สยามชำนาญกิจ
SIAM CHAMNAN KIT

Secure Coding

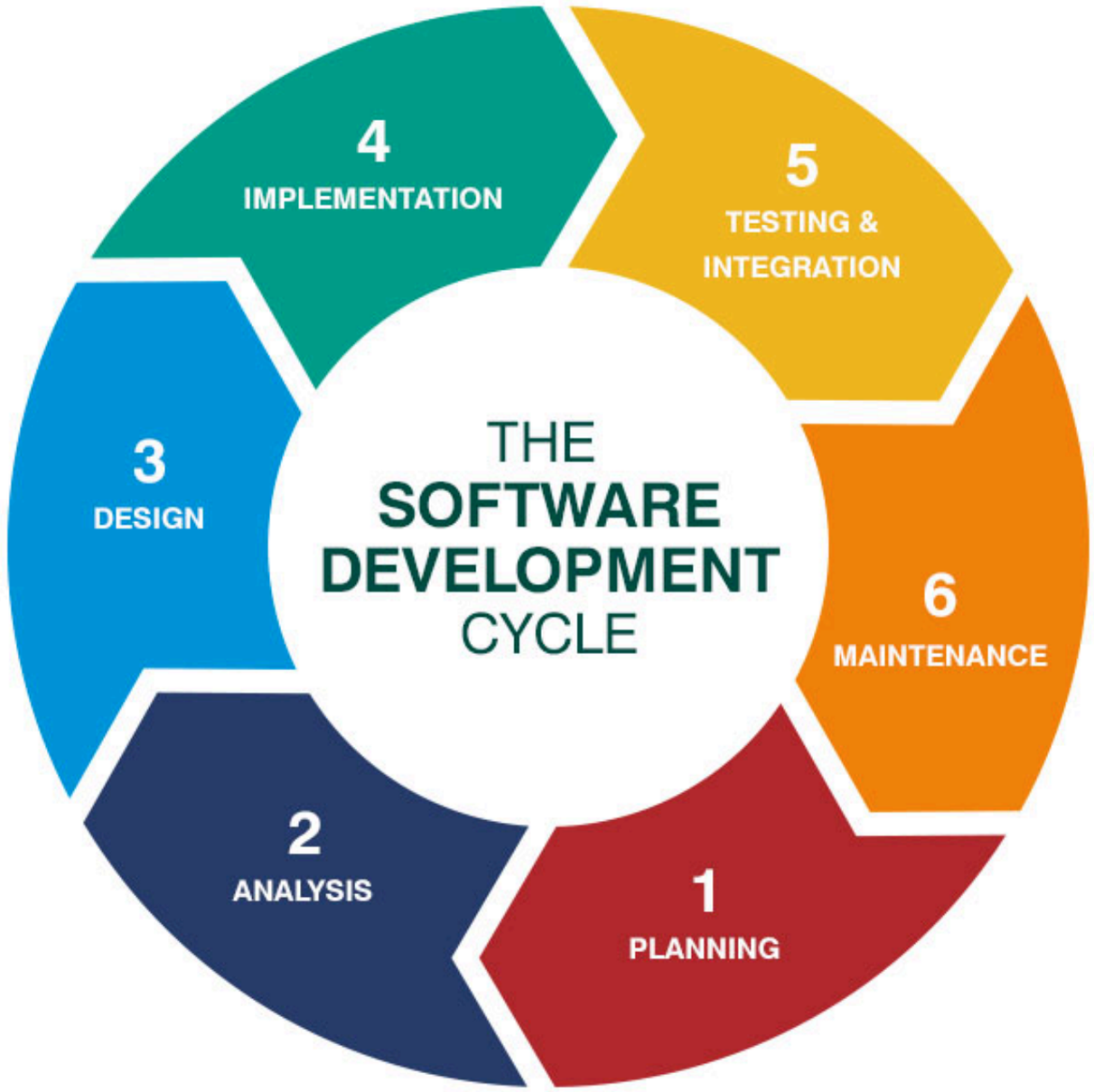
https://github.com/up1/course-secure-coding

Secure Coding

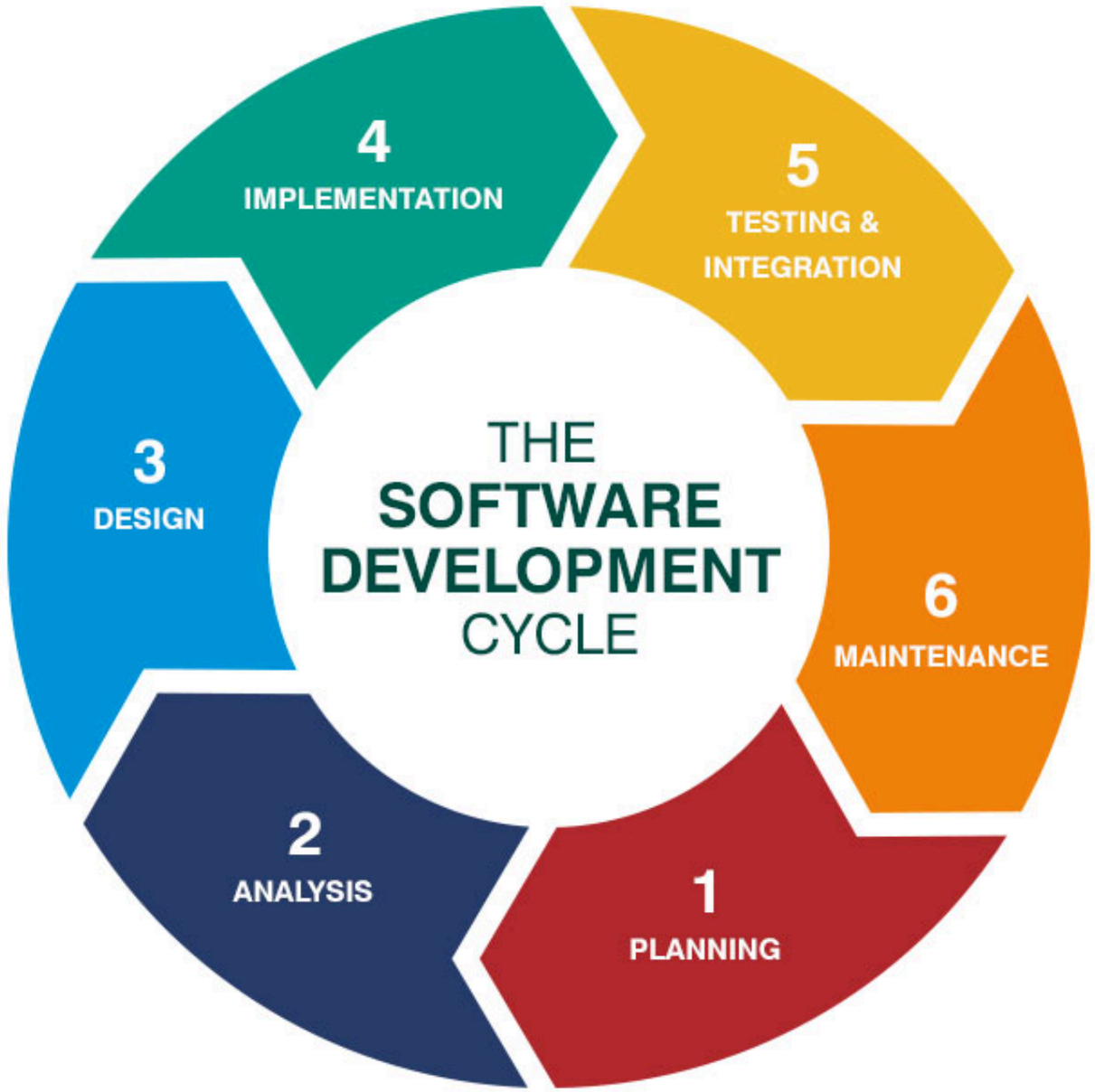
Software Delivery

Fast (Time to market)
High quality
High secure system

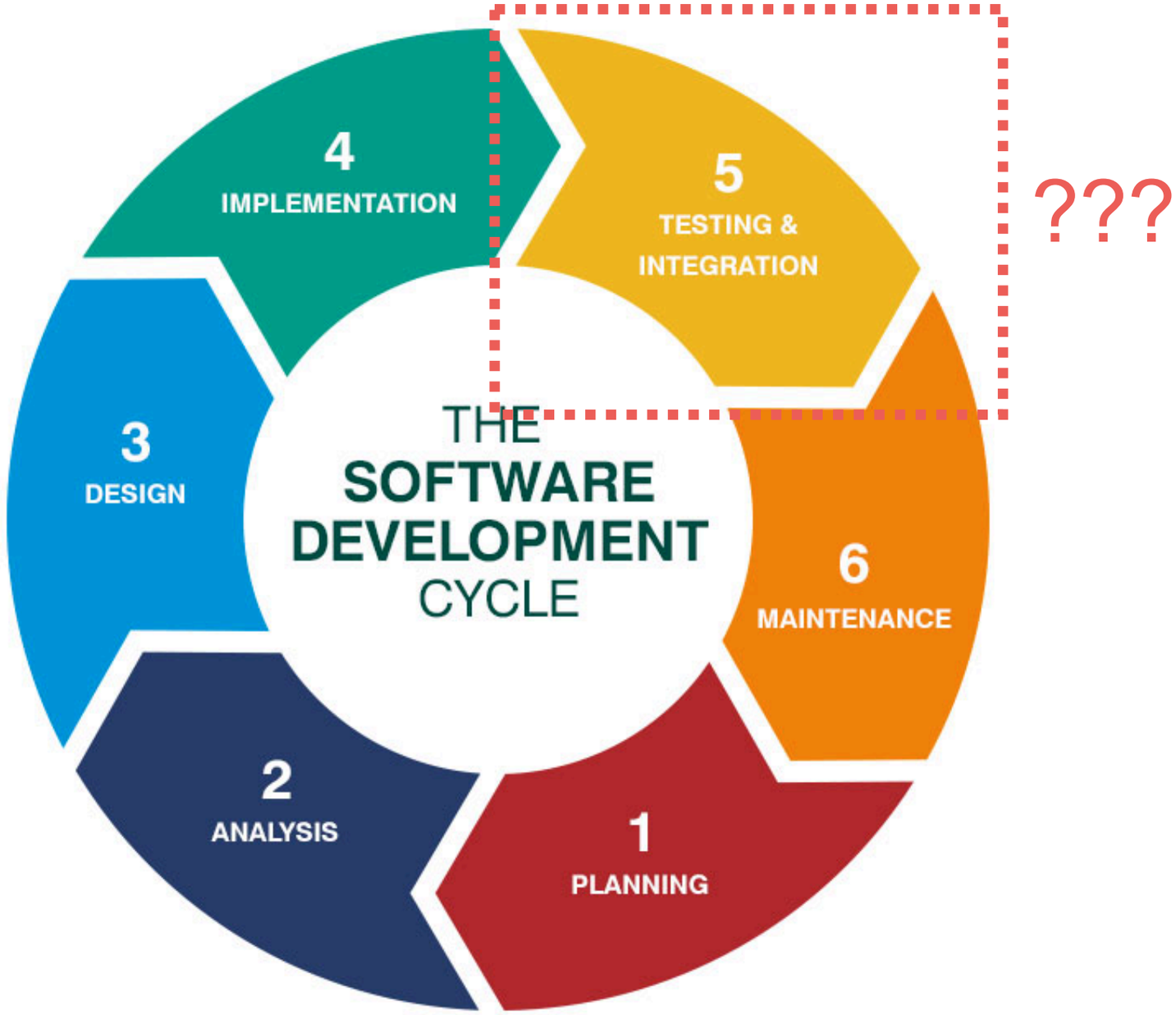
Software Development Life Cycle



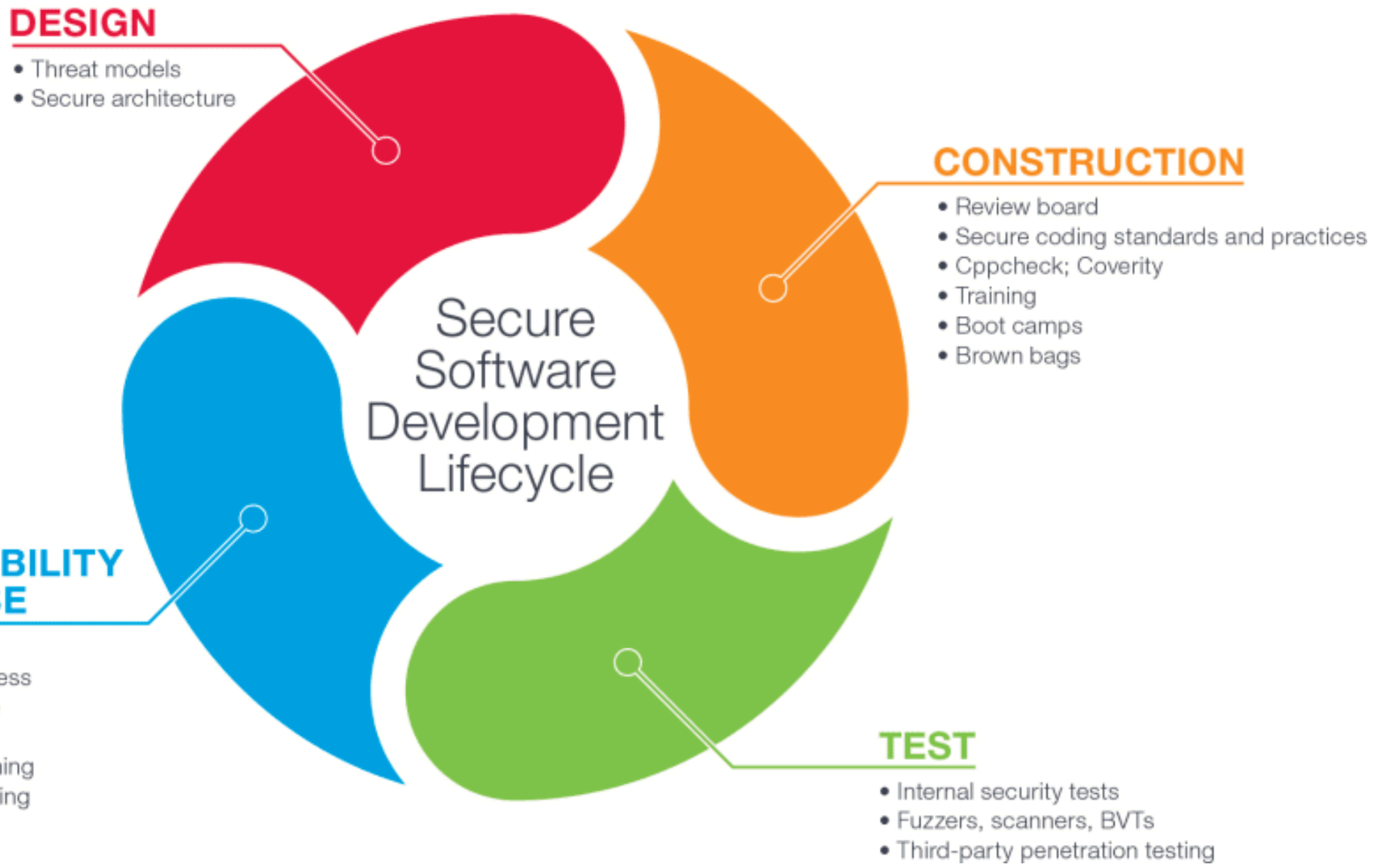
Security Testing ?



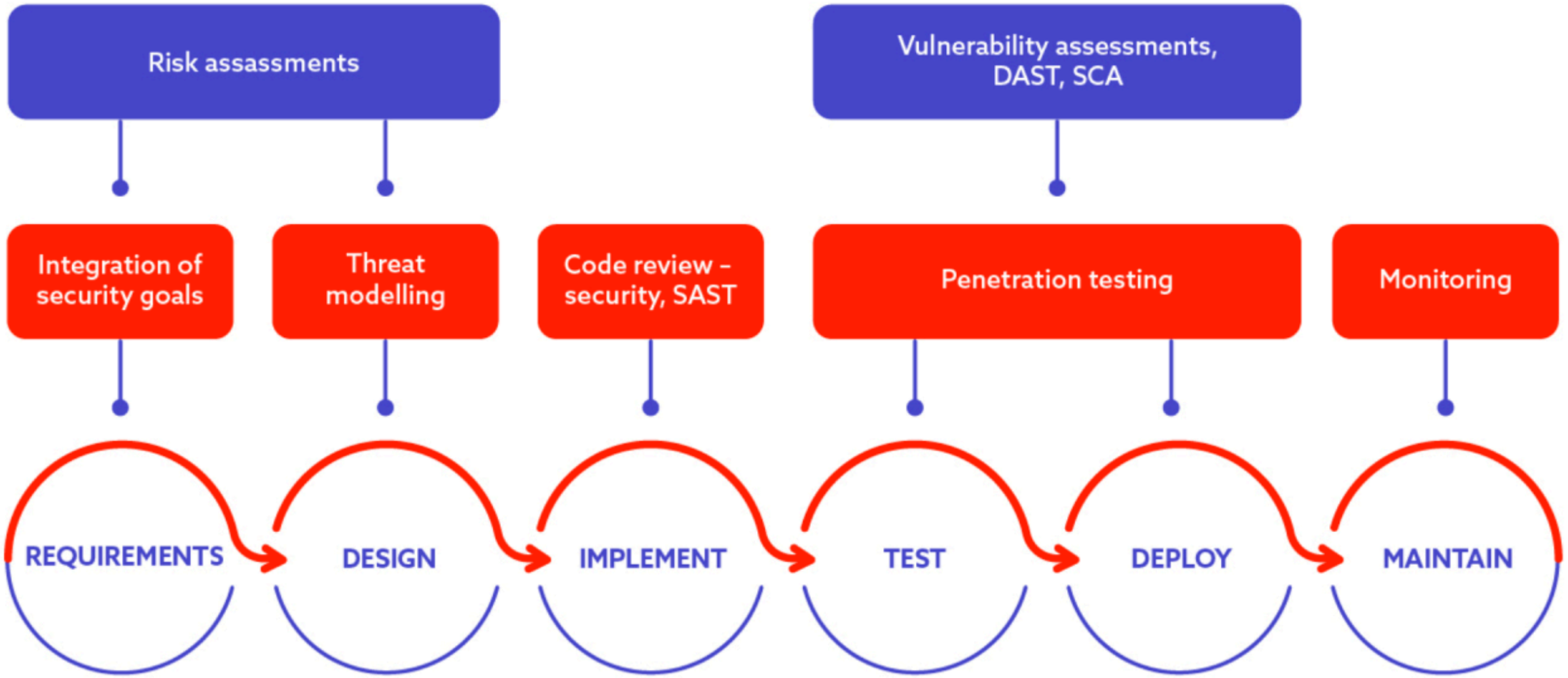
Security Testing ?



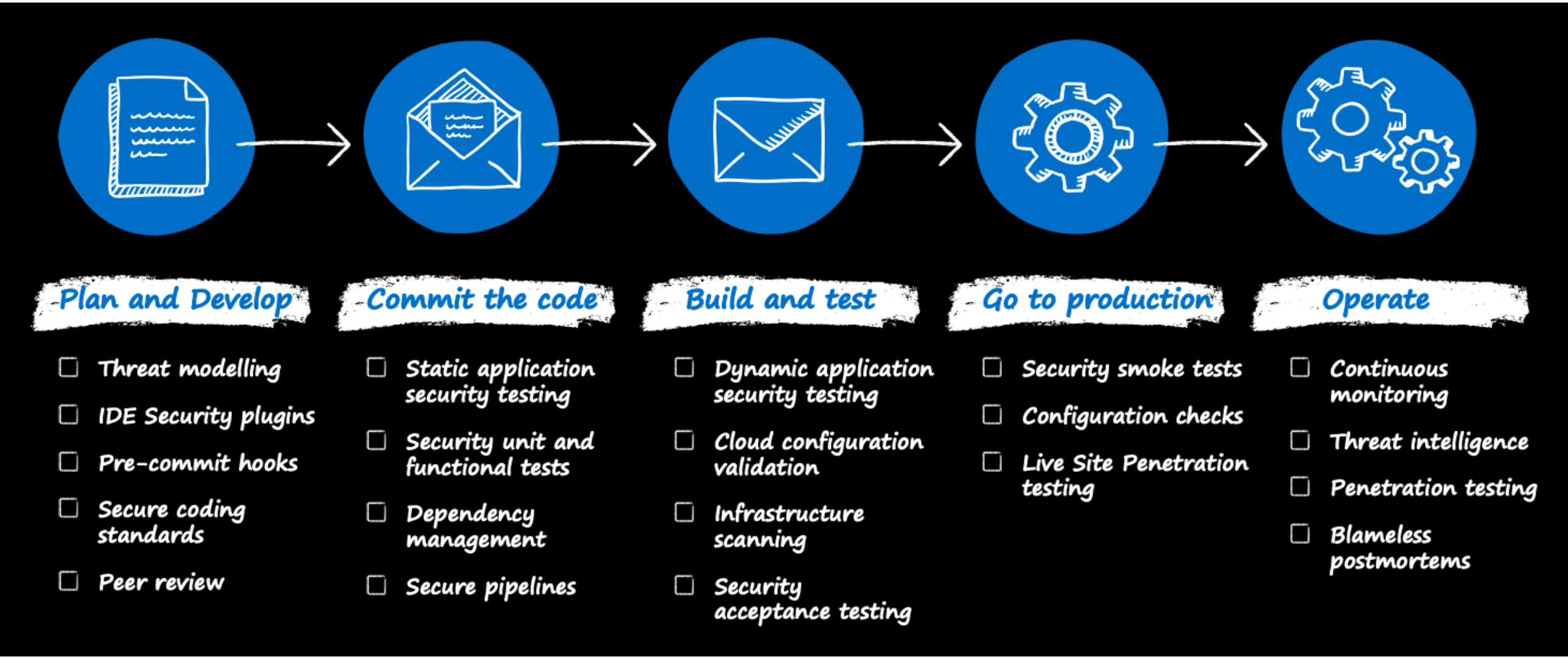
SDLC with Secure



SDLC with Secure

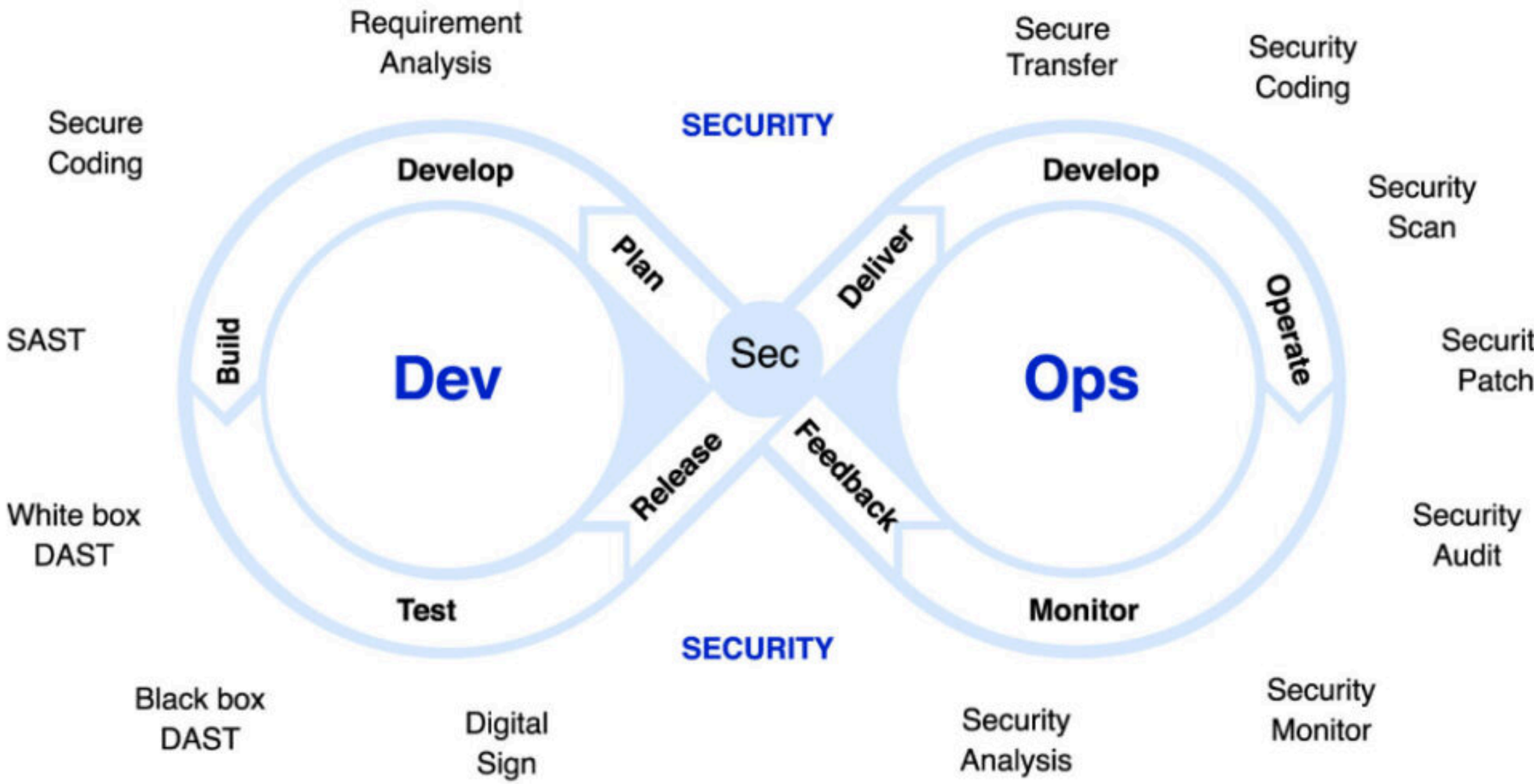


SDLC with Secure



https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/secure/devsecops-controls

DevSecOps



Goals

Security review

Reduce damage

Reduce cost

Early detection

Secure in delivery process

Secure in Delivery Process

Requirements	Planning & Design	Development	Testing / Pre-Deployment	Deployment	Sanitization & Disposal
Security Requirements to be shared such as checklist, Secure coding standards, Application Security guidelines, etc. Awareness sessions to developers	Review Design documents, System Analysis and Architecture review, Privacy Impact Assessment	Threat Modelling, Document Security Controls, Architecture review, Code review Documentation	DAST, SAST, Security/Penetration Testing, VAPT, MBSS compliance Check, Container Security guidelines check, API Security Testing	Conduct a final Security review, Integrate other security systems such as SIEM, IAM etc.	ICT Responsibilities: Disposal plan, media Sanitization, Closure of system
Tools: Manual (Documents need to be shared)	Tools: Manual	Tools: TM Tools, code review tools and manual	Tools: Burp suite, automated scanners, code scanners, VA Scanners, PT Tools, Compliance Scanners		

1. Requirements

Share security requirement

Security Checklist	Secure Coding standard	Application security guidelines

Secure Coding Standard

Avoid vulnerabilities
Focus on Web and API security

OWASP Top 10

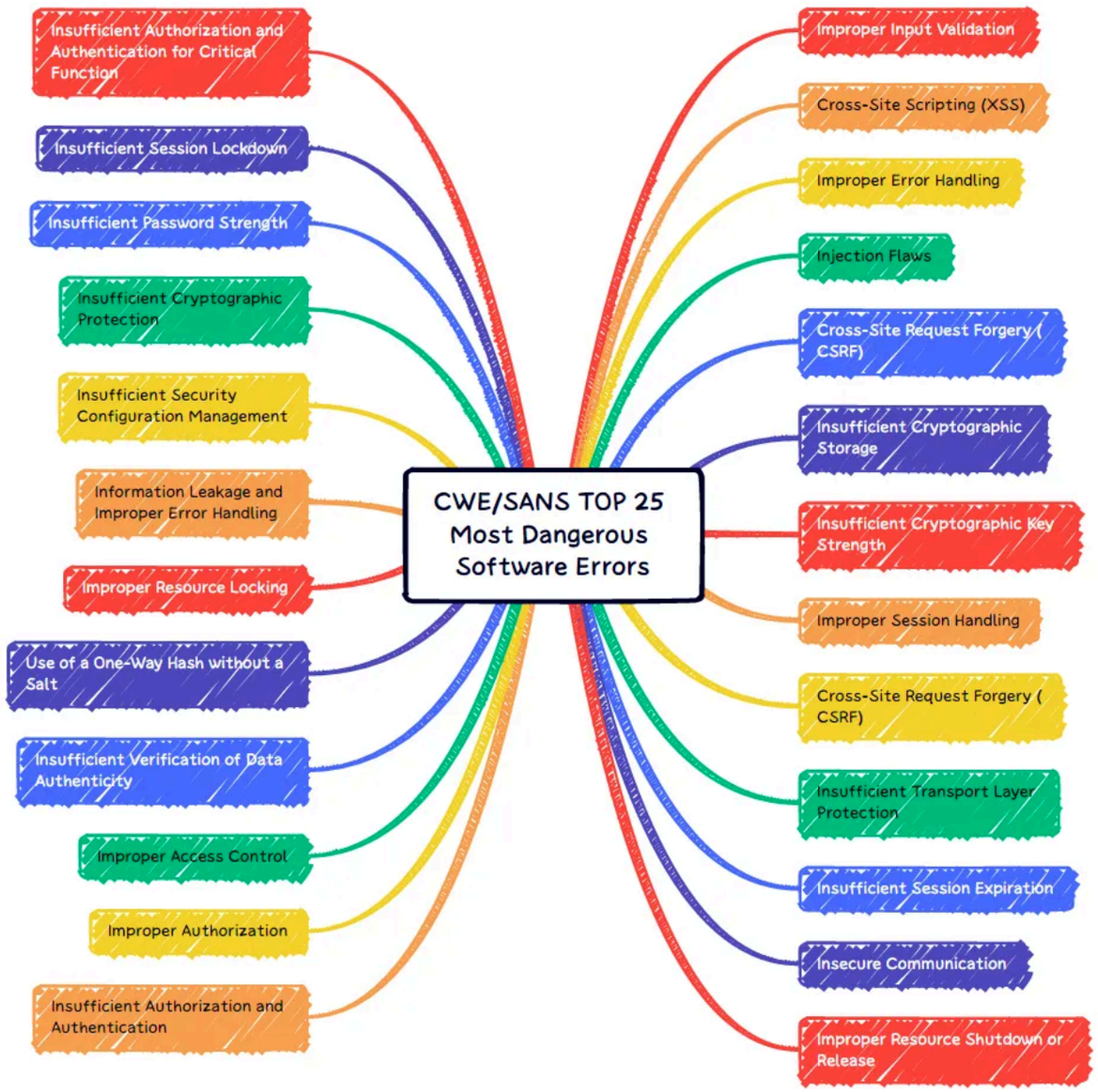
SANS 20 software errors

OWASP Top 10

https://owasp.org/API-Security/editions/2023/en/0x11-t10/

CWE/SANS Top 25 Errors

Common Weakness Enumeration



https://cwe.mitre.org/top25/

More …

Access administration
Authentication and authorization
Logging and monitoring system

TOP 10

https://owasp.org/Top10/

A01 :: Broken Access Control

https://owasp.org/Top10/A01_2021-Broken_Access_Control/

A01 :: Broken Access Control

Violation of the principle of least privilege
Bypassing access control
Access API with mission access control
CORS misconfiguration
CORS allows untrusted origin

A01 :: Broken Access Control

Expose sensitive information to unauthorized user
Insertion of sensitive information to sent data
Cross-Site Request Forgery (CSRS)

Required

User should only act based on specific permission
Incorrect permission -> unauthorized access

How to prevent ?

Deny access by default
Avoid duplication of access control log
Enforce **user ownership** when manipulating data

Workshop with NodeJS

Broken Access Control !!

GET /profile?username=alice

GET /profile?username=bob

Potential Security Issues

@nearform/sql

SQL Injection	Authentication	Input validation
Data exposure	Dependency security	Error handling

Solutions !!

Create and run automated test
Check JWT token in HTTP request header

Multiple factor authentication (MFA)	Strong password	Log all failed attempt

A02 :: Cryptographic Failure

https://owasp.org/Top10/A02_2021-Cryptographic_Failures/

A02 :: Cryptographic Failure

Weak or inexistent **cryptography** of sensitive data
Anything protected by privacy laws or regulations

Password	Credit-card number	Personal information
Health record	Business secret	

Common mistake !!

Weak or outdated cryptographic algorithm

Common mistake !!

Weak secret keys, use default

Use keys from online tutorials

Lack of traffic encryption (HTTPS)

Insufficient entropy in seed generation

How to prevent ?

Check all sensitive data is encrypted
Avoid storing sensitive data unnecessarily
Use up-to-date and strong standard algorithms
Proper key-secret management
Disable caching for response that contain sensitive data

Don’t store private keys in Git

Workshop with NodeJS

Weak hashing algorithm such as **md5**

```
let hashPassword = async function hashPassword(password) {
  return md5(password)
}

let comparePassword = async function comparePassword(password, hash) {
  return md5(password) === hash
}
```

Please change to Bcrypt

Potential Security Issues

Weak hashing algorithm

Lack of salt

Plain text password

MD5 !!

Security [edit]

One basic requirement of any cryptographic hash function is that it should be computationally infeasible to find two distinct messages that hash to the same value. MD5 fails this requirement catastrophically. On 31 December 2008, the CMU Software Engineering Institute concluded that MD5 was essentially "cryptographically broken and unsuitable for further use".[17] The weaknesses of MD5 have been exploited in the field, most infamously by the Flame malware in 2012. As of 2019, MD5 continues to be widely used, despite its well-documented weaknesses and deprecation by security experts.[18]

A collision attack exists that can find collisions within seconds on a computer with a 2.6 GHz Pentium 4 processor (complexity of $2^{24.1}$).[19] Further, there is also a chosen-prefix collision attack that can produce a collision for two inputs with specified prefixes within seconds, using off-the-shelf computing hardware (complexity 2^{39}).[20] The ability to find collisions has been greatly aided by the use of off-the-shelf GPUs. On an NVIDIA GeForce 8400GS graphics processor, 16–18 million hashes per second can be computed. An NVIDIA GeForce 8800 Ultra can calculate more than 200 million hashes per second.[21]

https://en.wikipedia.org/wiki/MD5#Security

Use Bcrypt

```
import { hash, compare } from 'bcrypt'

const saltRounds = 10

export async function hashPassword(password) {
  return await hash(password, saltRounds)
}

export async function comparePassword(password, hash) {
  return await compare(password, hash)
}
```

https://github.com/kelektiv/node.bcrypt.js

Testing for Weak Cryptography

Weak transport layer security (TLS)
Padding oracle
Sensitive information sent via unencrypted channel
Weak encryption

https://owasp.org/www-project-web-security-testing-guide/stable/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/README

A03 :: Injection

https://owasp.org/Top10/A03_2021-Injection/

A03 :: Injection

Cross-Site Scripting (XSS)
SQL injection
External control of filename or path

A03 :: Injection

Lack to validate user’s input

Validated	Filters	Sanitized

Input Validate Cheat Sheet

Input Validation Cheat Sheet

Introduction

This article is focused on providing clear, simple, actionable guidance for providing Input Validation security functionality in your applications.

Goals of Input Validation

Input validation is performed to ensure only properly formed data is entering the workflow in an information system, preventing malformed data from persisting in the database and triggering malfunction of various downstream components. Input validation should happen as early as possible in the data flow, preferably as soon as the data is received from the external party.

https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

A04 :: Insecure Design

https://owasp.org/Top10/A04_2021-Insecure_Design/

A05 :: Security Misconfiguration

https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

More …

OWASP Application Security Verification Standard

https://owasp.org/www-project-application-security-verification-standard/

2. Planing and Design

2. Planning and Design

Understand requirements, project timeline
Technology selection
Human resources
Required software and hardware

Security Architect	Security Officer	Security Tester

Security testing

What to test ?

When to test ?

What tools are required ?

Secure Design ?

Check all possible security design implementation
Security risk assessment
Review all design document
Threat modeling

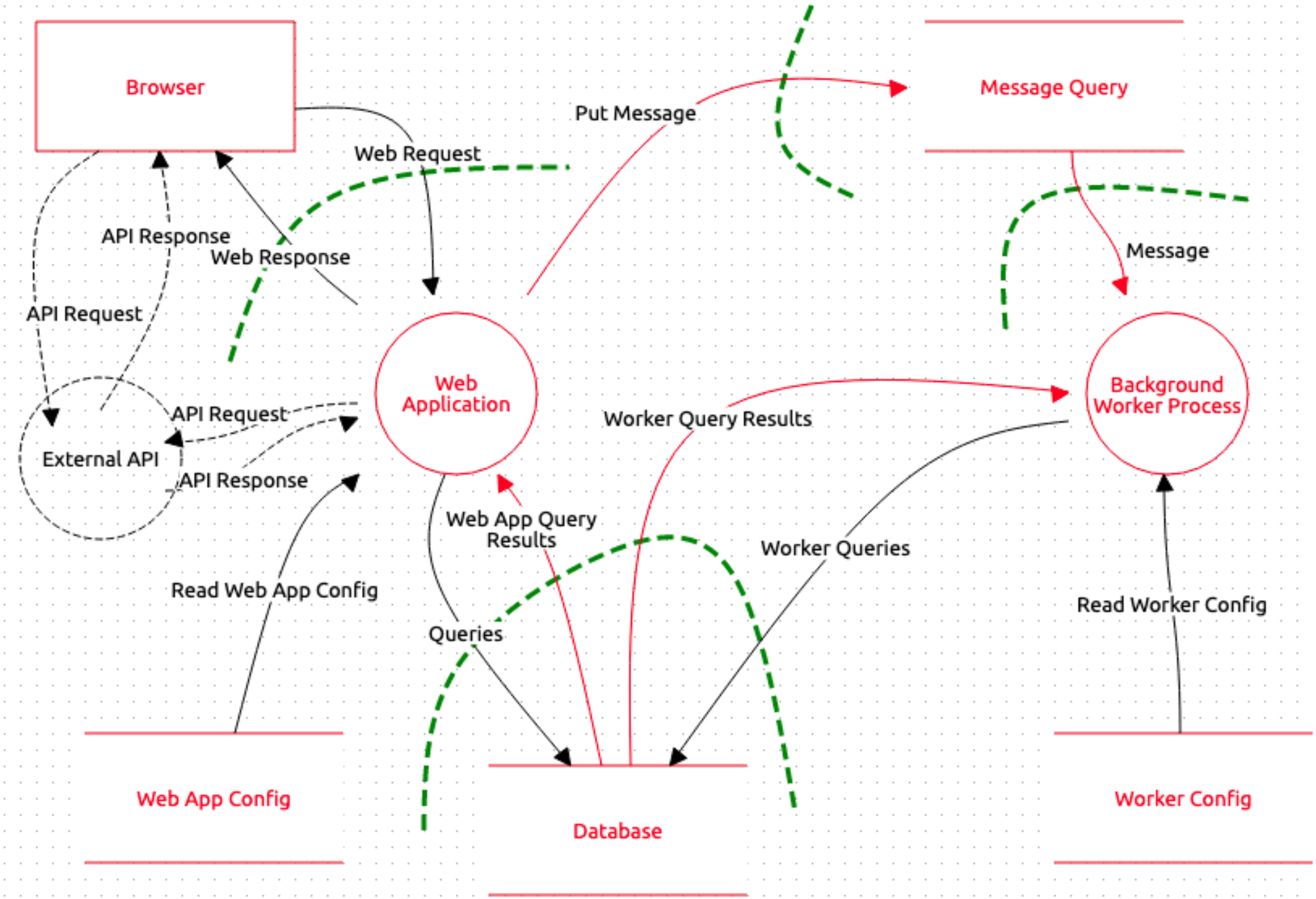
Threat Modeling

Process to analyze and modeling threat
Find solution and tools to protect/prevent

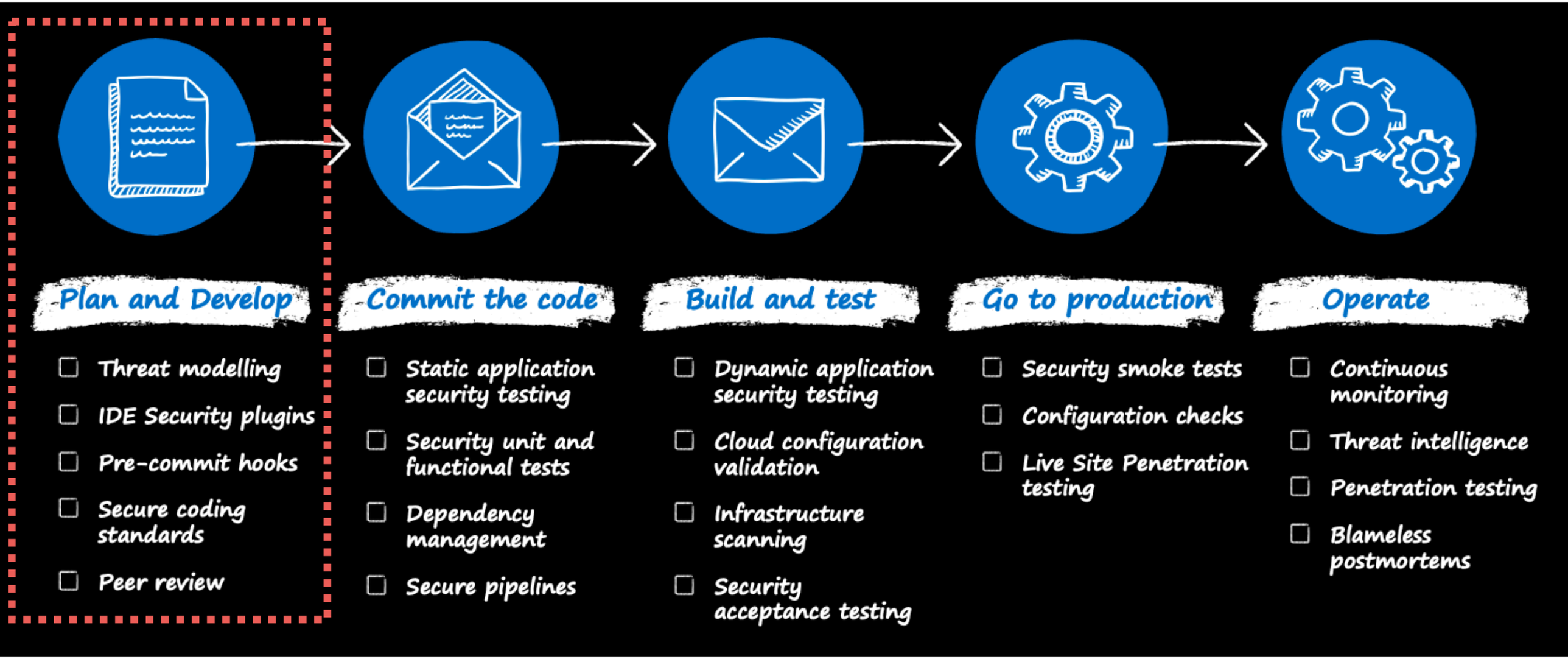
How attackers can abuse app ?	How to fix vulnerabilities ?	How important to fix issues ?

https://owasp.org/www-community/Threat_Modeling

Start with current Architecture



Plan and Develop



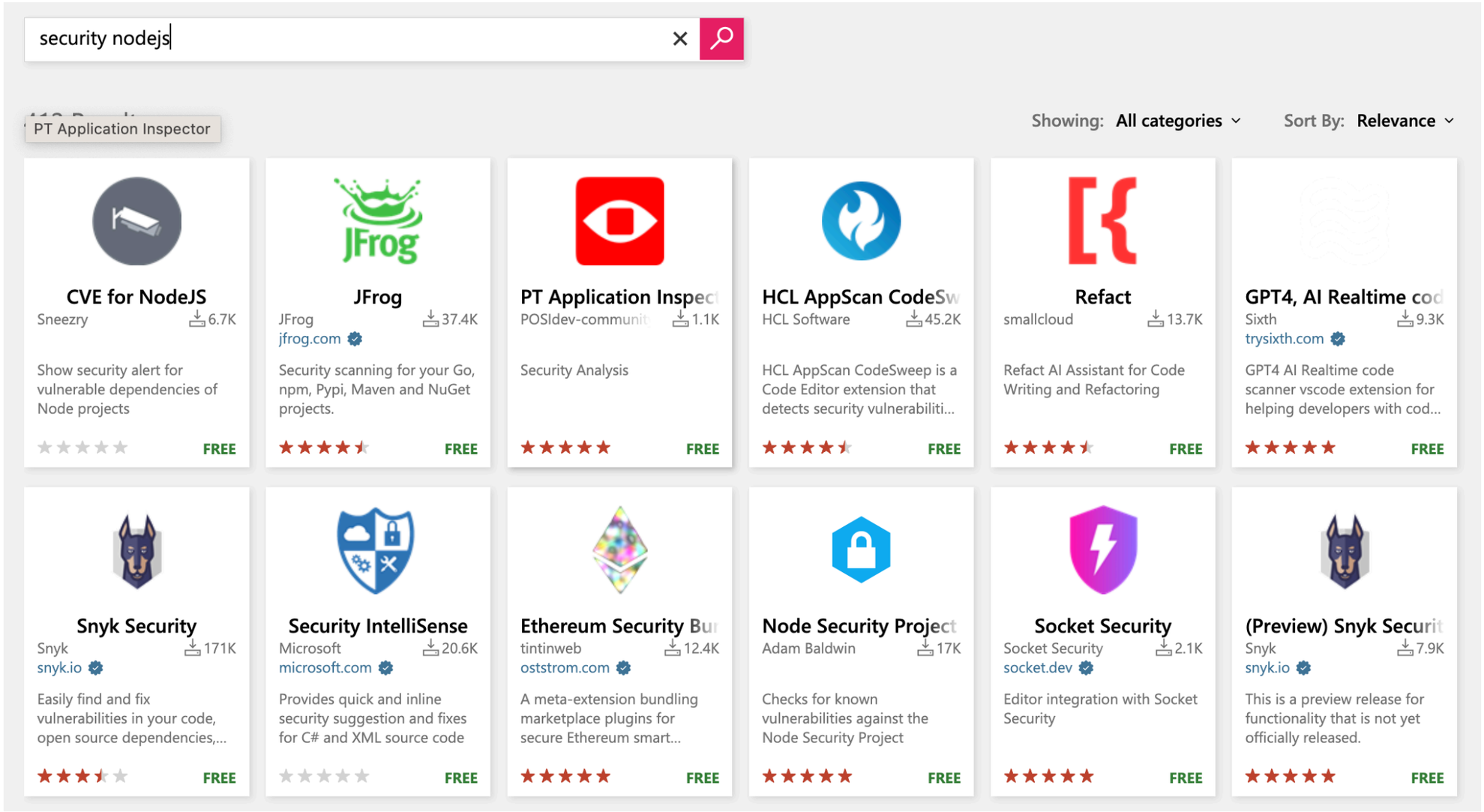
https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/secure/devsecops-controls

Plan and Develop

Threat modeling
IDE security plugins
Pre-commit hooks
Peer reviews and secure coding standard

IDE Security Plugins

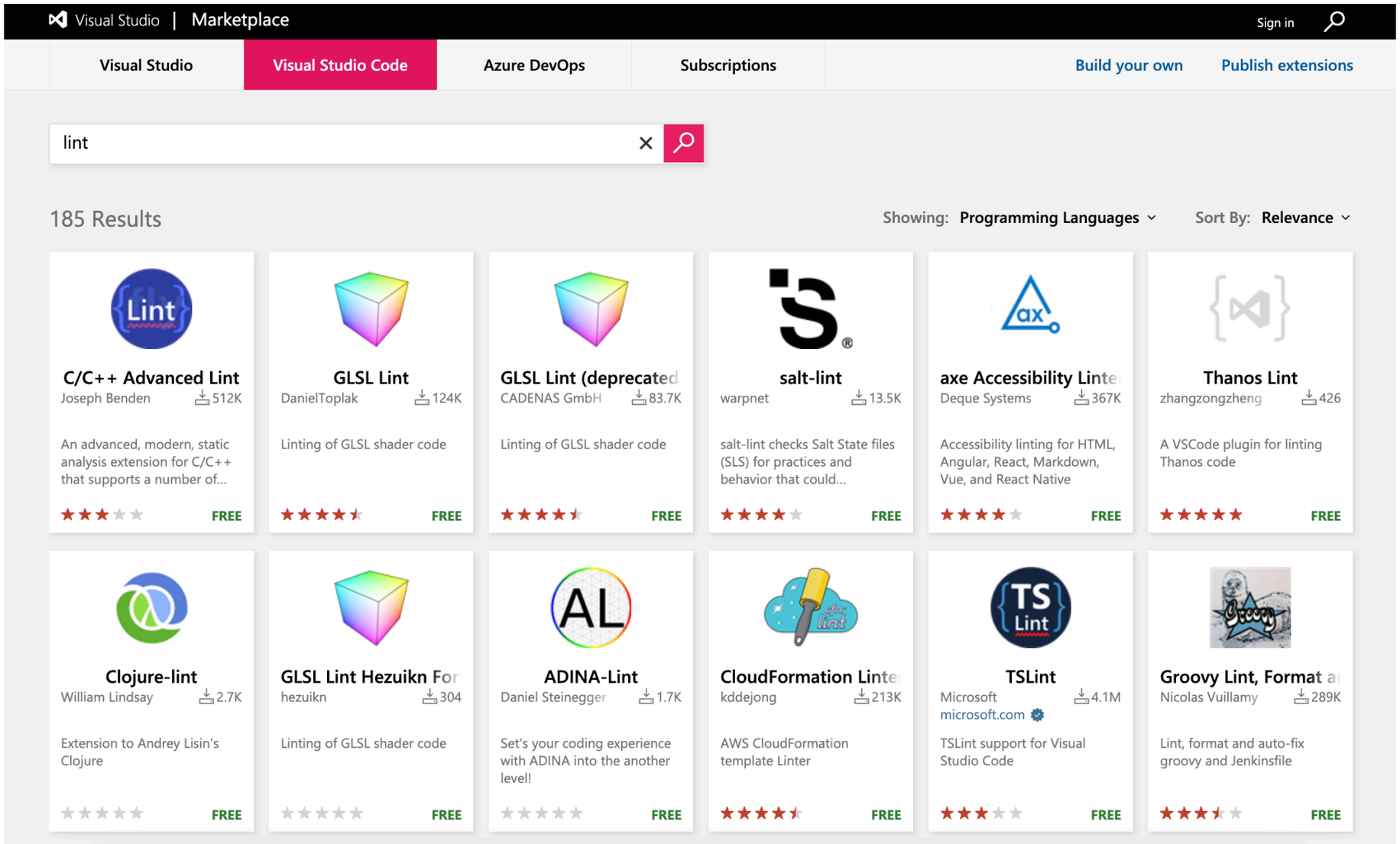
Lint and Static code analysis



https://marketplace.visualstudio.com/vscode

IDE Security Plugins

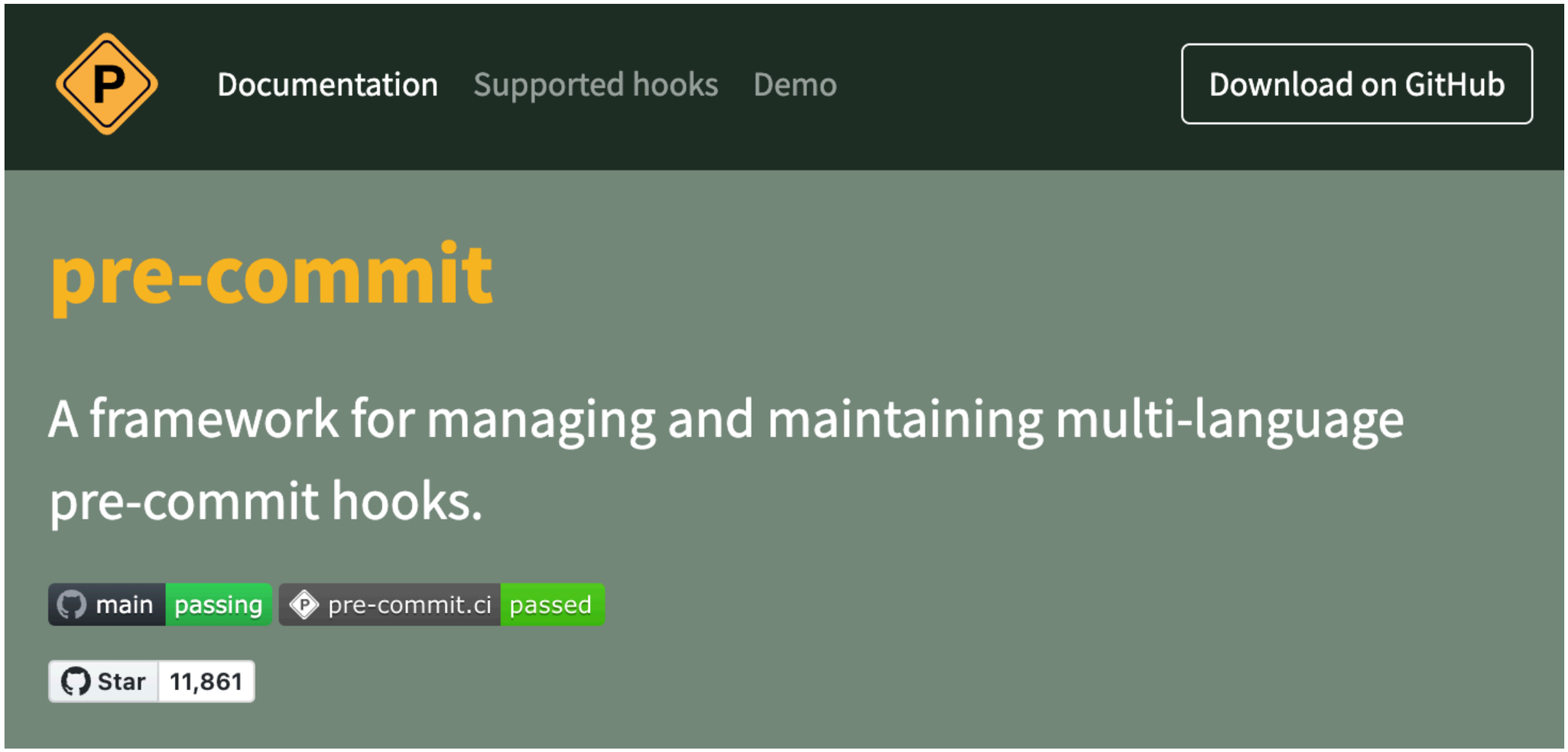
Lint and Static code analysis



https://marketplace.visualstudio.com/vscode

Git pre-commit hooks

Self-test in developer machine (fast feedback)



https://pre-commit.com/

Git pre-commit hooks

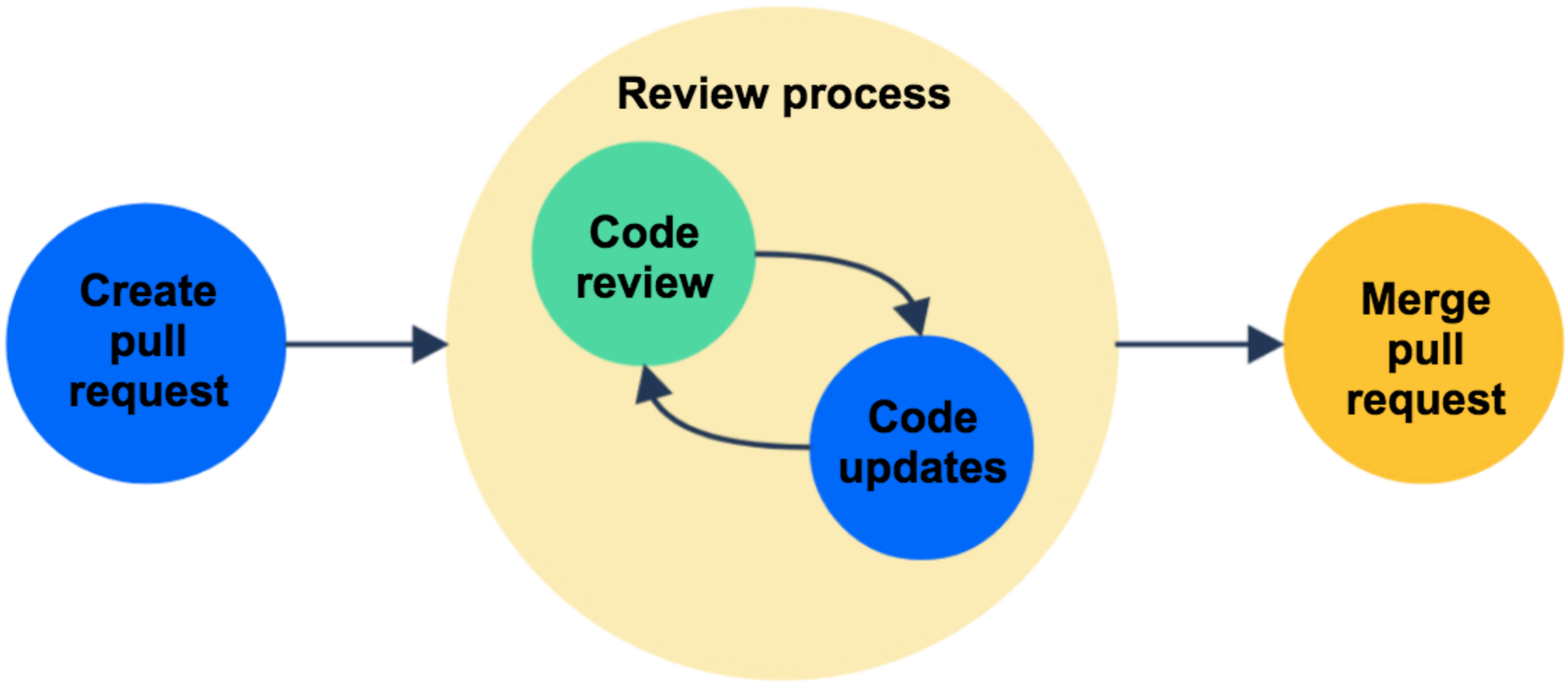
$git commit -m "fix(faker): credit card from faker error”

```
✔ Preparing lint-staged...
✔ Running tasks for staged files...
✔ Applying modifications from tasks...
✔ Cleaning up temporary files...
```

https://typicode.github.io/husky/

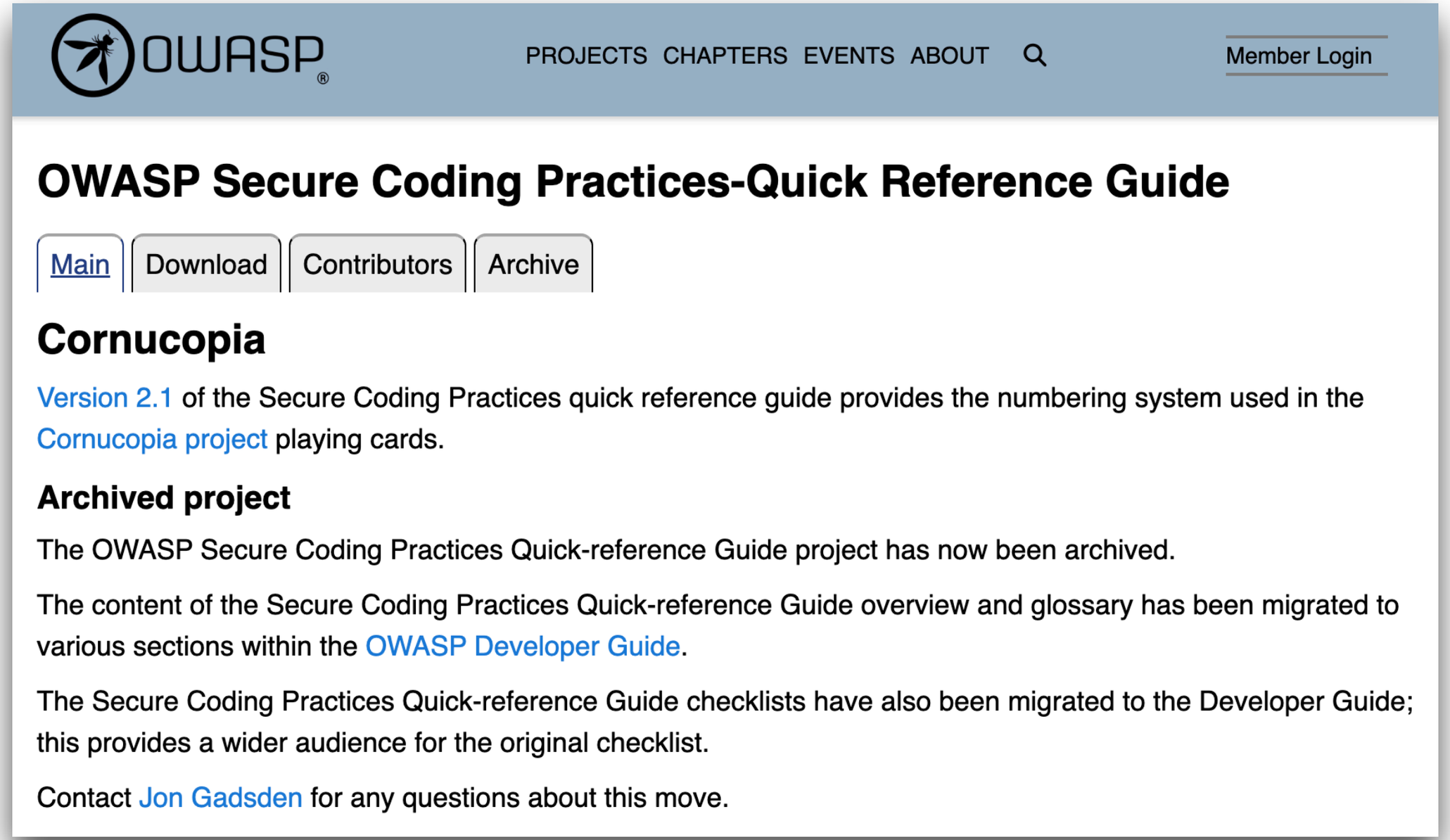
Peer Review and Secure coding standard

Use pull request (PR) and review



https://support.atlassian.com/bitbucket-cloud/docs/use-pull-requests-for-code-review/

OWASP Secure Coding Practices



https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/

When develop committed code …

Dependency checking
Static Application Security Testing (SAST)
Security pipeline
Dynamic Application Security Testing (DAST)

Dependency Checking

OWASP Dependency -Check

GitHub Dependabot

NPM audit

https://owasp.org/www-project-dependency-check/

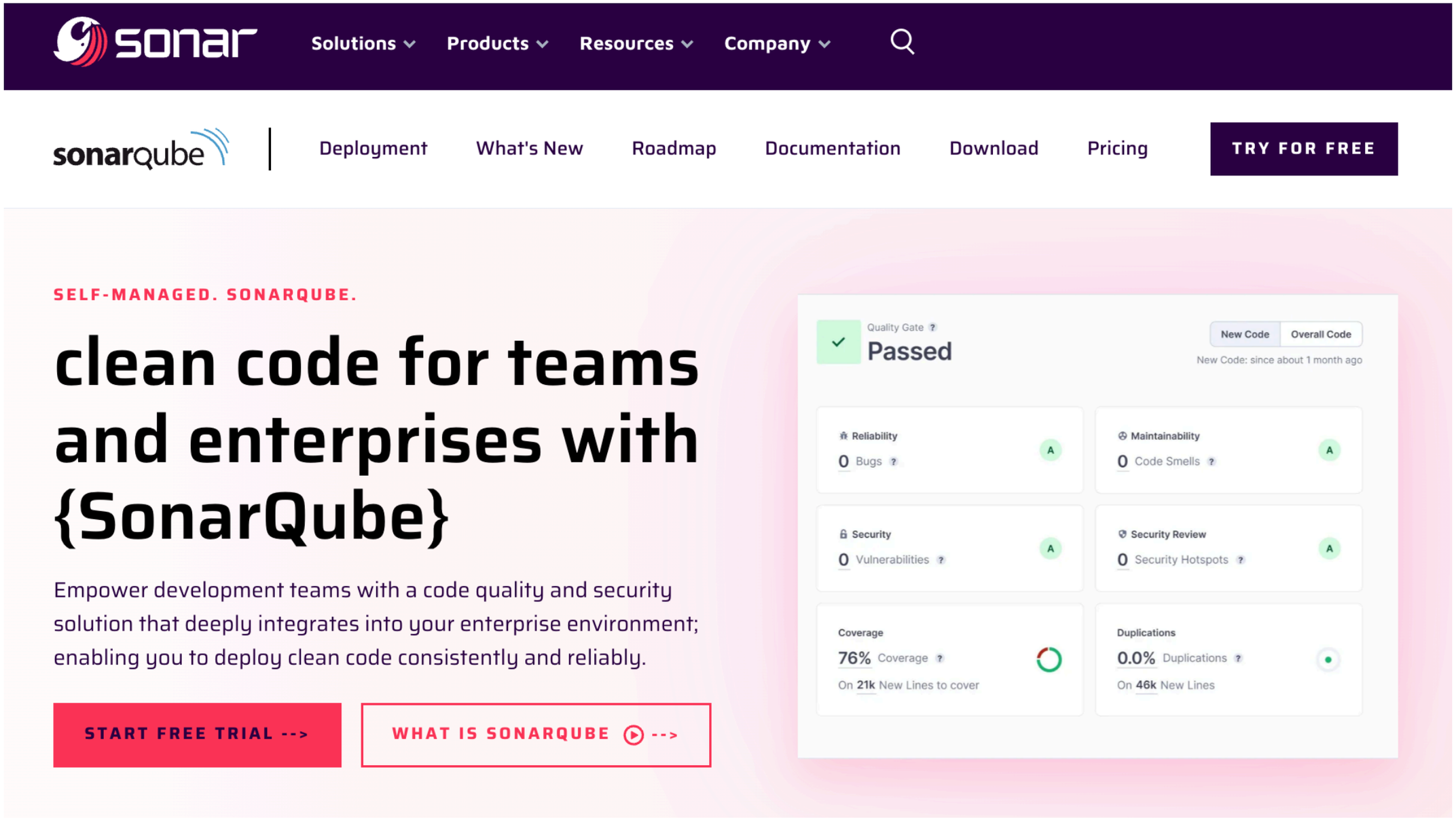
Static Application Security Testing (SAST)

Static Code Analysis (detect 50%)
Scan and review code
Identify source of vulnerabilities
White-box testing

Must be integrate into development process to help development team

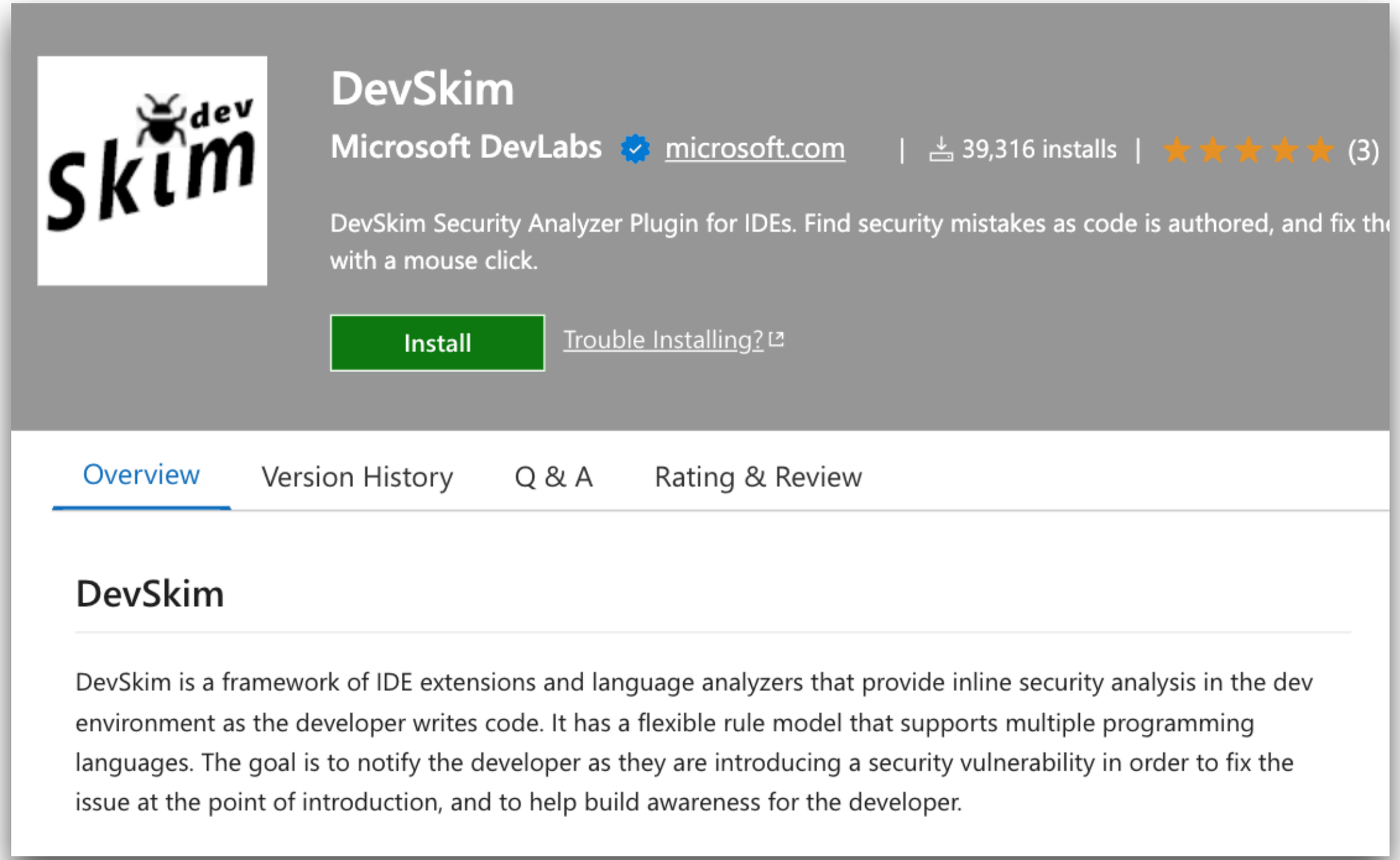
https://en.wikipedia.org/wiki/Static_application_security_testing

SonarQube



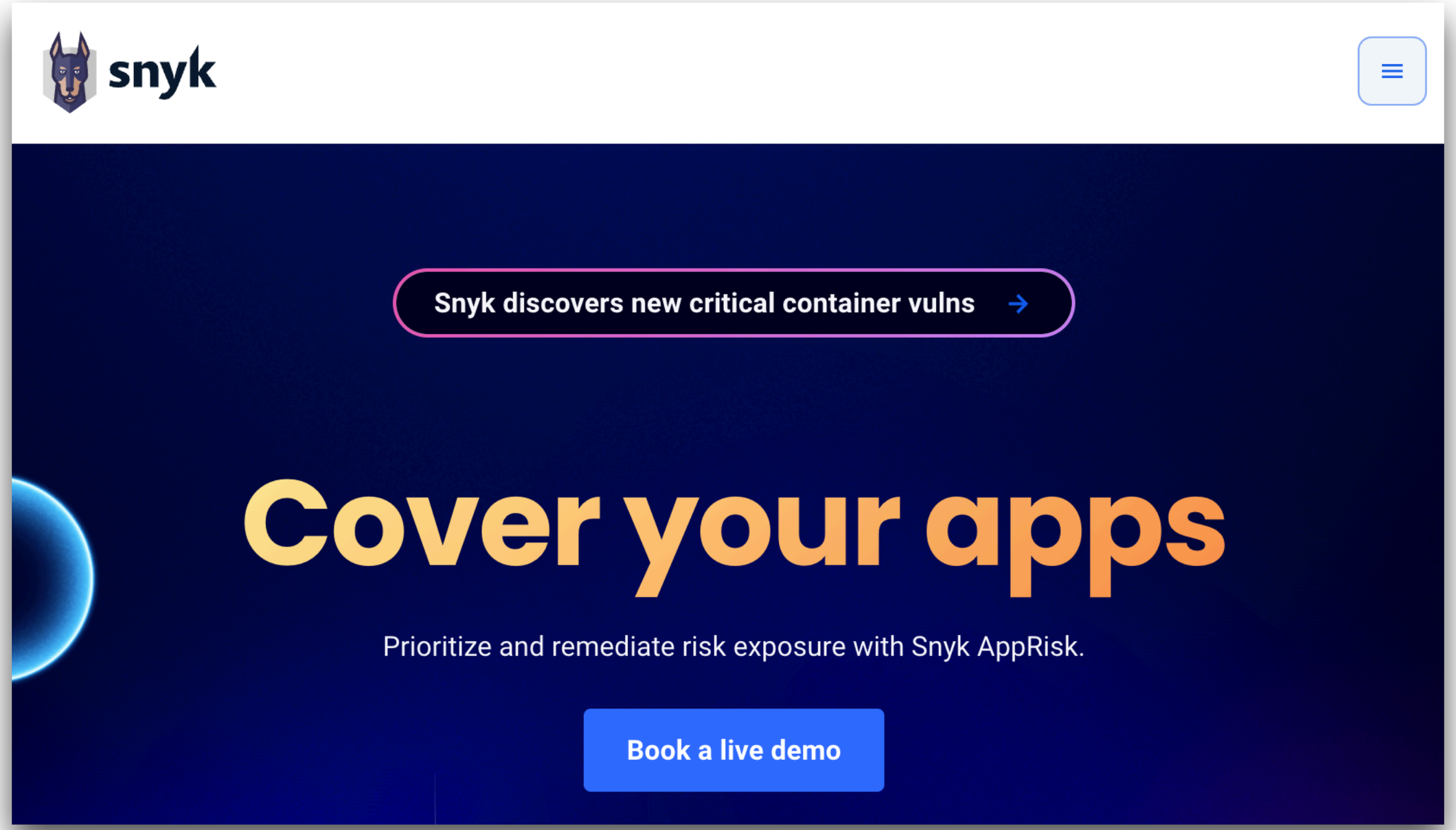
https://www.sonarsource.com/products/sonarqube/

DevSkim



https://marketplace.visualstudio.com/items?itemName=MS-CST-E.vscode-devskim

Snyk



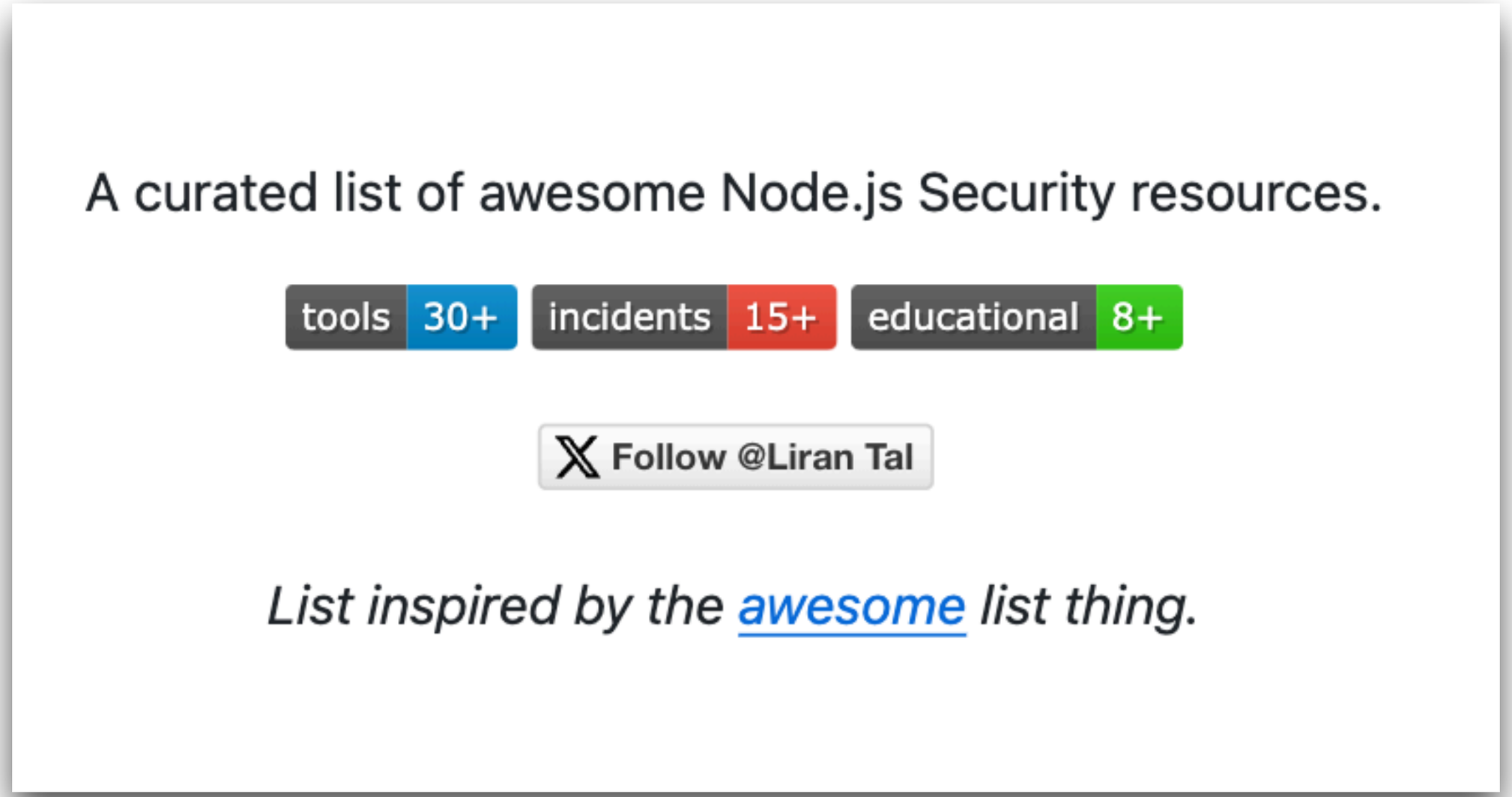
https://snyk.io/

More tools

Lint	Scan secret/ credential
ES/TS Lint	**Git-secrets**

https://owasp.org/www-community/Source_Code_Analysis_Tools

NodeJS



https://github.com/lirantal/awesome-nodejs-security

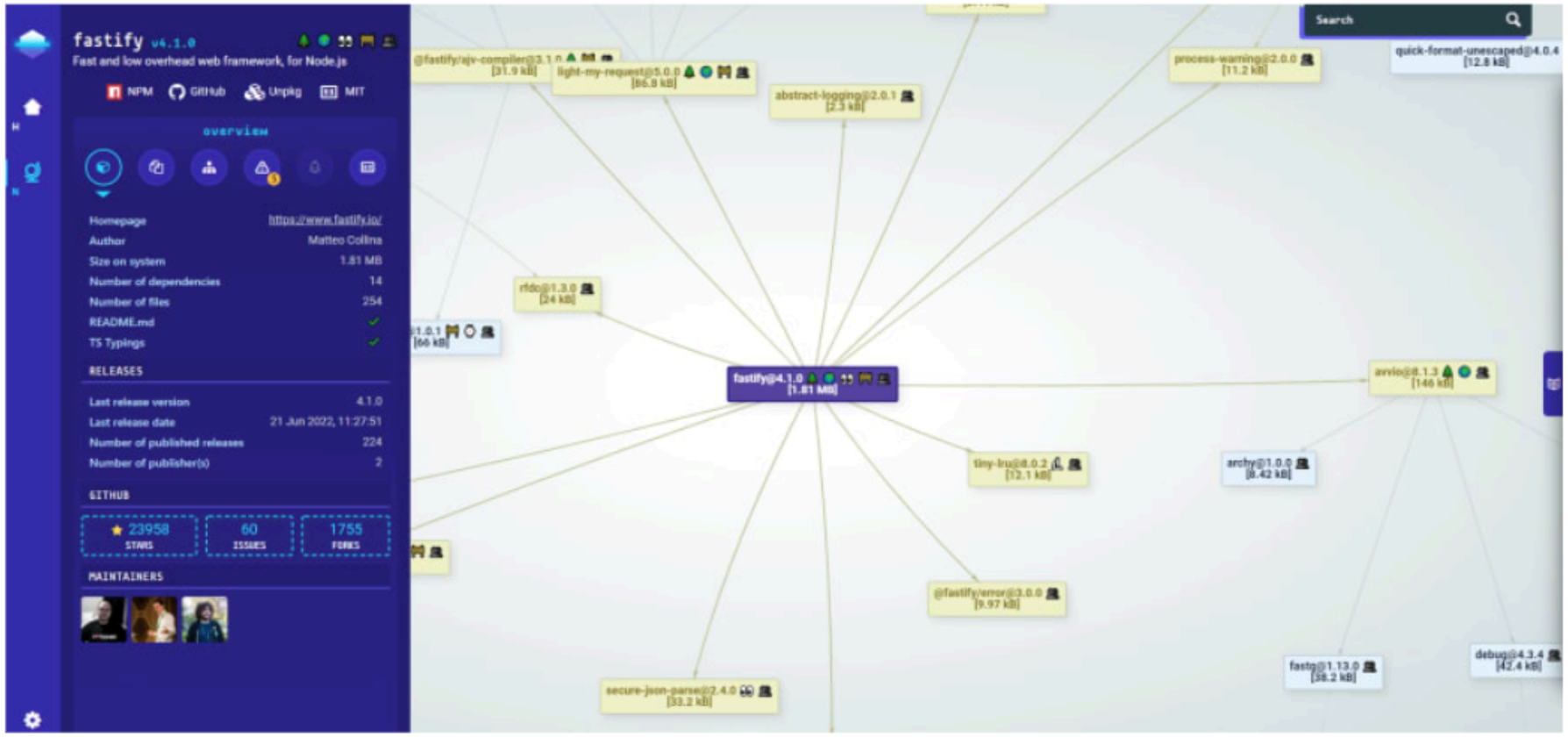
Node Secure CLI

🐢 Node-Secure CLI 🚀

a Node.js CLI to deeply analyze the dependency tree of a given NPM package or Node.js local app

VERSION V2.4.0 | LICENSE MIT | OPENSSF SCORECARD 8.8 | SLSA LEVEL 3 | BUILD PASSING

COVERAGE 100%

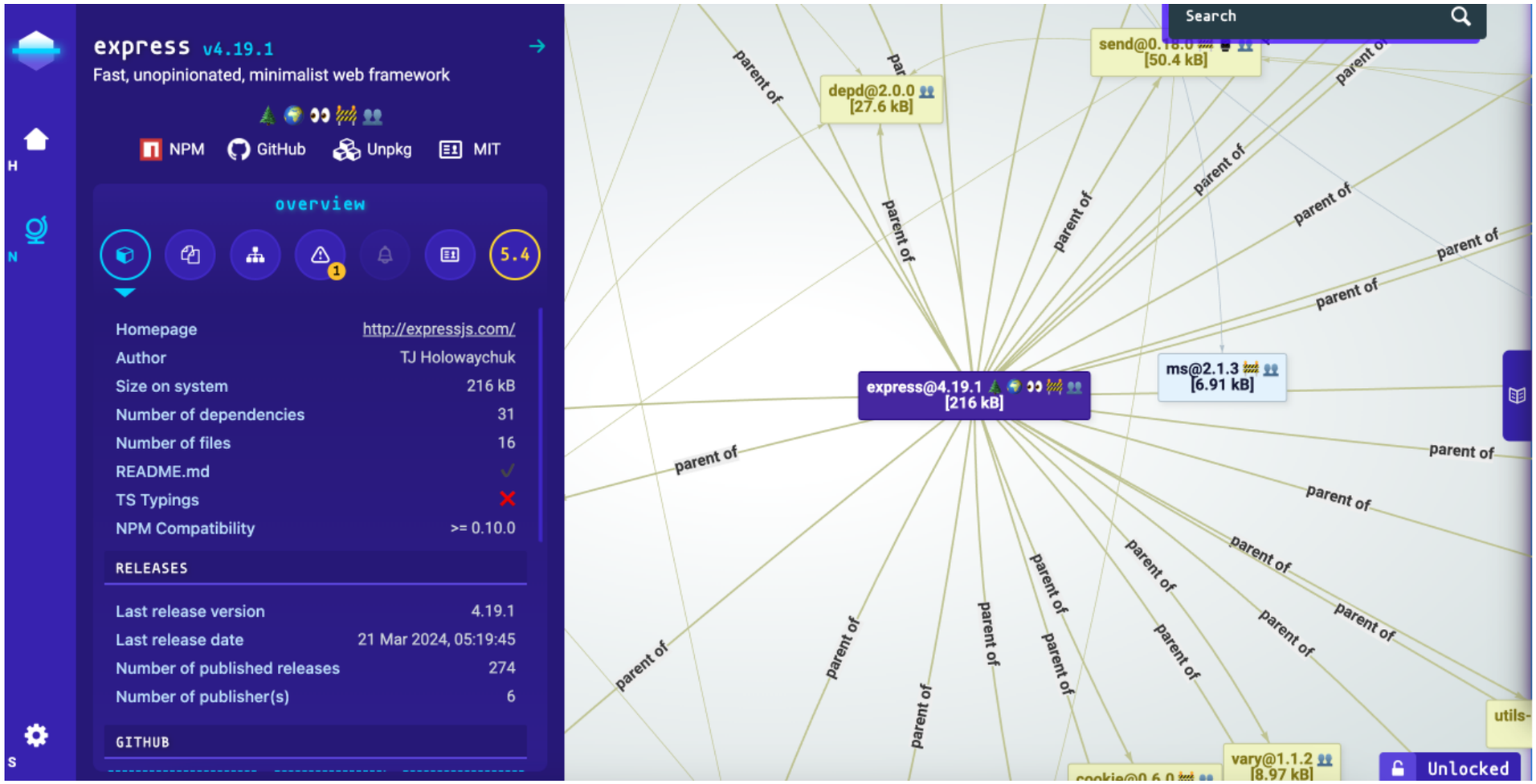


https://github.com/NodeSecure/cli

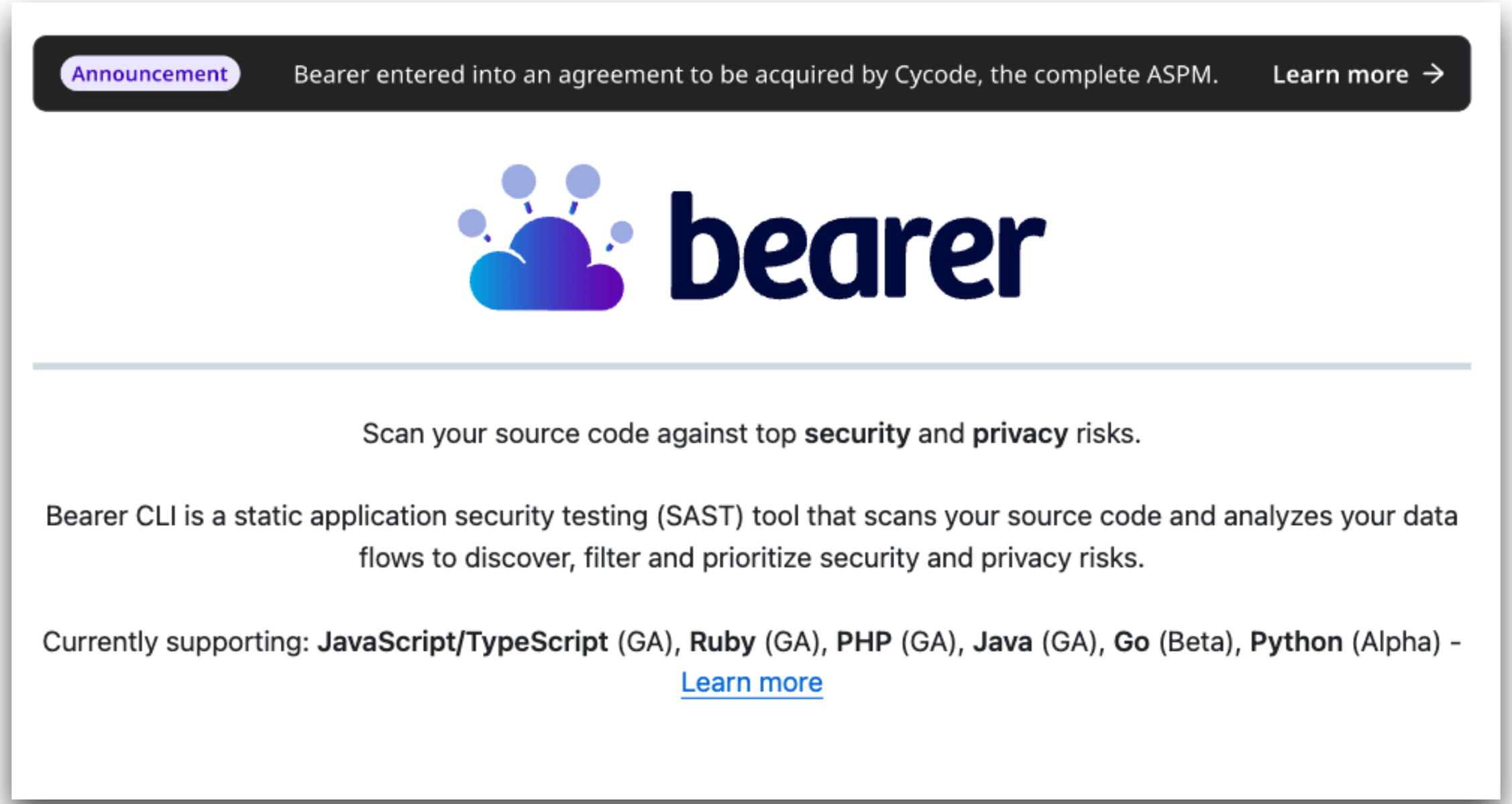
Node Secure CLI

$npm install -g @nodesecure/cli

$nsecure auto express



Bearer CLI



https://github.com/Bearer/bearer

Bearer CLI

Build-in rules in many languages
Support OWASP Top 10 and CWE Top 25
Privacy risks
Detect sensitive data

https://docs.bearer.com/reference/rules/

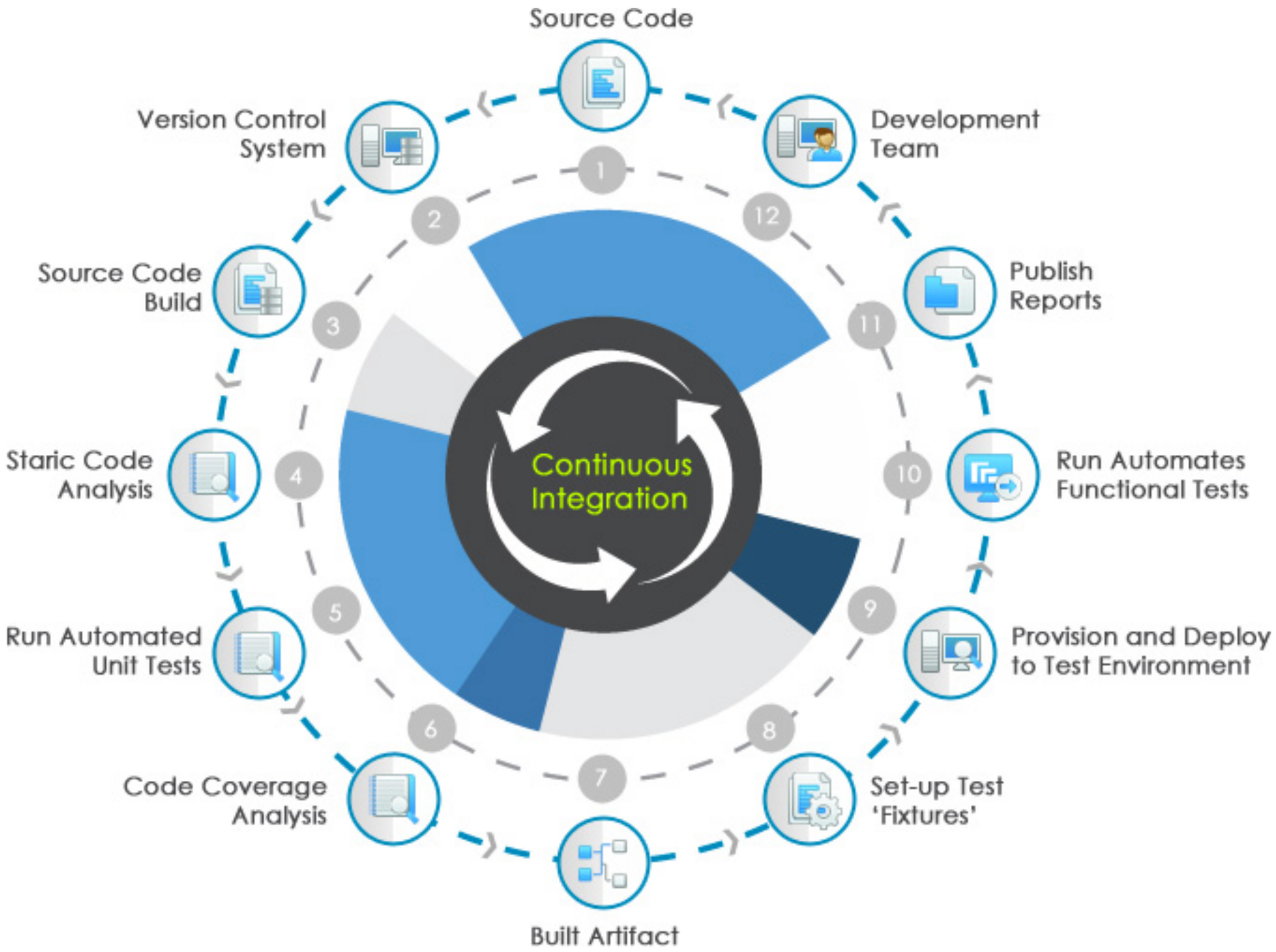
Security Pipeline

Security Pipeline

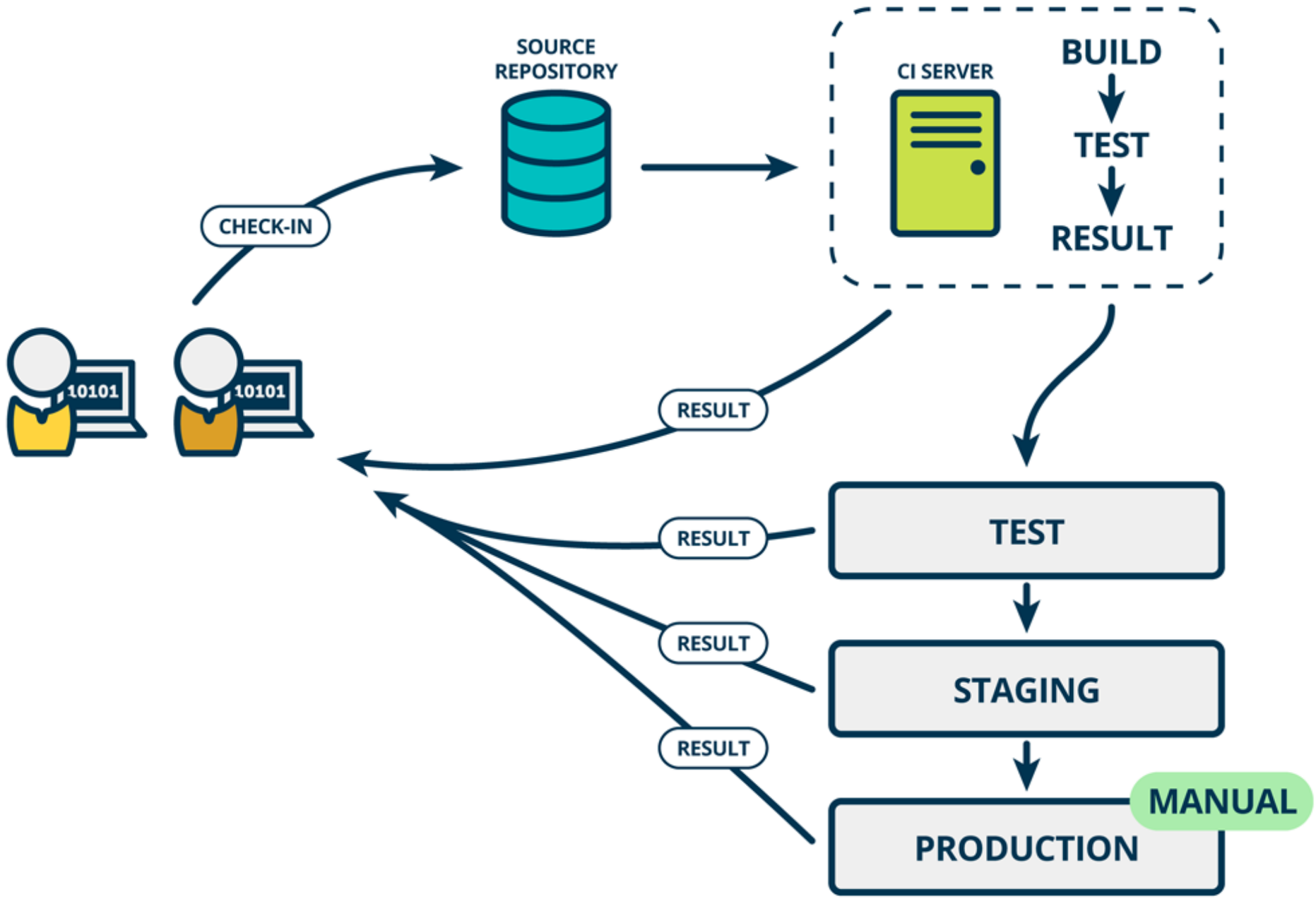
Implement security control in pipeline
Continuous Integration and Delivery

Build -> Test -> Deploy -> Monitoring

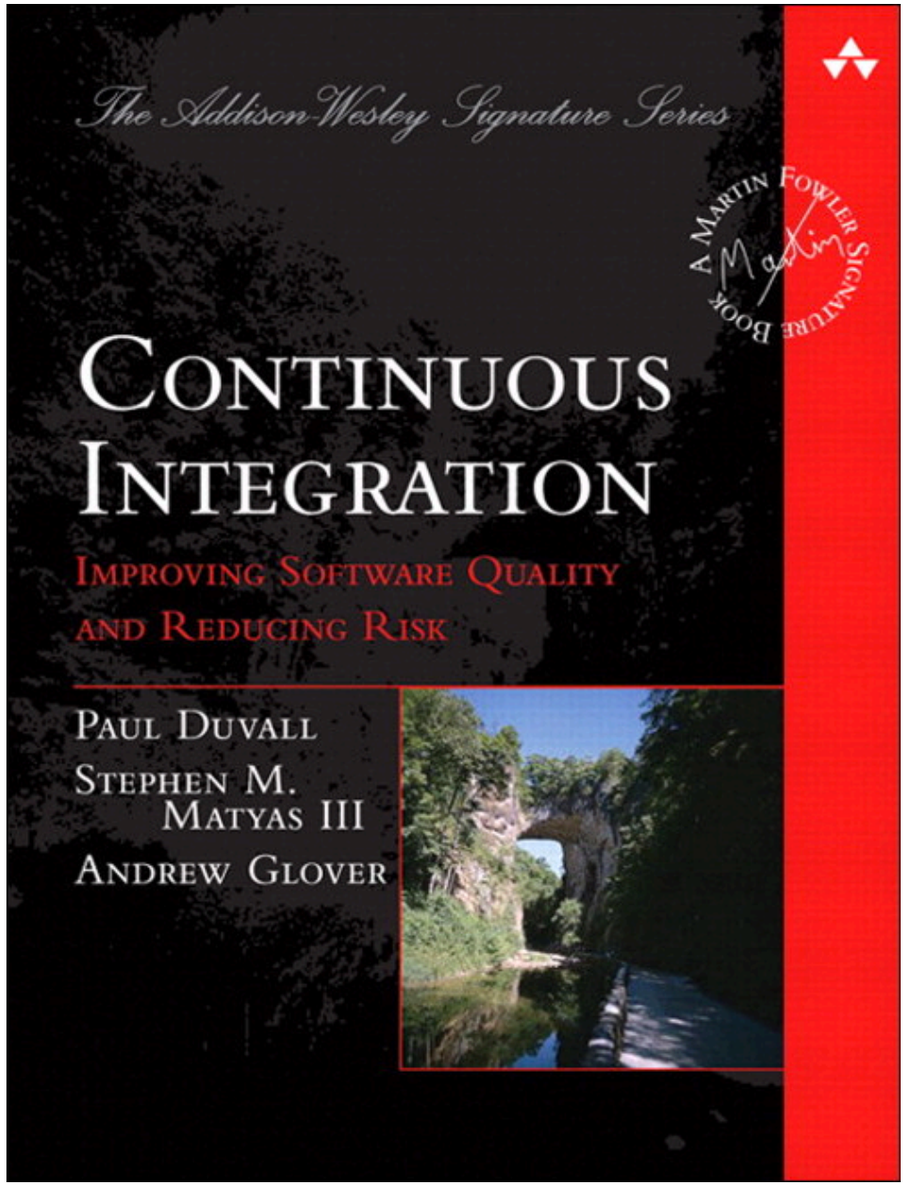
Continuos Integration

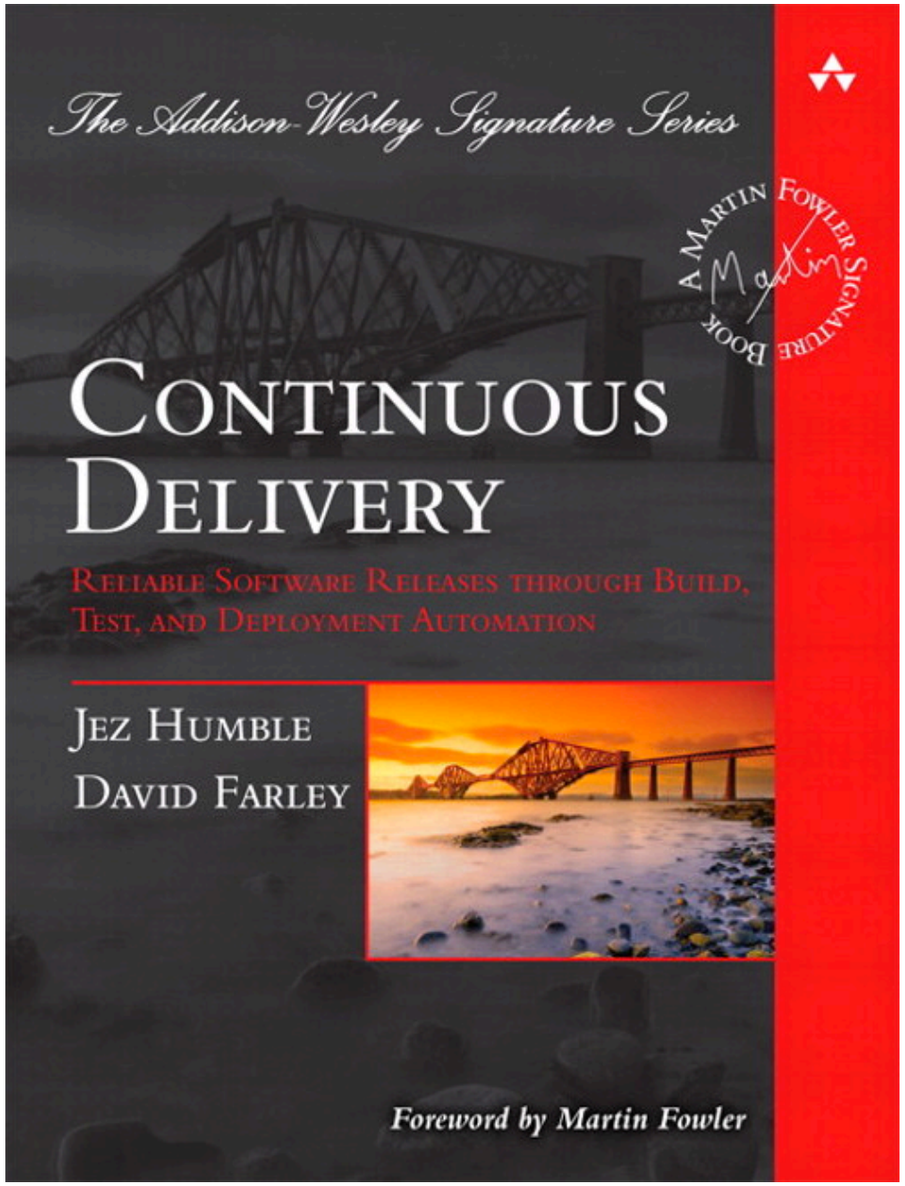


Continuous Delivery



Improve quality and reduce risk





Continuous Integration

Discipline to **integrate frequently**

Continuous Integration

Strive to make **small change**

Continuous Integration

Strive for **fast feedback**

Continuous Integration

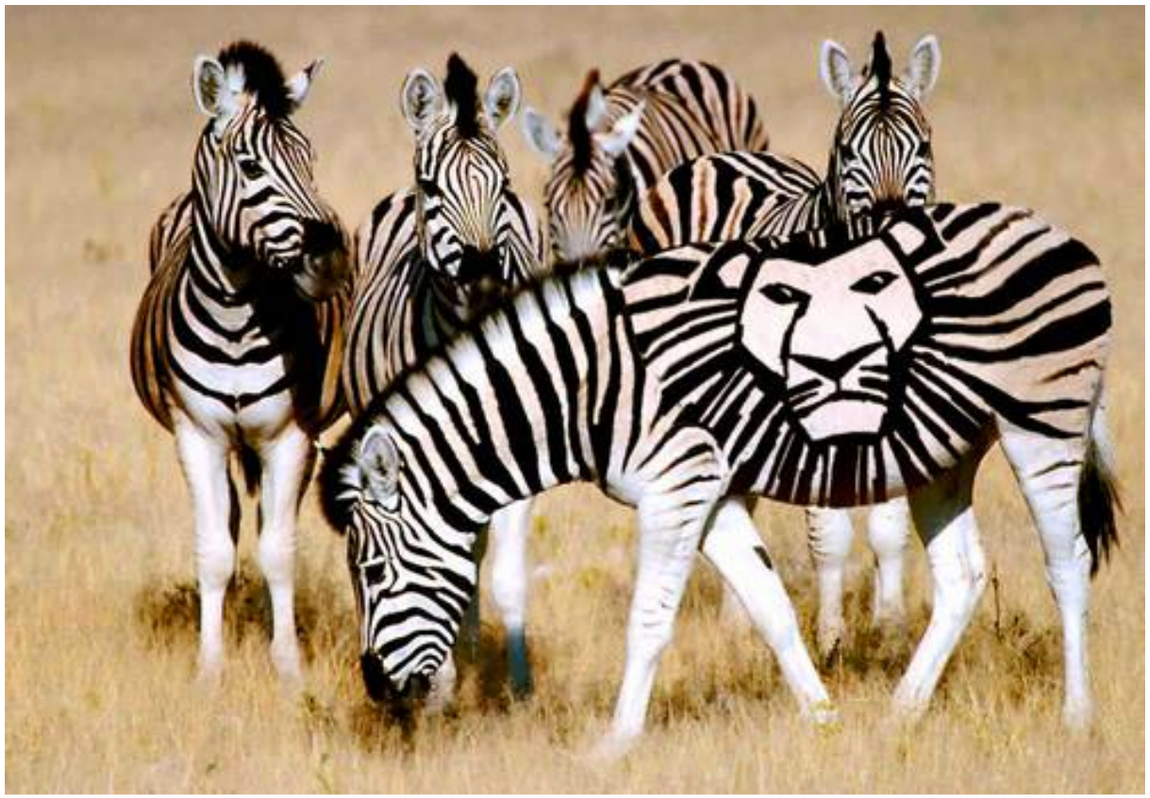
is a Software development **practices**

Workshop

Practice 1

Maintain a single source repository

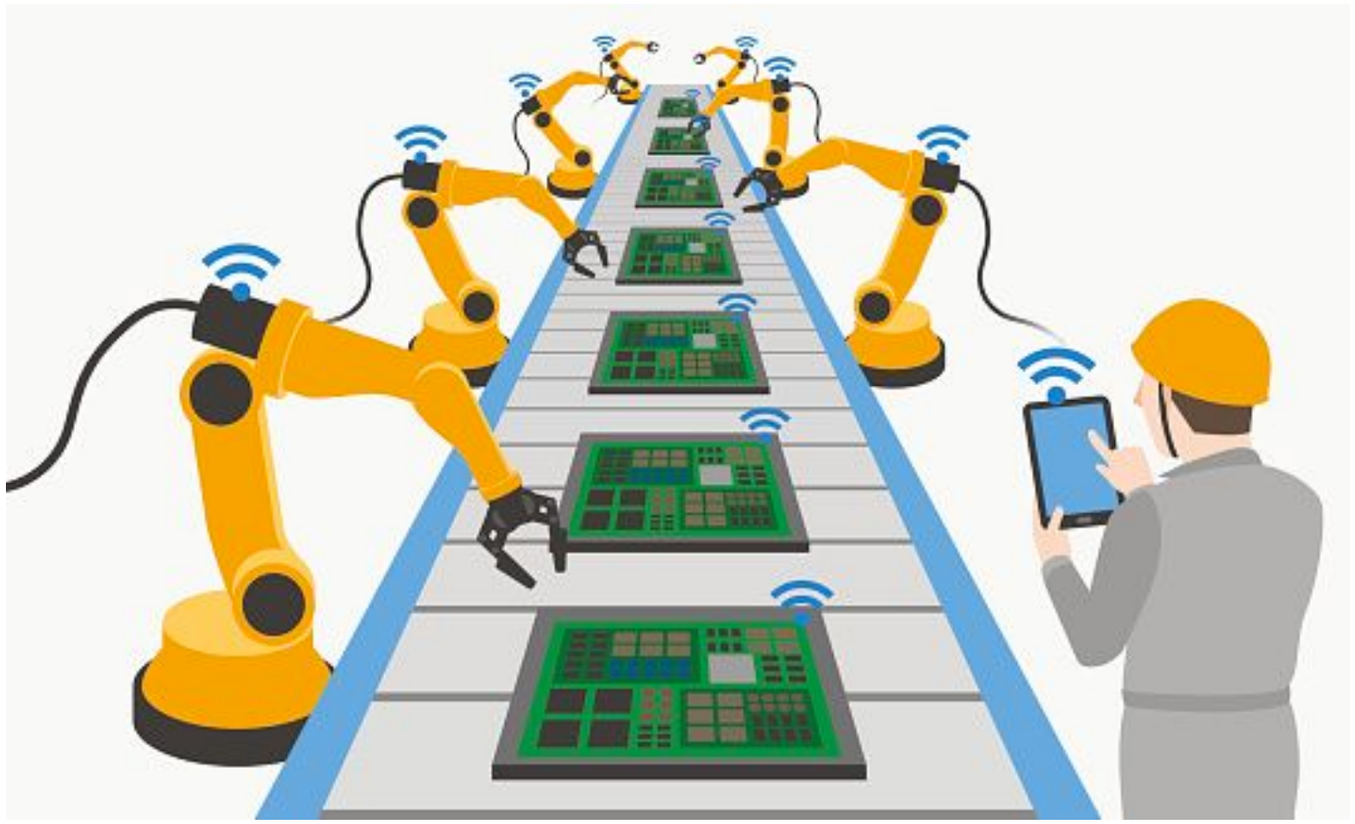
In general, you should store in source control everything you need to build anything



Practice 2

Automated the build

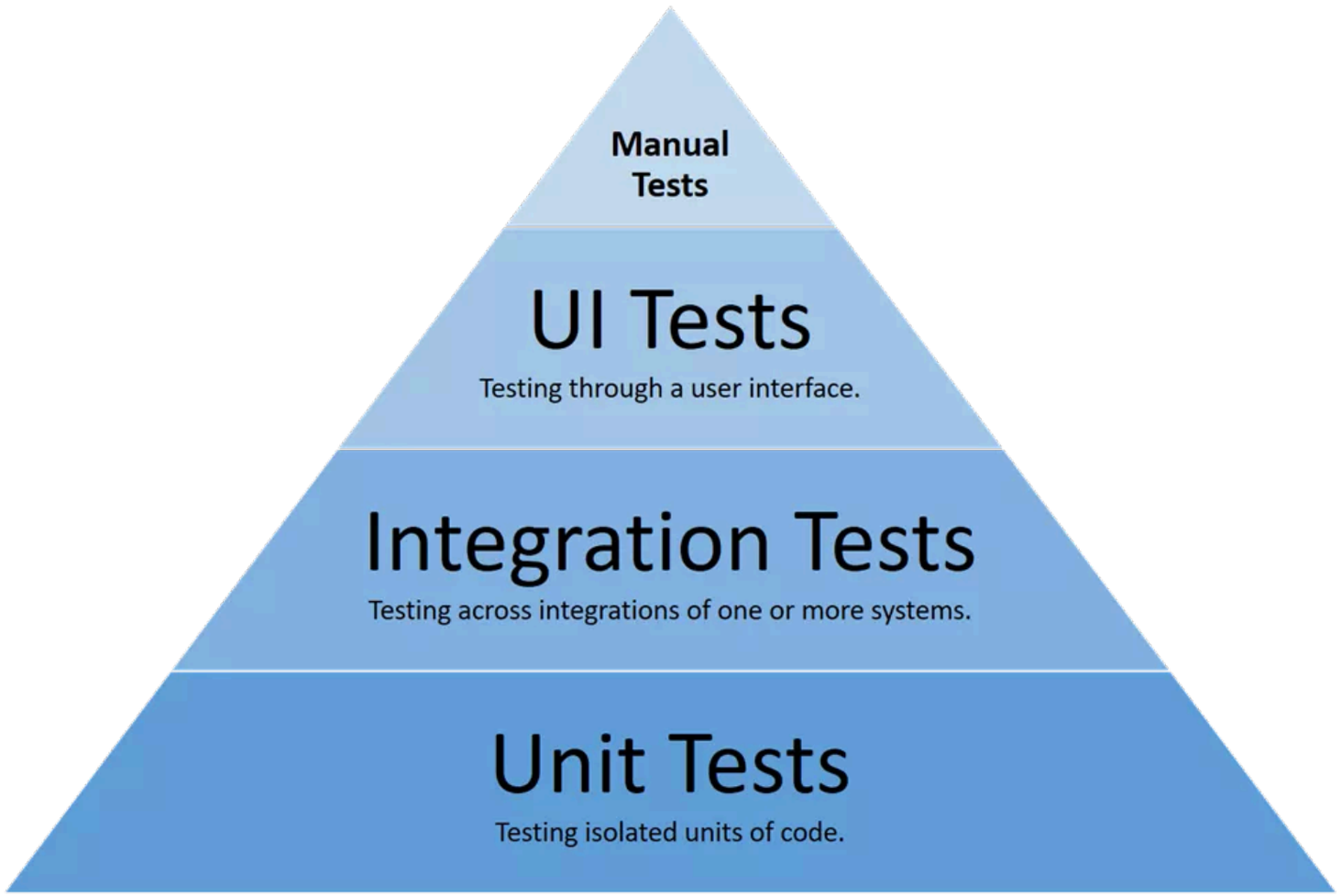
Automated environment for builds



Practice 3

Make your build **self-testing**

Build process => compile, linking and **testing**

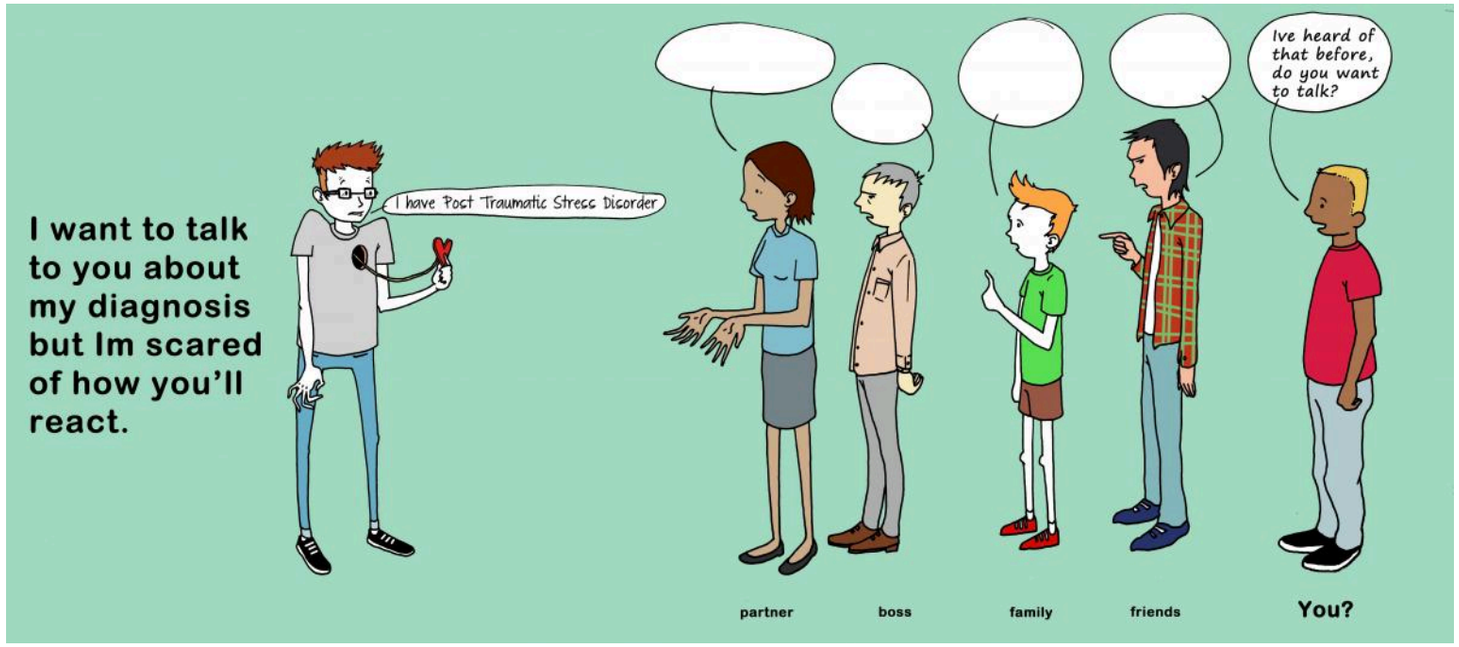


Practice 4

Everyone **commits** to the mainline everyday

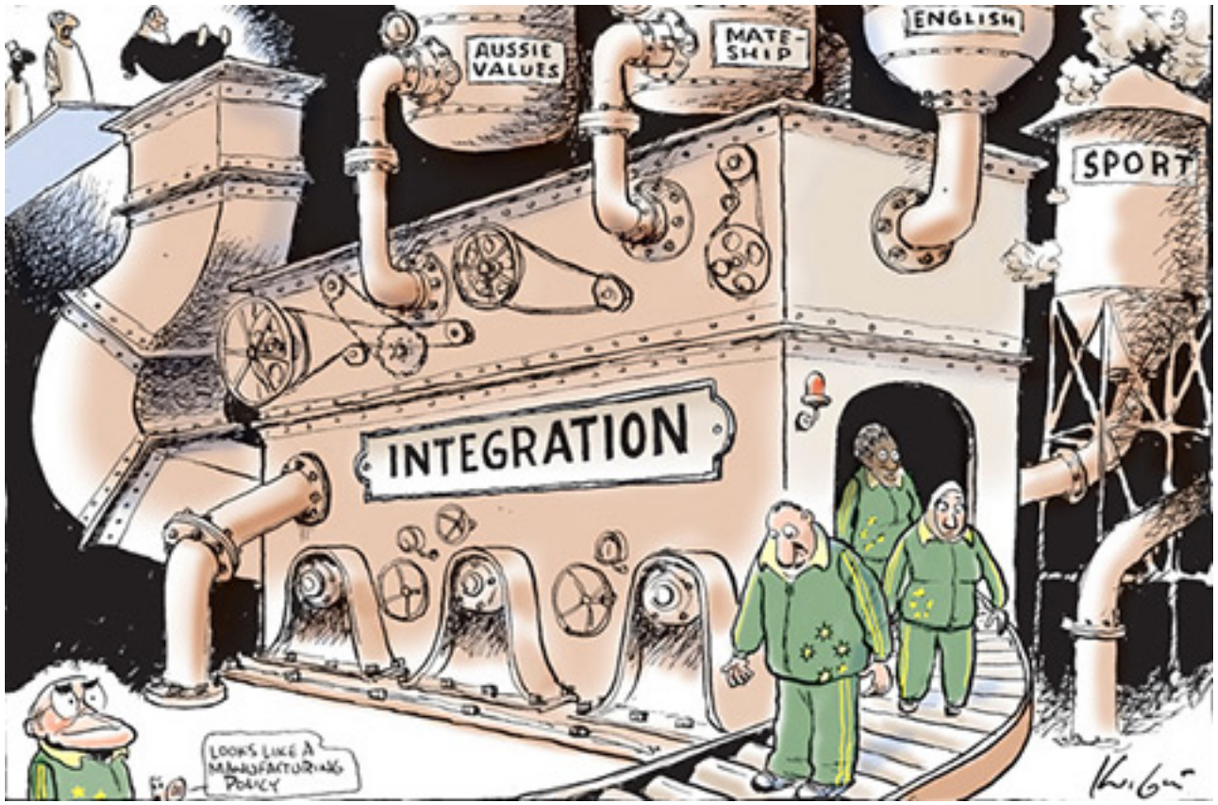
Integration is about **communication**

Integration allows developers to **tell** other developers



Practice 5

Every commits should build the mainline on an **Integration machine**



Practice 6

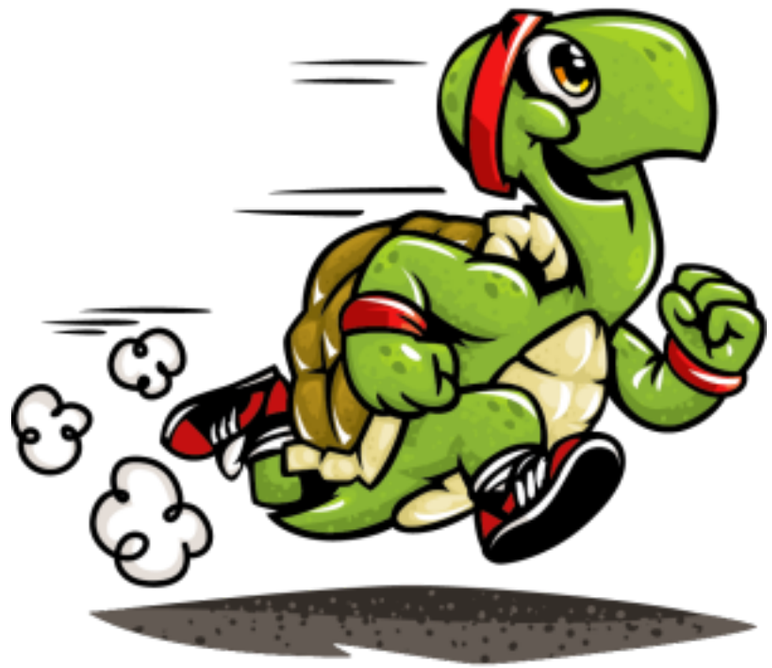
Fix broken builds **immediately**

“Nobody has a higher priority task than fixing the build”

Practice 7

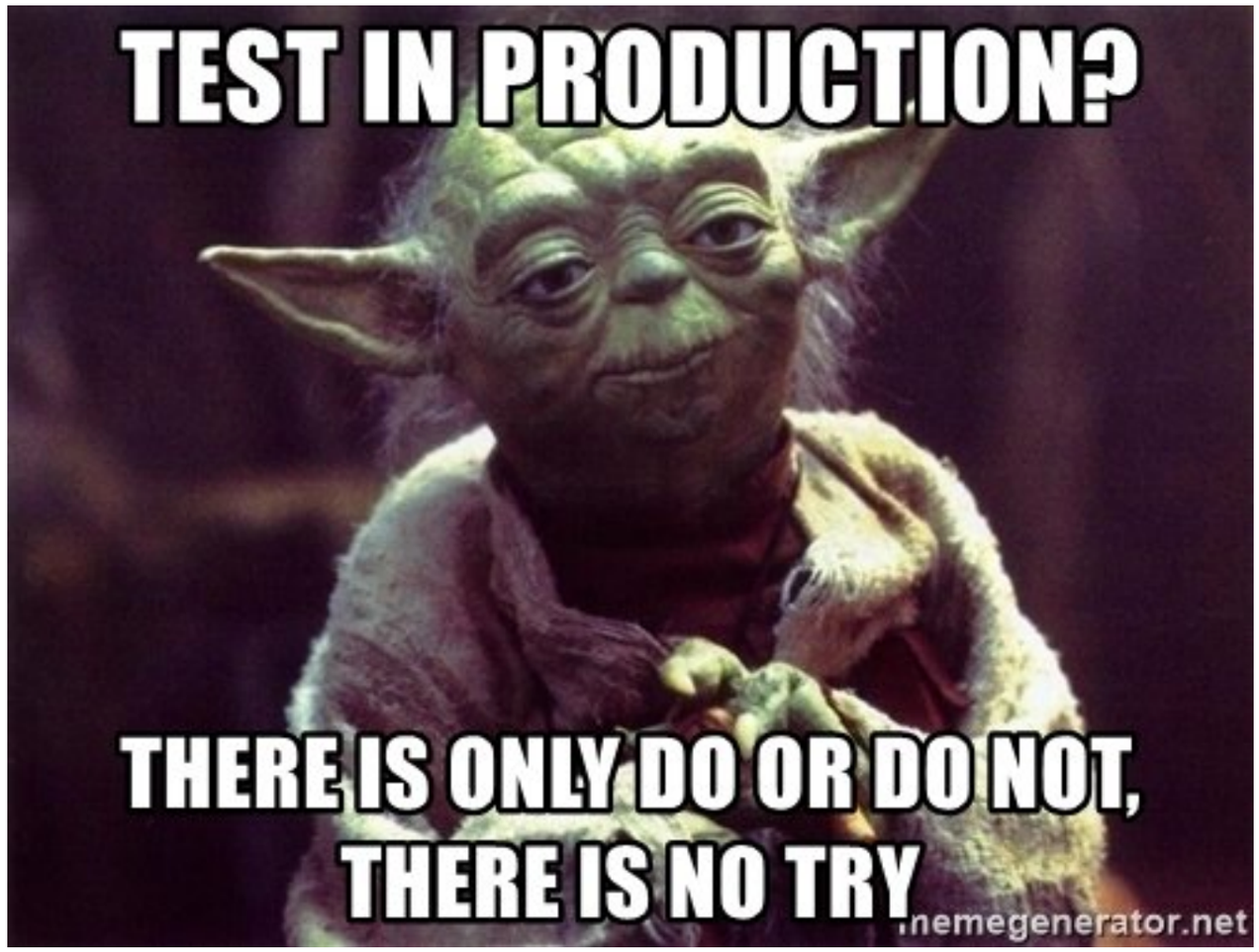
Keep the build **fast**

Continuous Integration is to provide rapid feedback



Practice 8

Test in clone of the **Production** environment



Practice 9

Make it easy for anyone to get
the latest executable

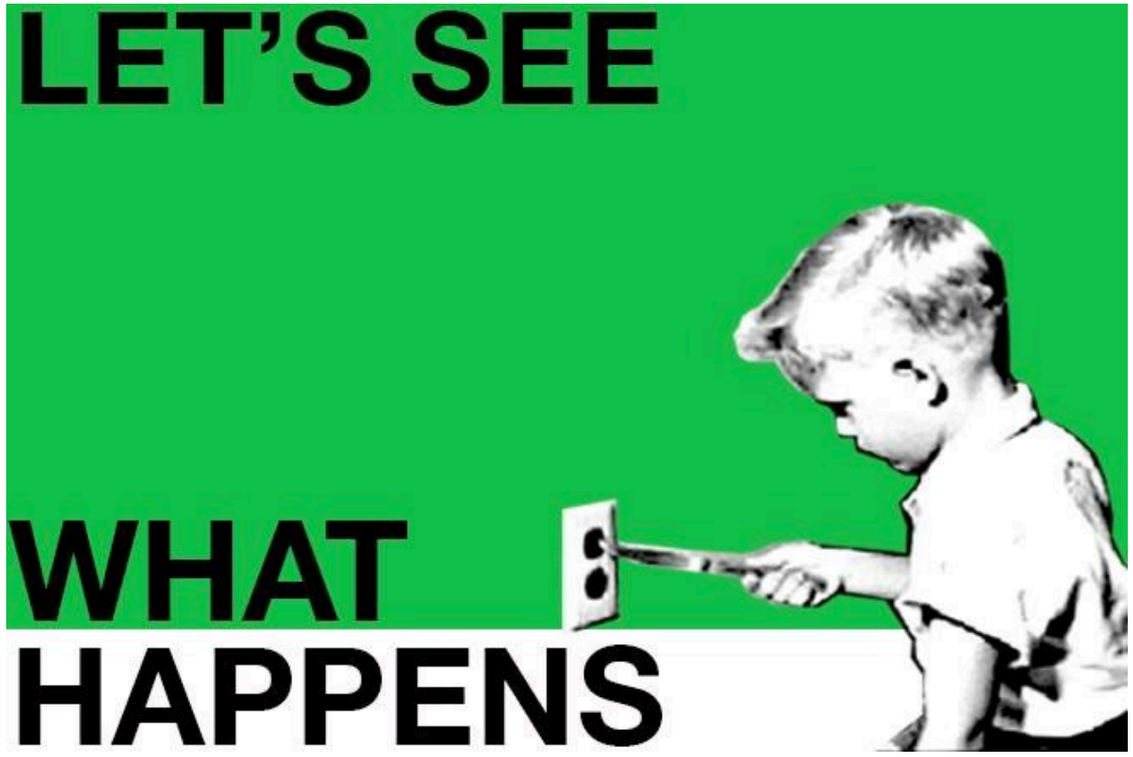
Make sure well known place where people can find

Practice 10

Everyone can see what's happening

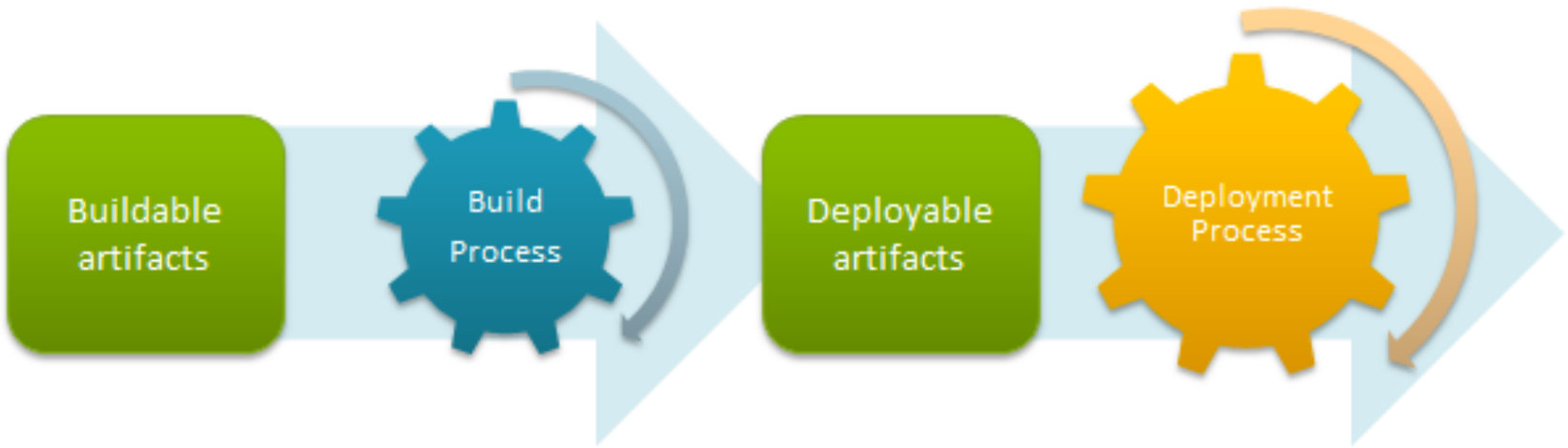
Easier to see the state of the system and changes

Show the good information

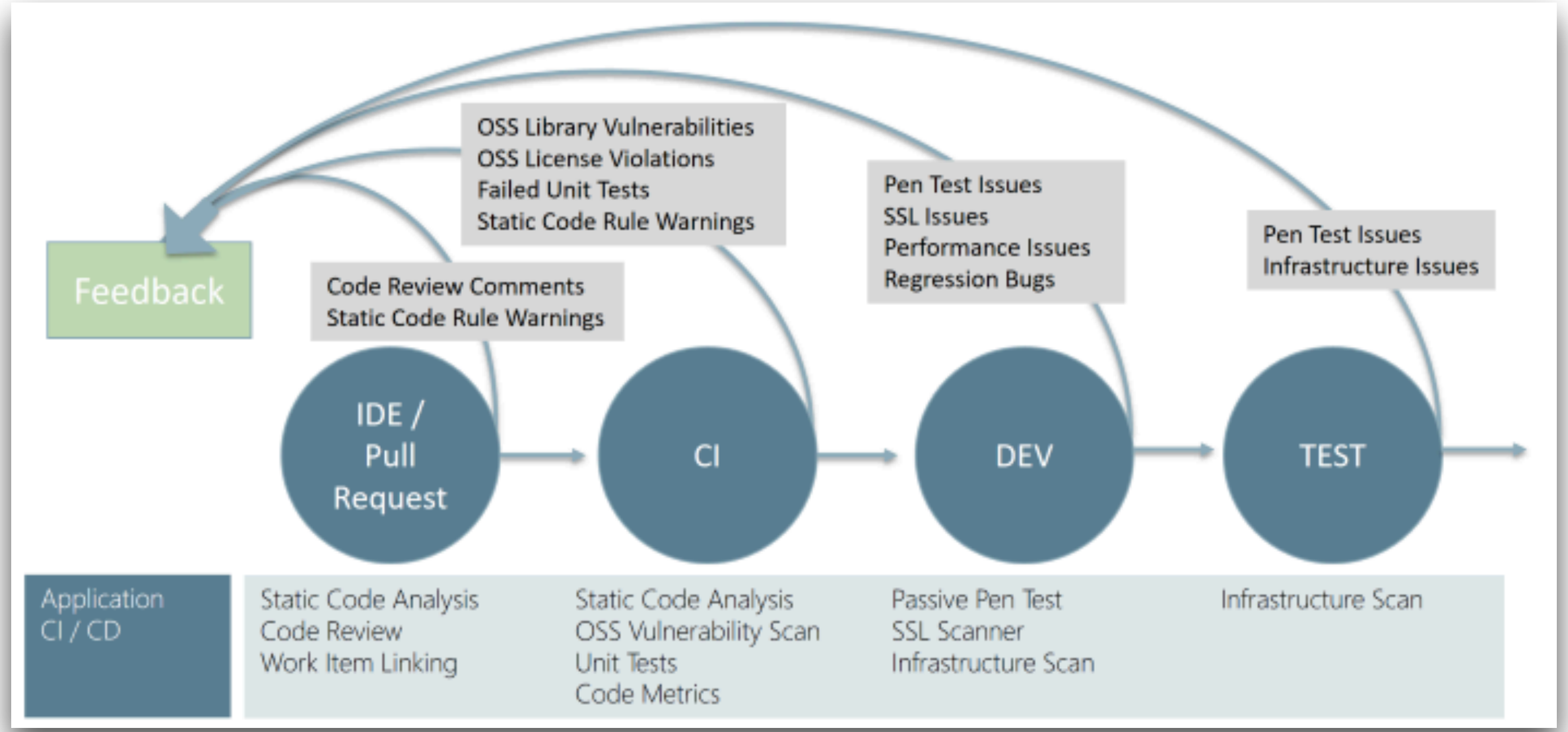


Practice 11

Automated deployment



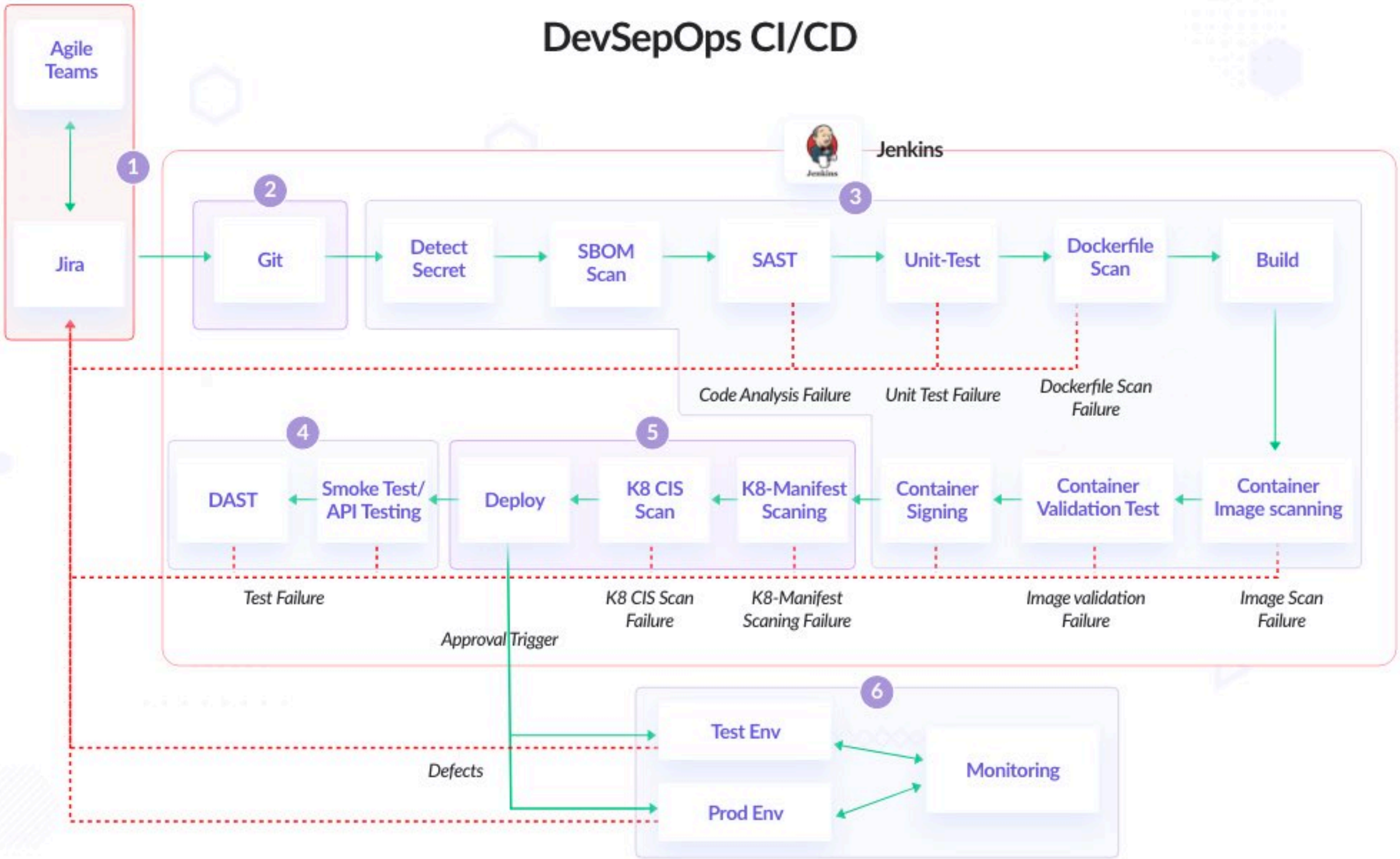
All about feedback loop



Workshop

Design your delivery process

DevSepOps CI/CD



Workshop implement your pipeline

Application and framework to manage and monitor the executable of **repeated tasks**

Jenkins

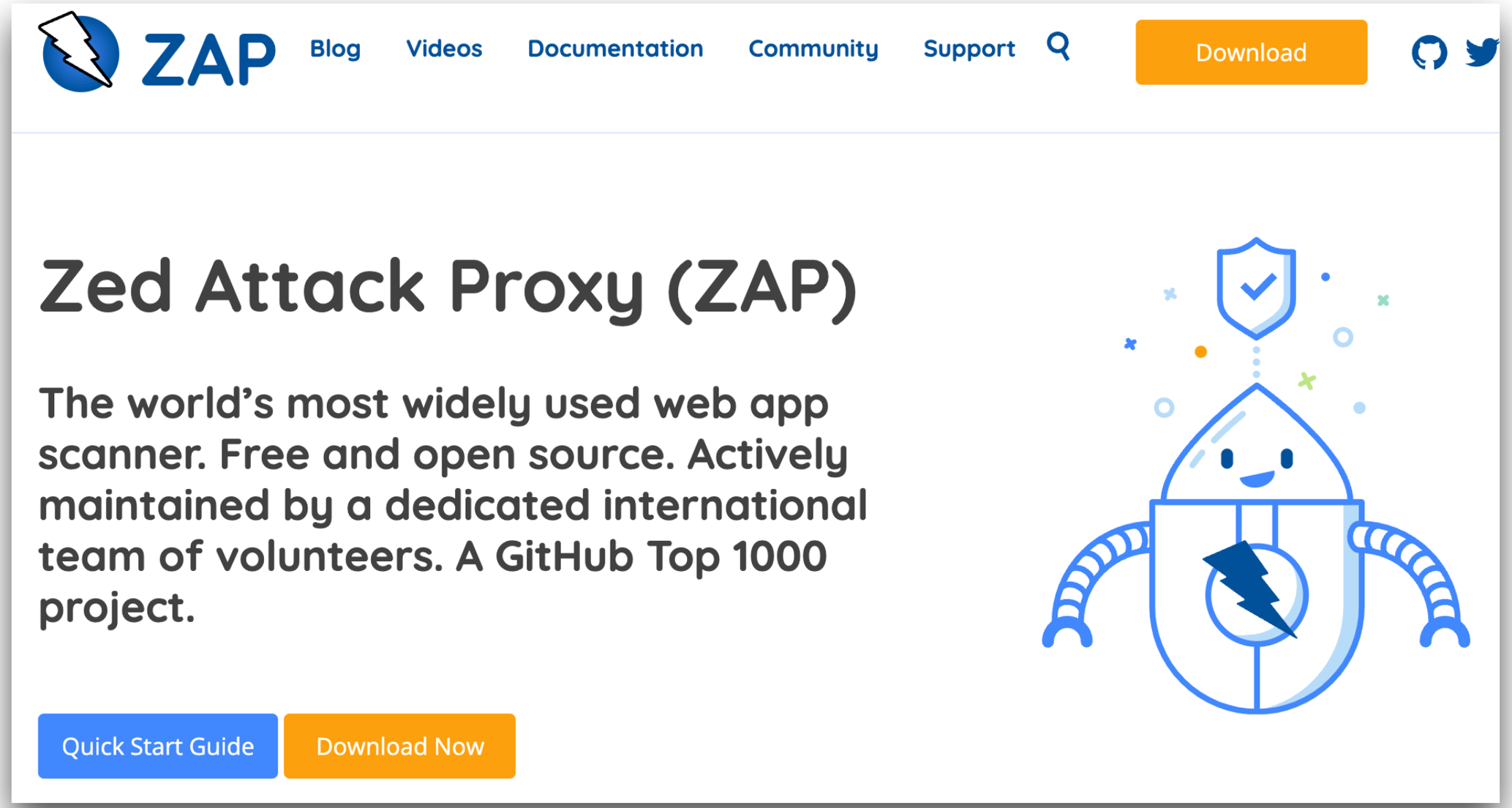
https://jenkins.io/

Dynamic Application Security Testing (DAST)

Black-box testing
Non-functional testing
Interact with application (frontend or backend)

https://en.wikipedia.org/wiki/Dynamic_application_security_testing

Zed Attack Proxy (ZAP)



https://www.zaproxy.org/

Q/A