- governing bodies must
  - approve cybersecurity risk management measures
  - oversee their implementation
  - liable for any breaches
  - undergo cybersecurity training and provide similar training to their employees

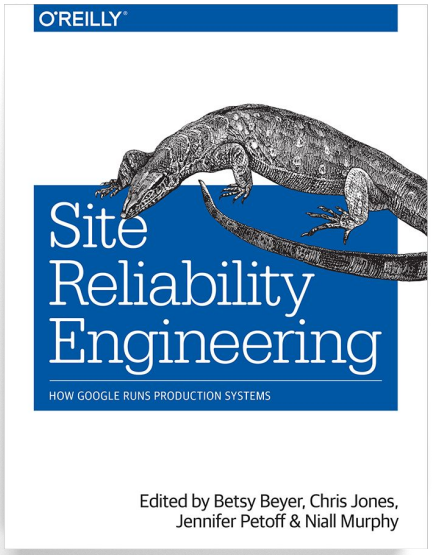
NIS 2 Directive - Minimal

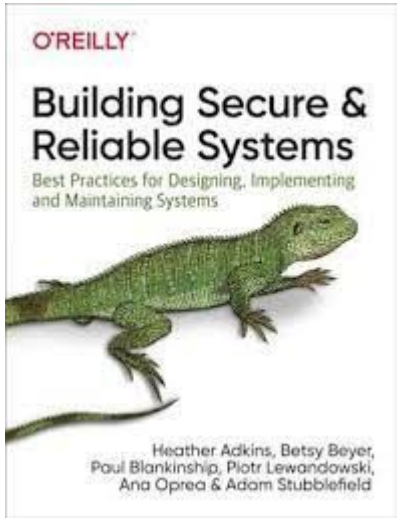
- Risk analysis
- Incident handling and reporting
- Business continuity (e.g., disaster recovery, crisis management)
- Supply chain security
- Policies and procedures to assess the effectiveness of cybersecurity risk-management
- Basic cyber hygiene practices and cybersecurity training
- Access control policies and asset management;
- Cryptography, multi-factor authentication … (where appropriate)

References


The
DevOps
Handbook
HOW TO CREATE WORLD-CLASS
AGILITY, RELIABILITY, & SECURITY
IN TECHNOLOGY ORGANIZATIONS
GENE KIM,
JEZ HUMBLE,
PATRICK DEBOIS,
& JOHN WILLIS


THE SCIENCE OF LEAN SOFTWARE AND DEVOPS
ACCELERATE
Building and Scaling High Performing
Technology Organizations
Nicole Forsgren, PhD
Jez Humble, and Gene Kim


O'REILLY
Site
Reliability
Engineering
HOW GOOGLE RUNS PRODUCTION SYSTEMS
Edited by Betsy Beyer, Chris Jones,
Jennifer Petoff & Niall Murphy


O'REILLY
Building Secure &
Reliable Systems
Best Practices for Designing, Implementing
and Maintaining Systems
Heather Adkins, Betsy Beyer,
Paul Blankinship, Piotr Lewandowski,
Ana Oprea & Adam Stubblefield

FIND OUT MORE

https://gamess.dk/

GameSS

WHO IS BEHIND GameSS

Partners behind the project

SDU

DTU

IT UNIVERSITY OF CPH

AALBORG UNIVERSITET

Happy 42

Collaborators

ERHVERVSAKADEMI AARHUS

kea
KØBENHAVNS ERHVERVSAKADEMI

Dania
ERHVERVSAKADEMI

UCN

UCL Erhvervsakademi og Professionshøjskole

Supported by

Uddannelses- og Forskningsministeriet

Ministry of Higher Education and Science Denmark