CCSP

CCSP EXAM CRAM

EXAM PREPARATION SERIES

2023 EDITION

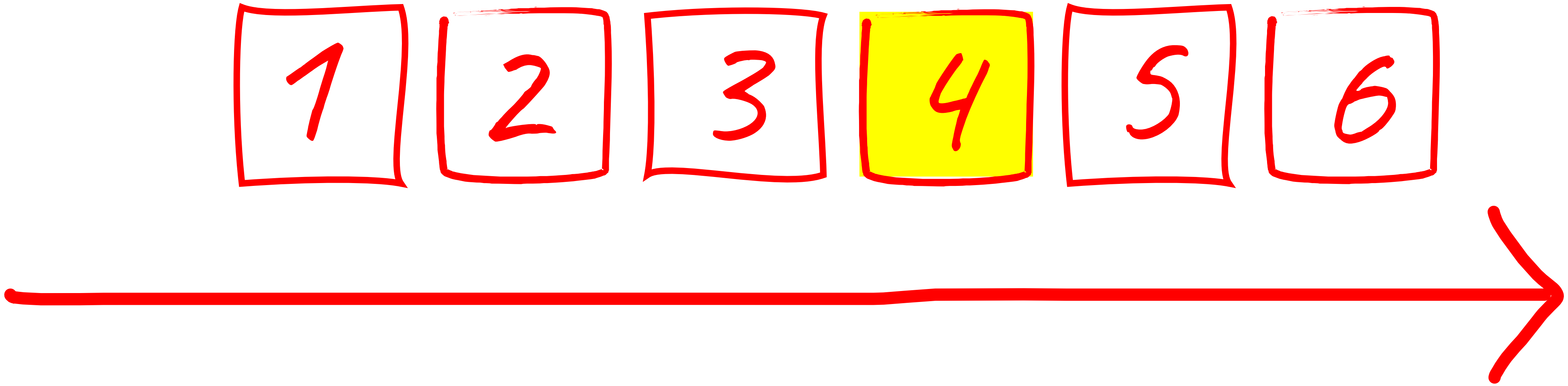
DOMAIN 4

Coverage of every topic in the official exam syllabus!

with **Pete Zerger** vCISO, CISSP, MVP

INTRODUCTION: SERIES OVERVIEW

LESSONS IN THIS SERIES



One lesson for each exam domain

...and a consolidated full course video when the series is complete

EXAM OBJECTIVES (DOMAINS)

DOMAIN	WEIGHT
1. Cloud Concepts, Architecture, and Design	17%
2. Cloud Data Security	20%
3. Cloud Platform and Infrastructure Security	17%
4. Cloud Application Security	**17%**
5. Cloud Security Operations	16%
6. Legal, Risk, and Compliance	13%

Domain 4 is the focus of this video

(ISC)²
CCSP
OFFICIAL
(ISC)²
CCSP
Certified Cloud Security Professional
Official Study Guide
Third Edition
SYBEX

CCSP

CSSP EXAM CRAM

THE COMPLETE COURSE

DOMAIN 4

Cloud Application Security

INSIDE CLOUD
AND SECURITY

EXAM ESSENTIALS - 4

Cloud development basics, pitfalls, vulnerabilities

Ensure performance, scalability, portability, interoperability, OWASP Top 10, SANS Top 25.

Application of Software Development Lifecycle (SDLC)

Development models (Agile, Waterfall), threat models (STRIDE, PASTA, DREAD), secure coding practices and standards.

Apply testing methodologies to application software

Functional and nonfunctional testing, static and dynamic testing, QA process in SDLC.

Manage software supply chain and secure software usage

Supply chain security, vendor assessment, API security practices, open-source vs third party.

Common application security technology, security controls

Design elements, data encryption in motion and at rest, orchestration, virtualization, tooling.

IAM solutions, common threats to identity and access

Federated identity, SSO, MFA, secrets management, user/privileged/service access.

4. CLOUD APPLICATION SECURITY

4.1 Advocate Training and Awareness for Application Security

Cloud Development Basics

Common Pitfalls

Common Cloud Vulnerabilities
(e.g., Open Web Application Security Project (OWASP) Top-10, SANS Top-25)

CLOUD DEVELOPMENT BASICS

Security by design

Declares security should be present throughout every step of the process.

Various models exist to help, like the **Building Security In Maturity Model (BSIMM)**.

Pairs well with DevSecOps

Shared security responsibility

The idea is that security is the responsibility of everyone from the most junior member of the team to senior management.

Describes the primary principle of DevSecOps

Security as a business objective

Risk mitigation through security controls should be a key business objective, similar to customer satisfaction or revenue.

Requires org-wide security awareness and commitment

COMMON PITFALLS

Common pitfalls of application security in the cloud

- ✓ Performance
- ✓ Scalability
- ✓ Interoperability
- ✓ Portability
- ✓ API security

Know the common pitfalls AND advantages of avoiding each!

COMMON PITFALLS

Performance

Cloud software development often relies on loosely coupled services.

Makes designing for and meeting performance goals more complex, as multiple components may interact in unexpected ways

Verify through end-to-end load and stress testing

Scalability

One of the key features of the cloud is the ability to scale allowing applications and services to grow and shrink as demand fluctuates.

Requires developers to think about how to retain state across instances and handle faults with individual servers

Scale out is better than scale up in the cloud

COMMON PITFALLS

Interoperability

is the ability to work across platforms, services, or systems and can be very important, especially multi-vendor and multi-cloud scenarios.

Interoperability across platforms increases service provider choice and can reduce costs

Portability

Designing software that can move between on premises and cloud environments or between cloud providers makes it **portable**.

Portability in a hybrid scenario requires avoiding use of certain environment and provider-specific APIs and tools.

The additional effort can make it harder to leverage some cloud advantages, and may require compromises

COMMON PITFALLS

API Security

Application programming interfaces (APIs), are relied on throughout cloud application design, development, and operation.

Designing APIs to work well with cloud architectures while remaining secure are both common challenges for developers and architects.

API security considerations

- ✓ Access control
- ✓ Data encryption
- ✓ Throttling
- ✓ Rate limiting

CSPs offer PaaS services that simplify addressing these concerns

COMMON CLOUD VULNERABILITIES

Several groups provide guidance on common application vulnerabilities and related security threats.

- ✓ Data breaches
- ✓ Data integrity
- ✓ Insecure application programming interfaces (APIs)
- ✓ Denial-of-Service

VULNERABILITIES

Common cloud vulnerabilities to avoid with SSDLC include

- ✓ Cloud Security Alliance (CSA)
- ✓ SANS Institute
- ✓ Open Web Application Security Project (OWASP)

ORGANIZATIONS

There are several that provide information on security threats,

4. CLOUD APPLICATION SECURITY

4.2 Describe the Secure Software Development Life Cycle (SDLC) Process

Business Requirements

Phases and Methodologies
(e.g., design, code, test, maintain, waterfall vs. agile)

BUSINESS REQUIREMENTS

Mature software development shops utilize an SDLC because it saves money and supports repeatable, quality software development.

SSDLC is fully successful only if the integration of security into an organization's existing SDLC is required for all development efforts.

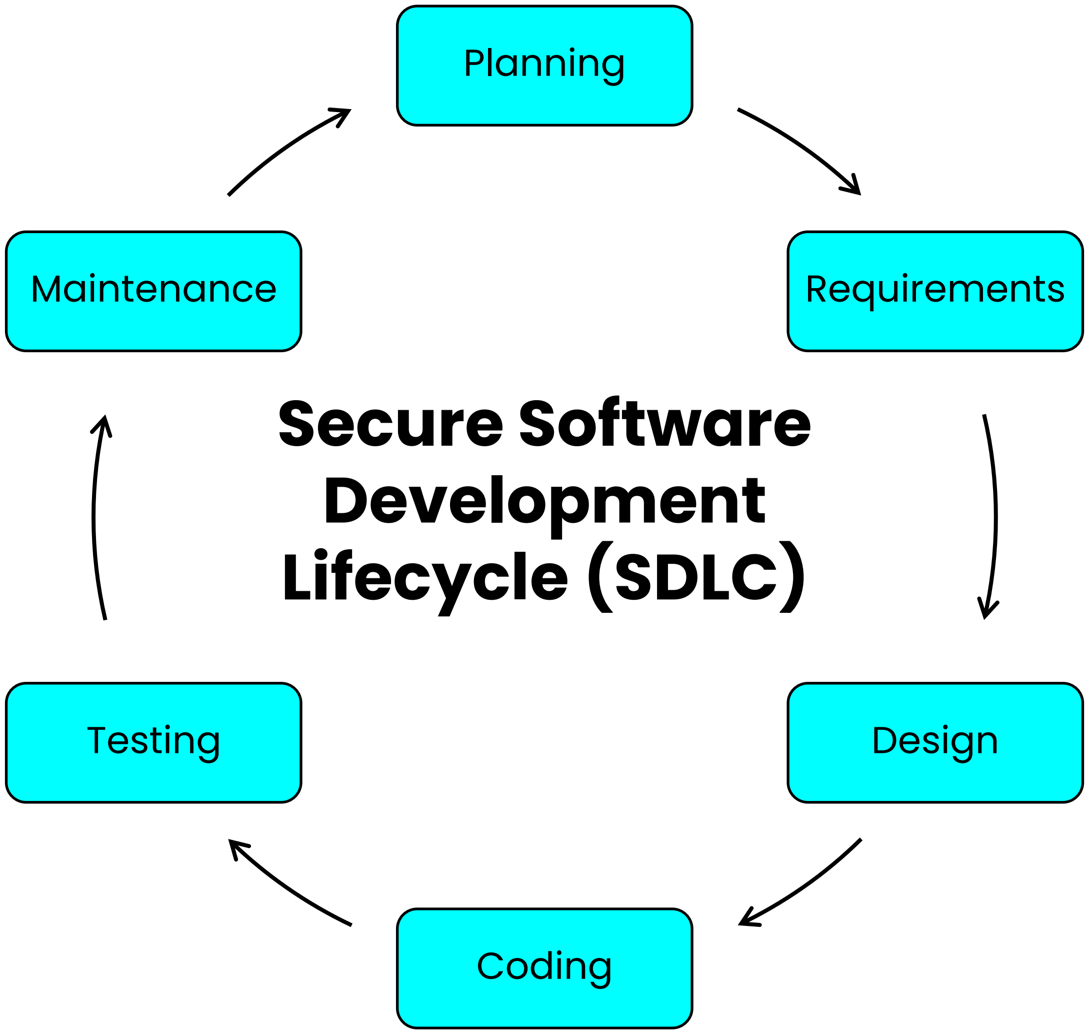
Business requirements capture what the organization needs its information systems to do.

Functional requirements detail what the solution must do, such as supporting up as max concurrent user requirements…

…which in turn support business requirements, like all workers being able to access a system to perform their assigned duties.

In addition to these functional requirements, the organization must also consider **security**, **privacy**, and **compliance** objectives

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)



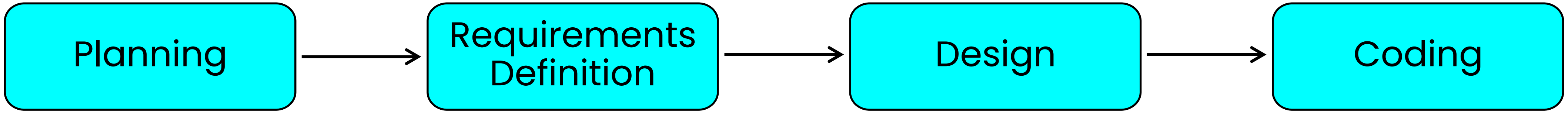
There are multiple variations of the secure SDLC

These are the common phases of the secure SDLC

Including the SSDLC acronym in the CBK

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

Regardless of which SDLC model a company uses, there are a few phases that appear in all the models:

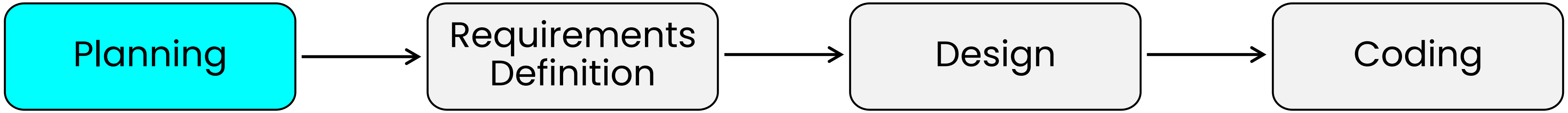


Planning → Requirements Definition → Design → Coding

Mentioned in the OSG, so ensure you are familiar!

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

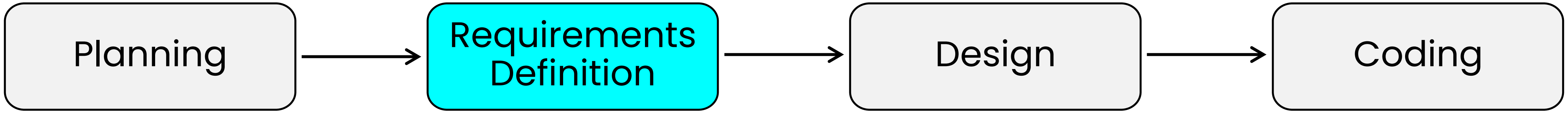
Regardless of which SDLC model a company uses, there are a few phases that appear in all the models:



Considers potential development work, focusing on determining need, feasibility, and cost.

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

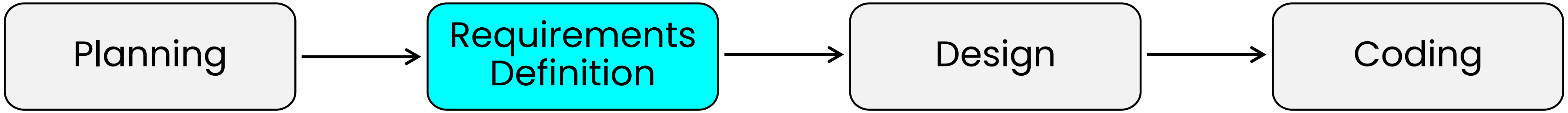
Regardless of which SDLC model a company uses, there are a few phases that appear in all the models:



Once an effort has been deemed feasible, user and business functionality requirements are captured.

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

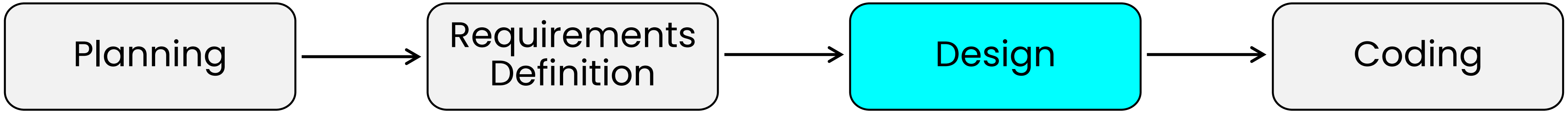
Regardless of which SDLC model a company uses, there are a few phases that appear in all the models:



Involves user, customer and stakeholder input to determine desired functionality, current system or app functionality, and desired improvements.

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

Regardless of which SDLC model a company uses, there are a few phases that appear in all the models:

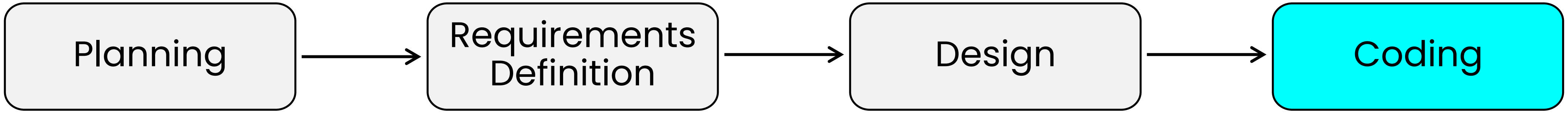


Design functionality, architecture, integration points and techniques, data flows, and business processes.

Solution is designed based on requirements gathered

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

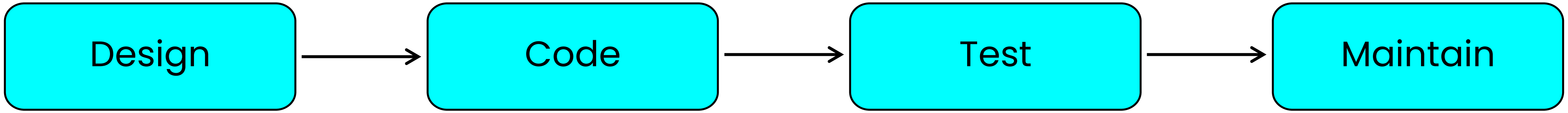
Regardless of which SDLC model a company uses, there are a few phases that appear in all the models:



Where the actual coding (work) happens

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

The CCSP exam outline mentions four phases: **design**, **code, test**, and **maintain**:

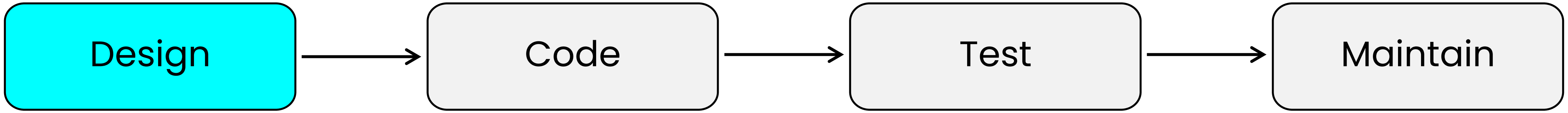


Design → Code → Test → Maintain

Mentioned in the OSG, so ensure you are familiar!

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

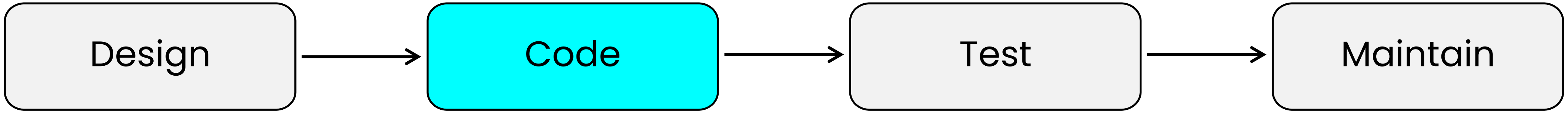
The CCSP exam outline mentions four phases: **design**, **code, test**, and **maintain**:



Solution is designed based on requirements gathered

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

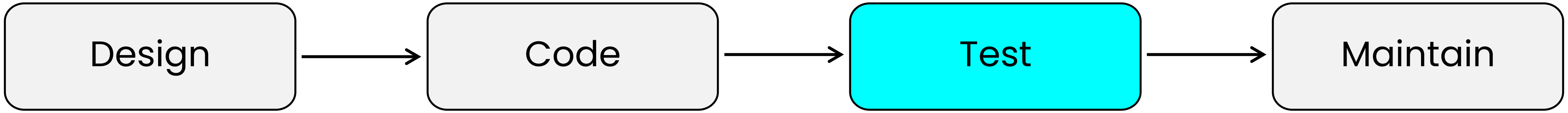
The CCSP exam outline mentions four phases: **design**, **code, test**, and **maintain**:



Where the actual coding (work) happens

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

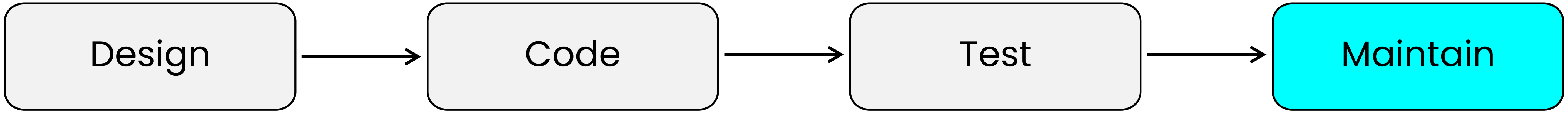
The CCSP exam outline mentions four phases: **design**, **code, test**, and **maintain**:



Testing to ensure software is functional, scalable, and secure

SECURE SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

The CCSP exam outline mentions four phases: **design**, **code, test**, and **maintain**:



Ongoing maintenance updates, patching, and checks to ensure software remains functional and secure

SOFTWARE DEVELOPMENT MODELS

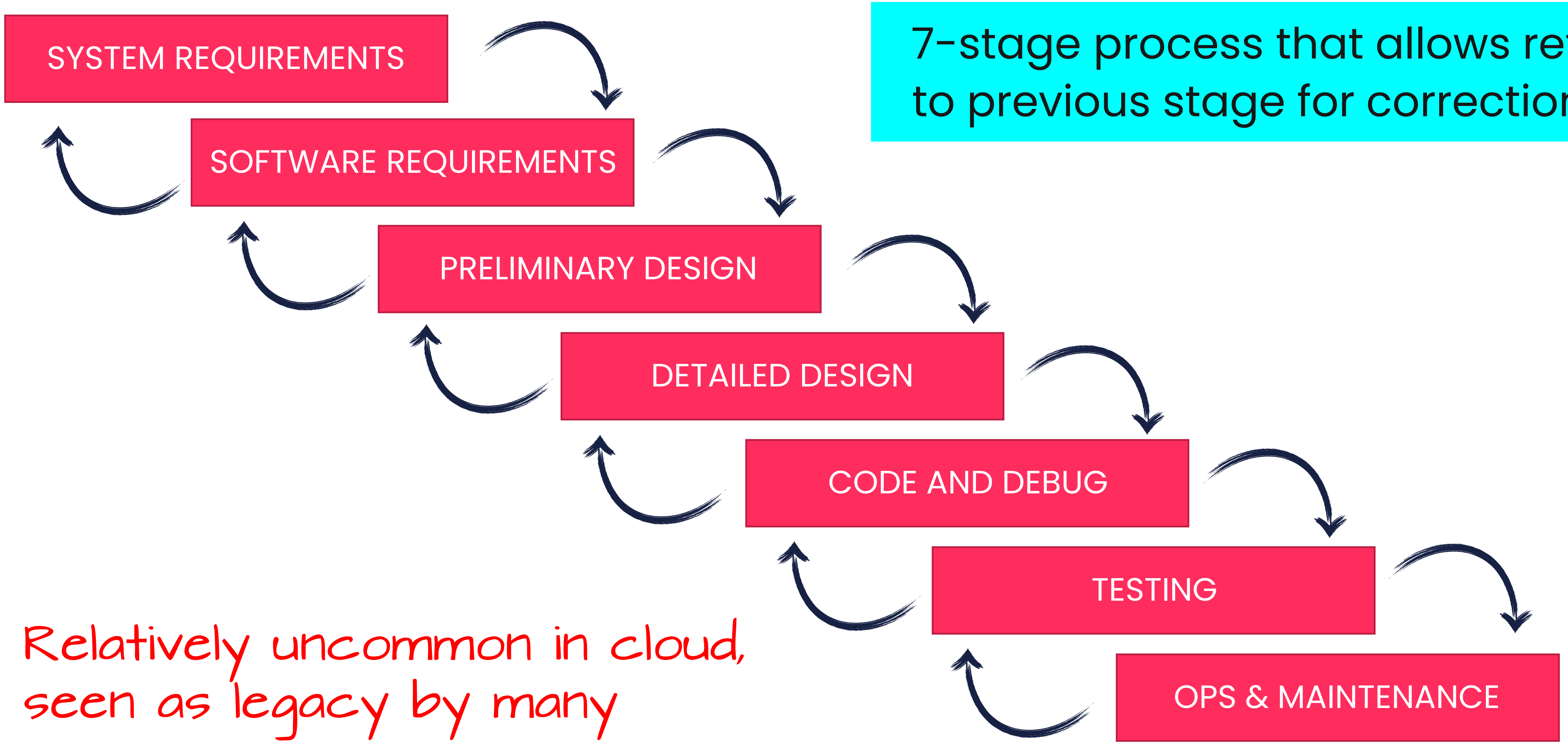
Agile | places an emphasis on the needs of the customer and quickly developing new functionality that meets those needs in an iterative fashion.

Allows quick response to changing requirements, rapid iteration

Waterfall | describes a sequential development process that results in the development of a finished product.

Requires clear requirements, stable environment, low change

WATERFALL MODEL



SYSTEM REQUIREMENTS

SOFTWARE REQUIREMENTS

PRELIMINARY DESIGN

DETAILED DESIGN

CODE AND DEBUG

TESTING

OPS & MAINTENANCE

7-stage process that allows return to previous stage for corrections

Relatively uncommon in cloud, seen as legacy by many

AGILE MODEL

model for software development
based on the following four principles

Individuals and interactions over processes and tools

Working software over comprehensive documentation

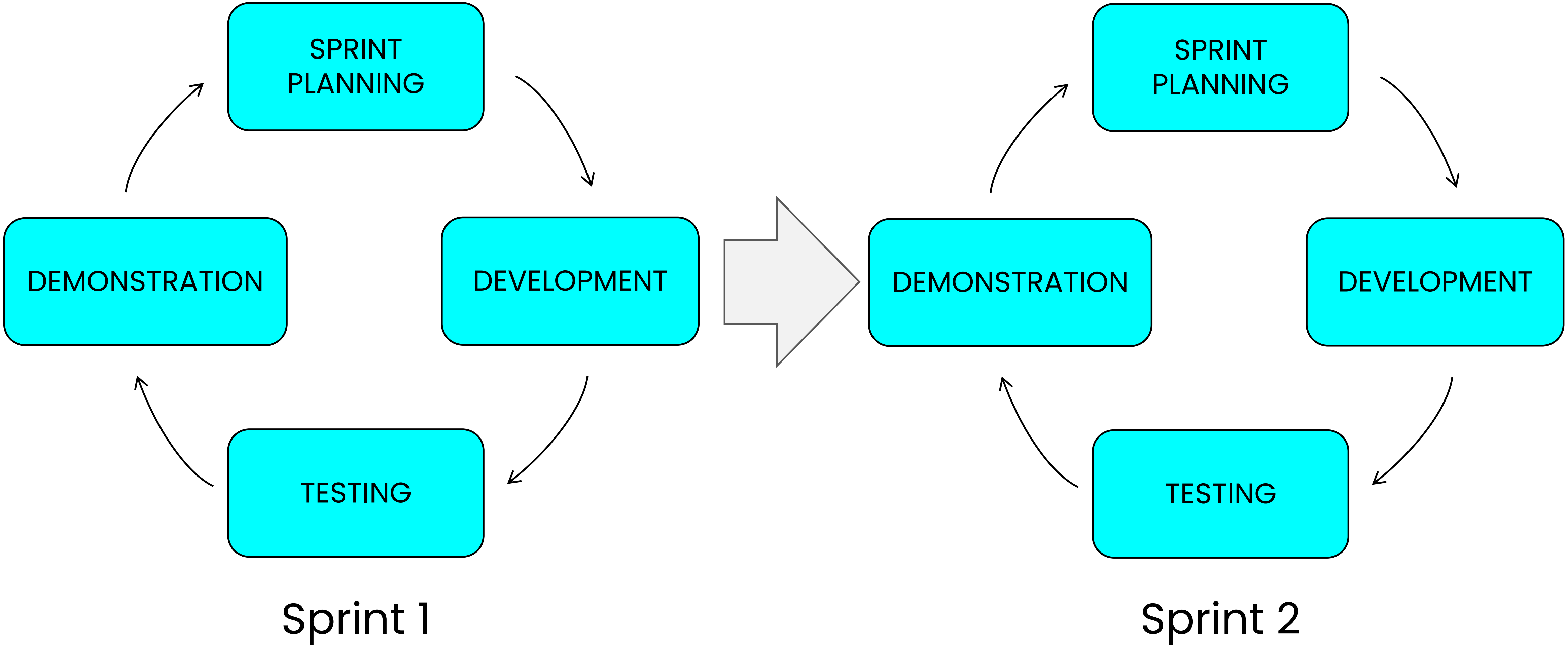
Customer collaboration over contract negotiation

Responding to change over following a plan

First described in the Manifesto for Agile Software Development (http://agilemanifesto.org) in 2001.

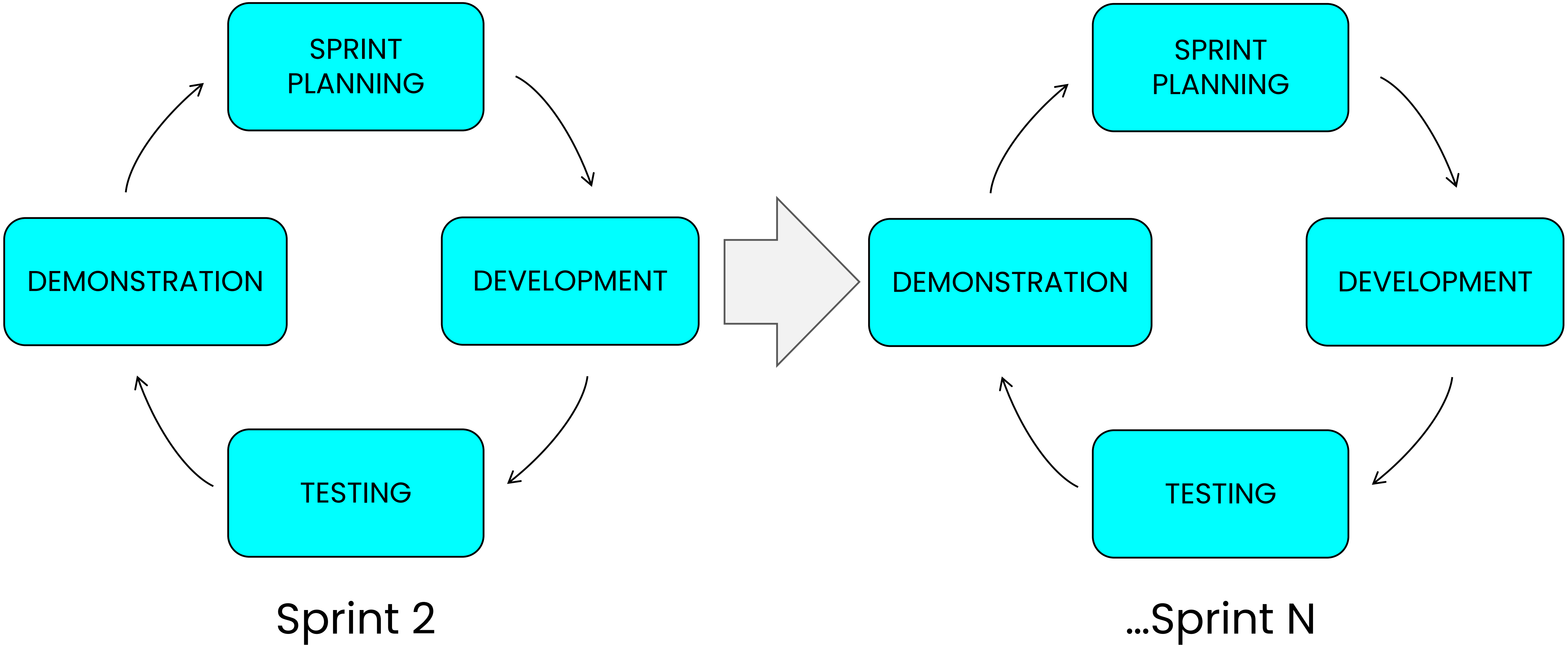
AGILE MODEL

Leverages an iterative (repeating) process called a **sprint**



AGILE MODEL

Leverages an iterative (repeating) process called a **sprint**



4. CLOUD APPLICATION SECURITY

4.3 Apply the Secure Software Development Life Cycle (SDLC)

Cloud-specific risks

Threat modeling
(e.g., Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege (STRIDE), Damage, Reproducibility, Exploitability, Affected Users, and Discoverability (DREAD), Architecture, Threats, Attack Surfaces, and Mitigations (ATASM), Process for Attack Simulation and Threat Analysis (PASTA))

Avoid common vulnerabilities during development

4. CLOUD APPLICATION SECURITY

4.3 Apply the Secure Software Development Life Cycle (SDLC)

Secure Coding
(e.g., Open Web Application Security Project (OWASP) Application Security Verification Standard (ASVS), Software Assurance Forum for Excellence in Code (SAFECode))

Software Configuration Management and Versioning

APPLY THE SECURE SOFTWARE DEVELOPMENT LIFE CYCLE

Cloud-Specific Risks

From CSA website and CBK (covered in Domain 3)

The Cloud Security Alliance details the top cloud-specific security threats in their list titled "**The CSA Egregious 11**"

1. Data Breaches
2. Misconfiguration and inadequate change control
3. Lack of cloud security architecture and strategy
4. Insufficient identity, credential access and key management
5. Account hijacking
6. Insider threat
7. Insecure interfaces and APIs
8. Weak control plane
9. "Metastructure" and "applistructure" failures
10. Limited cloud usage visibility
11. Abuse and nefarious use of cloud services

Cloud-Specific Risks

The Cloud Security Alliance details the top cloud-specific security threats in their list titled "**The CSA Egregious 11**"

In Domain 3, we covered from an architecture perspective,

Here, we will cover briefly from an SDLC perspective (DevSecOps, CI/CD)

The "CSA Egregious 11"

Data breaches Secrets management, data masking

Loss of sensitive data (PII, PHI, intellectual property) due to security breach.

Misconfiguration and inadequate change control

Software can offer the most secure configuration options, but if it is not properly set up, then the resulting system will have security issues.

CI/CD, infrastructure-as-code, release management

Lack of cloud security, architecture, and strategy

As organizations migrate to the cloud, some overlook security, or fail to consider their obligations in the shared responsibility model.

Insufficient identity, credential access, and key management

The public cloud offers benefits over legacy on-premises environments but can also bring additional complexities.

Developers can leverage identity-as-a-service (IDaaS) rather than building their own for stronger authentication & authorization controls

CLOUD-SPECIFIC RISKS

The "CSA Egregious 11"

Account hijacking

Credential theft, abuse, and/or elevation to carry out an attack.

Using existing identity providers / IDaas for your app reduces risk

Insider threat

Disgruntled employees, employee mistakes, and unintentional over-sharing.

Separation of duties, checks and balances in the release management process, such as approval gates

Insecure interfaces and APIs

Customers failing to secure access to systems gated by APIs, web consoles, etc.

Implement access controls, such as RBAC and access keys

Weak control plane

Weaknesses in the elements of a cloud system that enable cloud environment configuration and management (web console, CLI, and APIs)

Continuous Integration / Continuous Deployment (CI/CD)

Threat Modeling

Allows security practitioners to identify potential threats and security vulnerabilities

is often used as an input to risk management

Threat Modeling

Can be ***proactive*** or ***reactive***, but in either case, goal is to eliminate or reduce threats

3 approaches to threat modeling

Common approaches to threat modeling:

Focused on Assets. Uses **asset valuation** results to identify threats to the valuable assets.

Focused on Attackers. Identify potential attackers and identify threats based on the **attacker's goals**

Focused on Software. Considers **potential threats** against the software the org develops.

STRIDE

developed by Microsoft

- **S**poofing
- **T**ampering
- **R**epudiation
- **I**nformation disclosure
- **D**enial of service
- **E**levation of privilege

DOMAIN 4: THREAT MODELING

DREAD

based on answer to 5 questions

- **D**amage potential
- **R**eproducibility
- **E**xploitability
- **A**ffected users
- **D**iscoverability

DOMAIN 4: THREAT MODELING

PASTA

Stage I: Definition of Objectives

Stage II: Definition of Technical Scope

Stage III: App Decomposition & Analysis

Stage IV: Threat Analysis

Stage V: Weakness & Vulnerability Analysis

Stage VI: Attack Modeling & Simulation

Stage VII: Risk Analysis & Management

focuses on developing countermeasures based on asset value

DOMAIN 4: THREAT MODELING

A series of process steps for performing threat modeling

ATASM

Architecture
analysis of the system’s architecture

Threats
list all possible threats, threat actors, and their goals

Attack Surfaces
identify components exposed to attack

Mitigations
analyze existing mitigations in place

Because it is NOT actually a threat model itself, it can be used with threat models like STRIDE, DREAD, and PASTA.

AVOIDING COMMON CLOUD VULNERABILITIES

Like all risk mitigations, a layered approach combining multiple types of controls is a best practice, including:

Training and awareness

Training for developers is critical, because they make decisions about how to design and implement system components.

Awareness of common flaws like injection attacks prevent coding mistakes

Documented process

Secure SDLC should be well documented and communicated to all team members designing, developing, and operating systems.

Similar to security policies, must be understood and followed by developers

AVOIDING COMMON CLOUD VULNERABILITIES

Like all risk mitigations, a layered approach combining multiple types of controls is a best practice, including:

Test-driven development

Focusing on meeting acceptance criteria can be one way of simplifying the task of ensuring that security requirements are met

Having well-defined test cases for security requirements can help avoid vulnerabilities such as OWASP Top 10 application security risks.

Ensures developers know what tests will be conducted against their code

can be used to build these

💡 Common cloud vulnerabilities are well-known and documented in lists like the **OWASP Top 10** and **CSA Egregious 11**.

SECURE CODING

Secure Coding

The practice of designing systems and software to avoid security risks

Essentially a proactive risk mitigation practice

Standards and organizations exist that work to mature these practices

Exam also mentions SANS and SAFECode standards

OWASP

- Cloud Native Application Security Top 10
- Top 10 Web Application Security risks

OWASP: TOP 10 WEB APPLICATION SECURITY RISKS

The **OWASP Top 10** is an awareness document that represents a broad consensus about the most critical security risks to web applications.

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

Changes from year-to-year

FOR THE EXAM
Be familiar with meaning, don’t worry about order

OWASP: CLOUD-NATIVE APPLICATION SECURITY TOP 10

The primary goal is to provide assistance and education for organizations looking to adopt cloud-native applications securely.

1. Insecure cloud, container or orchestration configuration
2. Injection flaws (app layer, cloud events, cloud services)
3. Improper authentication & authorization
4. CI/CD pipeline & software supply chain flaws
5. Insecure secrets storage
6. Over-permissive or insecure network policies
7. Using components with known vulnerabilities
8. Improper assets management
9. Inadequate 'compute' resource quota limits
10. Ineffective logging & monitoring (e.g. runtime activity)

Know the solutions and best practices covered in this series

CWE/SANS: TOP 25 Most Dangerous Software Errors

The **TOP 25 Most Dangerous Software Errors** is not specific to cloud native environments like the OWASP Cloud Native App Security draft.

1. Out-of-bounds Write buffer overflow
2. Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
3. Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
4. Improper Input Validation Prevents injection
5. Out-of-bounds Read buffer overflow
6. Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
7. Use After Free buffer overflow
8. Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Changes from year-to-year

FOR THE EXAM
Know attack types, don't memorize the list.

CWE/SANS: TOP 25 Most Dangerous Software Errors

SANS Top 25 uses the Common Weaknesses Scoring System, or **CWSS**.

9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. NULL Pointer Dereference
12. Deserialization of Untrusted Data
13. Integer Overflow or Wraparound
14. Improper Authentication
15. Use of Hard-coded Credentials
16. Missing Authorization
17. Improper Neutralization of Special Elements used in a Command ('Command Injection')
18. Missing Authentication for Critical Function

Input validation fixes 11, 12, 13

Changes from year-to-year

FOR THE EXAM
Know attack types, don't memorize the list.

CWE/SANS: TOP 25 Most Dangerous Software Errors

The **CWSS** is composed of three scores: 1) **base finding** score, 2) **environmental** score , and 3) **attack surface** score

19. Improper Restriction of Operations within the Bounds of a Memory Buffer buffer overflow
20. Incorrect Default Permissions
21. Server-Side Request Forgery (SSRF) On OWASP List
22. Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
23. Uncontrolled Resource Consumption DoS
24. Improper Restriction of XML External Entity Reference
25. Improper Control of Generation of Code ('Code Injection')

Changes from year-to-year

FOR THE EXAM
Know attack types, don't memorize the list.

CWE/SANS: TOP 25 Most Dangerous Software Errors

The **CWSS** is composed of three scores: 1) **base finding** score, 2) **environmental** score , and 3) **attack surface** score

ATTACK TYPES and CONCEPTS

1. Injection attacks
2. Buffer overflow attacks
3. Directory / path traversal
4. Denial of Service (DoS) / Distributed DoS (DDoS)
5. Race condition
6. Authentication (AuthN) and Authorization (AuthZ)

Changes from year-to-year

FOR THE EXAM
Know attack types, don't memorize the list.

An understanding of attack types should be enough for exam day

ADDITIONAL READING

EXTRA CREDIT!

Not necessary to read these for exam day, but good for future reference!

- ✓ OWASP Cloud Native Application Security TOP 10
- ✓ OWASP Top 10 Web Application Security Risks
- ✓ SANS Top 25 Most Dangerous Software Errors
- ✓ SAFECode Secure Software Development

INJECTIONS (INJECTION ATTACKS)

Improper input handling

used to compromise web front-end and backend databases

SQL injection attacks

Use unexpected input to a web application to gain unauthorized access to an underlying database.

NOT new and can be prevented through good code practices

Countermeasures: Input validation, use prepared statements, and limit account privileges.

INJECTIONS (INJECTION ATTACKS)

Improper input handling

used to compromise web front-end and backend databases

SQL injection attacks

Use unexpected input to a web application to gain unauthorized access to an underlying database.

NOT new and can be prevented through good code practices

Countermeasures: Input validation, use prepared statements, and limit account privileges.

BUFFER OVERFLOWS

attacks attackers use to exploit **poorly written software**.

Buffer Overflow

exists when a developer does not validate user input to ensure that it is of an appropriate size (allows Input that is too large can "overflow" memory buffer).

prevent with INPUT VALIDATION !

DIRECTORY TRAVERSAL

Gaining access to restricted directories

If an attacker is able to gain access to restricted directories through HTTP, it is known as a **directory traversal attack**.

One of the simplest ways to perform directory traversal is by using a ***command injection attack*** that carries out the action.

If successful, may allow attacker to get to site's root directory,

Most vulnerability scanners will check for weaknesses with directory traversal/command injection and inform you of their presence.

To secure your system, you should run a scanner and keep the web server software patched.

RESOURCE CONSUMPTION

these are a class of attacks

Denial of-Service

is a resource consumption attack intended to prevent legitimate activity on a victimized system.

Distributed **Denial of-Service**

a DoS attack utilizing multiple compromised computer systems as sources of attack traffic.

COUNTERMEASURES: firewalls, routers, intrusion prevention (IDPS), SIEM, disable broadcast packets entering/leaving, disable echo replies, patching

RACE CONDITIONS

A condition where the system's behavior is dependent on the **sequence or timing** of other uncontrollable events.

Time-of-Check-to-Time-of-Use (TICTOU)

a timing vulnerability that occurs when a program checks access permissions too far in advance of a resource request.

Problem occurs when the state of the resource changes between the time of the check and the time it is actually used

file locking, transactions in file system or OS kernel

It becomes a bug when one or more of the possible behaviors is undesirable.

SECURE CODING

SAFECode

First published "**Fundamental Practices for Secure Software Development**"

Informed by existing models, including OWASP, CVE, CWE and the Microsoft SDL

Designed to help software industry adopt and use these best practices effectively

Last updated in 2019, unlikely to appear on CCSP exam

Includes guidance on software design, secure coding practices, testing, validation, third-party risks, and handling vulnerabilities

CCSP

CSSP EXAM CRAM

THE COMPLETE COURSE

DEMO

A quick look at the OWASP Top 10 and the SANS Top 25

Focus on familiarity with the concepts, risk, and mitigations how do you handle source code securely?

Code Repositories

This is where source code and related artifacts (such as libraries) are stored

- ✓ Do not commit sensitive information
- ✓ Protect access to your code repositories
- ✓ Sign your work
- ✓ Keep your development tools (IDE) up-to-date

Integrated Development Environment

Most code repositories today use **Git**, the worlds most widely used modern version control system

Configuration & Change Management

Can prevent security related incidents and outages

Configuration Management

ensures that systems are configured similarly, configurations are known and documented.

Baselining ensures that systems are deployed with a common baseline or starting point, and imaging is a common baselining method.

Change Management

helps reduce outages or weakened security from unauthorized changes.

Versioning uses a labeling or numbering system to track changes in updated versions of software.

Approaches vary, but often include a major version, minor version, and patch version strategy (23.05.02)

CONFIGURATION MANAGEMENT

Tracks the way that systems are set up:
hardware and software (OS and applications)

SCM
Software Configuration Management

Baselining is an important component of configuration management.

a baseline is a **snapshot** of a system or application at a given point in time

should also create **artifacts** that may be used to help understand system configuration

system and component-level **versioning**

applications depend on compute resources and software components

CONFIGURATION MANAGEMENT

Tracks the way that systems are set up:
hardware and software (OS and applications)

An emerging strategy and standard in tracking software versions is **software bill of materials (SBOM)**.

The SBOM lists all of the components in an application or service, including open source or proprietary code libraries.

SBOM is mentioned briefly in OSG, but not in exam syllabus.

Something you can expect to see more of in the future

4. CLOUD APPLICATION SECURITY

4.4 Apply Cloud Software Assurance and Validation

Functional and non-functional testing

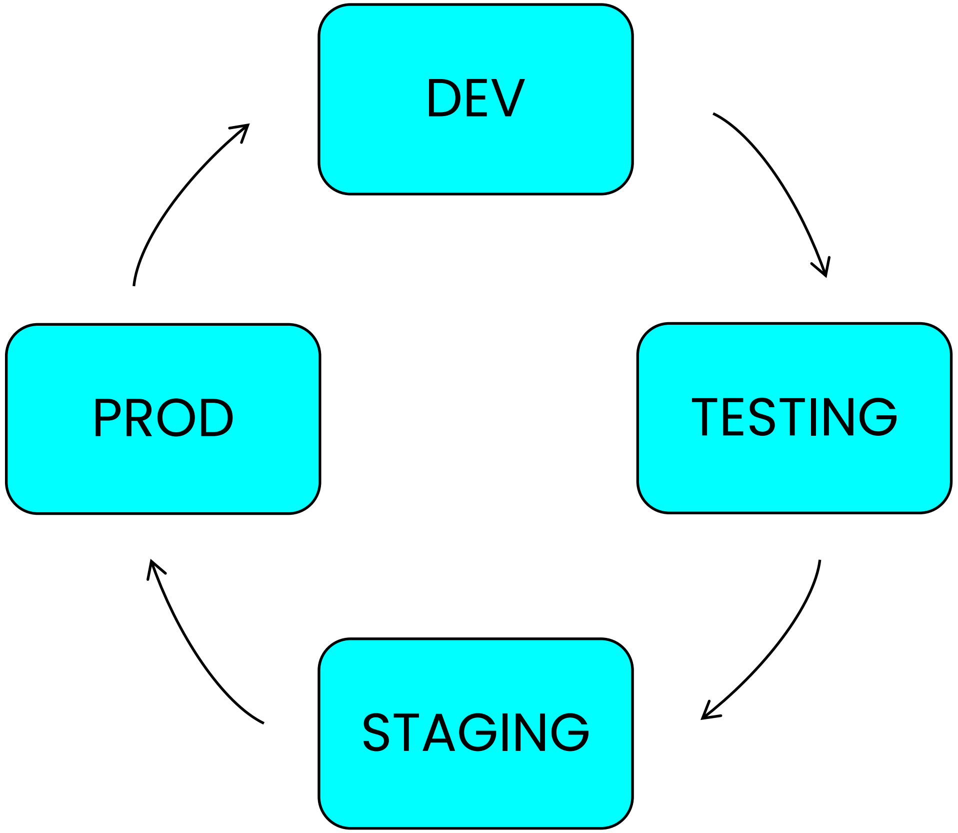
Security testing methodologies
(e.g., blackbox, whitebox, static, dynamic, Software Composition Analysis (SCA), interactive application security testing (IAST))

Quality assurance (QA)

Abuse case testing

ENVIRONMENT

Secure environments for development, testing, and staging before moving the application into production are necessary.



Environments map to phases of application development, debugging, testing, and release.

Development. Where the application is initially coded, often through multiple iterations (versions).

Testing. where developers integrate all of their work into a single application. Regression testing to ensure functionality is as expected.

Staging. where we ensure quality assurance before we roll it out to production. QA happens here !

Production. where the application goes live, and end-users have the support of the IT team.

FUNCTIONAL AND NON-FUNCTIONAL TESTING

Functional testing

determines if software meets functionality requirements defined earlier in the SSDLC

takes multiple forms, including:

integration testing that validates whether components work together,

regression testing that validates whether bugs were reintroduced between versions

user acceptance testing, which test how users interact with and operate the software

Focuses on specific features and functionality

FUNCTIONAL AND NON-FUNCTIONAL TESTING

Non-functional testing

focuses on the quality of the software

looks at software qualities like stability and performance.

methods include load, stress, recovery, and volume tests

Examines the way the system operates as a whole, not the specific functions

FUNCTIONAL SECURITY REQUIREMENTS

Functional vs Non-Functional security requirements

What is the difference?

Functional security requirements

Define a system or its component and specifies what it must do.

Captured in use cases, defined at a component level.

EXAMPLE: application forms must protect against injection attacks.

Non-functional security requirements

Specify the system's quality, characteristics, or attributes.

Apply to the whole system (system level)

EXAMPLE: security certifications are non-functional.

FROM DOMAIN 1

SECURITY TESTING METHODOLOGIES

Static

Application Security Testing

analysis of computer software performed without actually executing programs

tester has access to the underlying framework, design, and implementation

tests "inside out"

requires source code

Dynamic

Application Security Testing

a program which communicates with a web application (executes the application).

tester has no knowledge of the technologies or frameworks that the application is built on

tests "outside in"

no source code required

SECURITY TESTING METHODOLOGIES

White box testing

conducted with full access to and knowledge of systems, code, and environment

Static application testing is one example

"full knowledge testing"

Black box testing

conducted as an external attacker would access the code, systems, or environment,

tester has no knowledge of any of these elements at the outset of a test.

"zero knowledge testing"

no source code required

SECURITY TESTING METHODOLOGIES

IAST

Interactive Application Security Testing

analyzes code for vulnerabilities while it's being used

focuses on real time reporting to optimize testing and analysis process

Often built into CI/CD automated release testing

Unlike static and dynamic testing, IAST analyzes the internal functions of the application while it is running

SECURITY TESTING METHODOLOGIES

SCA

Software Composition Analysis

is used to track the components of a software package or application

is of special concern for apps built with open-source software components

because open-source components often involve reusable code libraries

Automated, combines application security and patch management

SCA tools identify flaws/vulnerabilities in these included components, ensures latest versions are in use, etc.

Quality Assurance

QA is responsible for ensuring that the code delivered to the customer through the cloud environment is quality code, defect-free, and secure.

PROCESS: is frequently a combination of automated and manual validation testing techniques.

Typically involves reviews, testing, reporting, and other activities to complete the QA process.

GOAL: is to ensure software meets standards or requirements.

ROLE: The role of QA is significantly expanded in a DevOps or DevSecOps team, where QA is embedded throughout the development process

TESTS: QA should be involved in many testing activities, such as load, performance and stress testing, as well as vulnerability management.

WHAT IS AN Abuse Case?

A way to use a feature that was not expected by the implementer, allowing an attacker to influence the feature or outcome of use of the feature based on the attacker action (or input).

Describes unintended and malicious use scenarios of the application, describing how an attacker could do this.

ABUSE CASE TEST

Abuse Case Test

Focuses on using features in ways that weren't intended by the developer.

May exploit weaknesses or coding flaws from perspective of multiple personas: malicious user, abusive user, and unknowing user

Can help orgs to consider security features and controls needed for an application

Testing generally focuses on documented abuse cases

OWASP provides an **"Abuse Case Cheat Sheet"** in their cheat sheet series at owasp.org

4. CLOUD APPLICATION SECURITY

4.5 Use Verified Secure Software

With exception below, all Domain 4 content comes from OSG Chapter 6!

Securing application programming interfaces (API)

Supply-chain management
(e.g., vendor assessment)

Third-party software management
(e.g., licensing)

OSG chap 5

Validated open-source software

APIs (SOAP or REST)

is a set of exposed interfaces that allow programmatic interaction between services. no user/human involved

SOAP is a standard communication protocol system that uses XML technologies

REST is an architectural model that uses HTTPS for web communications to offer API endpoints

Security features from CSP include API gateway, authentication, IP filtering, throttling, quotas, data validation

Also ensure that storage, distribution, and transmission of access keys is performed in a secure fashion.

Supply Chain

Today, most services are delivered through a chain of multiple entities

Supply Chain

A secure supply chain includes **vendors** who are secure, reliable, trustworthy, reputable

Due diligence should be exercised in assessing vendor security posture, business practices, and reliability

Supply Chain

A secure supply chain includes **vendors** who are secure, reliable, trustworthy, reputable

May include periodic attestation requiring vendors to confirm continued implementation of security practices

Supply Chain

A secure supply chain includes **vendors** who are secure, reliable, trustworthy, reputable

A vulnerable vendor in the supply chain puts the organization at risk

Supply Chain Evaluation

Traditional vendor evaluation options may include

On-Site Assessment . Visit organization, interview personnel, and observe their operating habits.

Document Exchange and Review . Investigate dataset and doc exchange, review processes.

Process/Policy Review . Request copies of their security policies, processes, or procedures.

Third-party Audit. Having an independent auditor provide an unbiased review of an entity's security infrastructure.

These are all ways a CSP might evaluate a vendor!

Vendor evaluation in the cloud

Companies with hundreds or thousands of customers (like AWS, Azure, GCP) cannot support direct vendor assessment.

Instead, review audit and certification reports from the CSP

Third-party Audit. Review an independent auditor's unbiased review of an entity's security infrastructure.

Review **SOC-2 Type II report**, and **ISO/IEC 27001, 27017, 27018 reports** to verify efficacy of the CSPs physical and logical controls for securing facilities, infrastructure, and data.

As we saw in Domain 3, major CSPs generally make these reports available to customers for review on-demand!

THIRD PARTY SOFTWARE MANAGEMENT

The use of third-party software adds additional risk.

Third-party software risks

A third party may have limited access to your systems but will often have direct access to some portion of your data.

Typical issues addressed in software vendor assessment include:

- Where in the cloud is the software running? Is this on a well-known CSP, or does the provider use their own cloud service?
- Is the data encrypted at rest and in transit, and what encryption technology is used?
- How is access management handled?
- What event logging can you receive?
- What auditing options exist?

The focus is risk
to data security

OSS vs PROPRIETARY

Open Source

One in which the vendor makes the license freely available and allows access to the source code, though it might ask for an optional donation.

There is no vendor support with open source, so you might pay a third party to support in a production environment.

EXAMPLE: One of the more popular open-source firewalls is **pfsense**, the details for which can be found at https://www.pfsense.org/.

Proprietary

Are more expensive but tend to provide more/better protection and more functionality and support (at a cost).

Many vendors in this space, including Cisco, Checkpoint, Pal Alto, Barracuda. but "no source code access"

VALIDATED OPEN-SOURCE SOFTWARE

All software, including open-source software (OSS), must be validated in a business environment.

Risks of open-source software

Some argue that open-source software is more secure because the source code is available to review.

Adequate validation testing is required and may be achieved through :

- Sandbox testing
- Vulnerability scans
- Third-party verifications

EXAMPLE:

OSS projects like OpenSSL and Apache have contained serious vulnerabilities in the past

While more visibility into a problem can result in better security outcomes, the transparency of OSS is NOT a guarantee of security.

4. CLOUD APPLICATION SECURITY

4.6 Comprehend the Specifics of Cloud Application Architecture

Supplemental security components
(e.g., web application firewall (WAF), Database Activity Monitoring (DAM), Extensible Markup Language (XML) firewalls, application programming interface (API) gateway)

Cryptography

Sandboxing

Application virtualization and orchestration
(e.g., microservices, containers)

SUPPLEMENTAL SECURITY COMPONENTS

Web App Firewall

aka "WAF"

protects web applications by filtering and monitoring HTTP traffic between a web application and the Internet.

typically protects web applications from common attacks like XSS, CSRF, and SQL injection.

Often include OWASP core rule sets (CRS)

XML Firewall

used to protect services that rely on XML based interfaces including some web apps

provides request validation and filtering, rate limiting, and traffic flow management

Usually implemented as a proxy

SUPPLEMENTAL SECURITY COMPONENTS

Database Activity Monitoring

aka "DAM"

combines network data and database audit info in real time to analyze database activity for unwanted, anomalous, or unexpected behavior.

monitors application activity, privileged access, and detects attacks through behavioral analysis

Most CSPs offer some form of DAM tooling

API Gateway

monitors traffic to your application services, exposed as API endpoints

provides authentication and key validation services that control API access

e.g. Amazon API Gateway, Azure API Management

CCSP

CSSP EXAM CRAM

THE COMPLETE COURSE

DEMO

Examining OWASP core rule sets (CRS) in web app firewalls

EXAMPLE FOR CONTEXT:

Capabilities will vary by CSP

INSIDE CLOUD
AND SECURITY

CLOUD SECURITY CONTROLS - SOLUTIONS

Firewall Considerations in a Cloud Environment

One reason that we need a good firewall is to filter incoming traffic to protect our cloud-hosted infrastructure and applications from hackers or malware.

For example, the most common cloud firewall is the Web Application Firewall (WAF)

Cost

Cost is one of the reasons for WAF popularity. It meets a common need, is easy to configure, and is less expensive than more function-rich NGFW and SWG options.

Need for Segmentation:

Network segmentation should be supported with appropriate traffic filtering/restriction with the firewall type that is most appropriate for the use case.

The firewall can filter traffic between virtual networks and the Internet.

Open Systems Interconnection (OSI) Layers

A network firewall works on Layer 3, stateful packet inspection at layers 3/4.

Many cloud firewalls, like Web Application Firewalls work at Layer 7 of the OSI.

CRYPTOGRAPHY

Cryptography in Domain 4 touches on three areas:

- ✓ Data at rest
- ✓ Data in motion
- ✓ Key management

PROTECTING DATA AT REST

These features generally include a customer-managed key option

How can we encrypt different types of data **at rest**?

Storage Service Encryption

CSPs usually encrypt by default

CSP storage providers usually protect data at rest by automatically encrypting before persisting it to managed disks, Blob Storage, file, or queue storage.

Full Disk Encryption

CSPs offer in the IaaS model

helps you encrypt Windows and Linux IaaS VMs disks using BitLocker (Windows) and dm-crypt feature of Linux to encrypt OS and data disks.

Transparent data encryption (TDE)

Helps protect SQL Database and data warehouses against threat of malicious activity with real-time encryption and decryption of database, backups, and transaction log files at rest without requiring app changes.

PROTECTING DATA IN MOTION

How can we encrypt different types of **data in motion**?

> Data in motion is most often encrypted using **TLS** (**HTTPS**)
>
> Hybrid (site-to-site) and cross-cloud connectivity is often encrypted by **VPN**

While similar in function, TLS has largely replaced SSL

Sandboxing

Places the systems or code into an isolated, secured environment where testing can be performed.

Cloud sandboxing architectures often create independent, ephemeral environments for testing.

Enables patch and test and ensures a system is secure before putting it into a production environment.

Also facilitates investigating dangerous malware.

Sandboxes provide an environment for evaluating the security of code without impacting other systems.

Containerization

Examples include Docker and Kubernetes

A lightweight, granular, and portable way to package applications for multiple platforms.

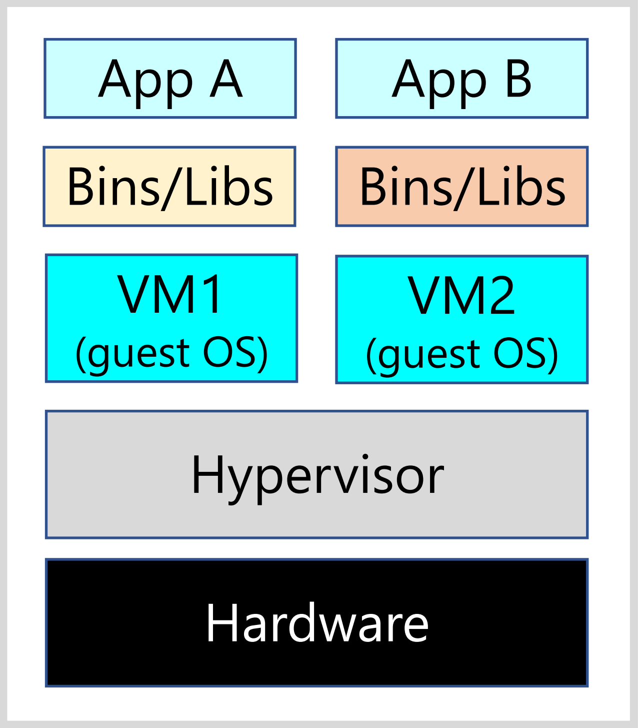
Reduces overhead of server virtualization by enabling containerized apps to run on a shared OS kernel.

containers do not have their own OS !

Can be used in some cases to isolate existing applications developed to run in a VM with a dedicated operating system.

TYPE 1 HYPERVISOR

"Bare metal"

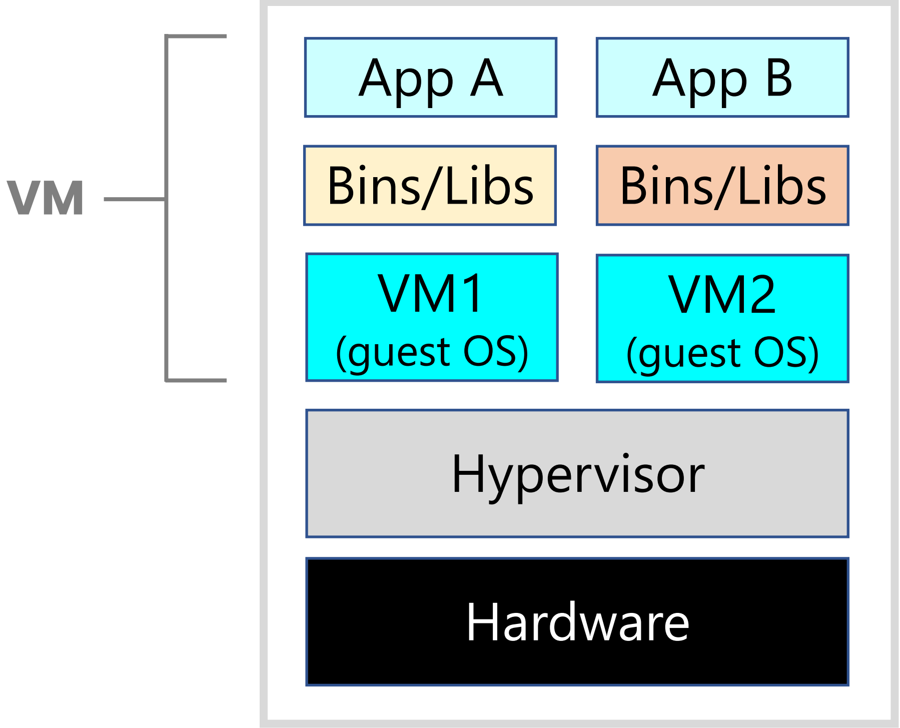


VMware ESXi, KVM
Microsoft Hyper-V

VIRTUALIZATION SECURITY: CONTAINERS

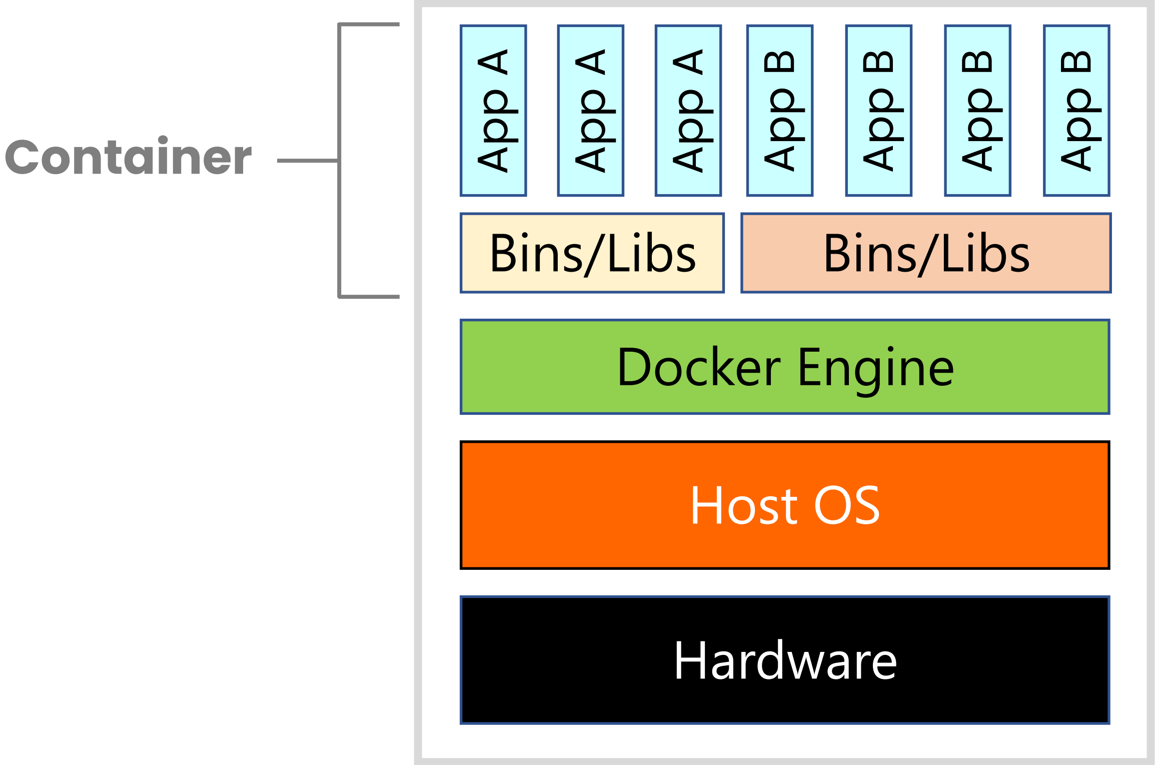
TYPE 1 HYPERVISOR

"Bare metal"



CONTAINER HOST

Usually, a cloud VM



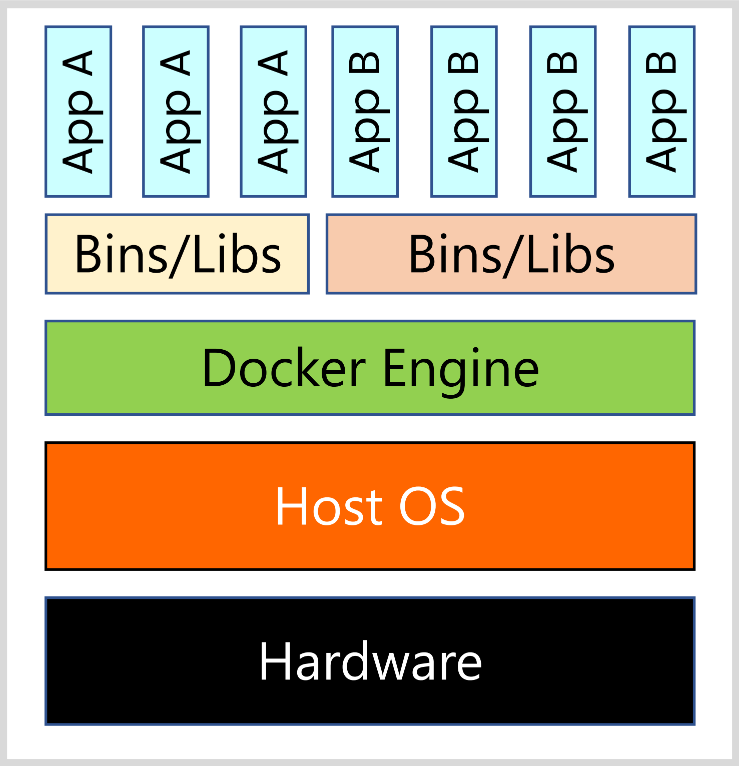
Each VM has its own OS kernel and memory, resulting in more overhead

Containers are isolated, but share a single OS kernel, as well as bins/libs where possible

VIRTUALIZATION SECURITY: **CONTAINERS**

CONTAINER HOST

Usually, a cloud VM



Core components in a container platform (Docker, Kubernetes):

-Orchestration/scheduling controller

-Network, storage

-Container host

-Container images

-Container registry

The isolation is logical, isolating processes, compute, storage, network, secrets, and management plane

CONTAINER ORCHESTRATION

Kubernetes a **container orchestration platform** for scheduling and automating the deployment, management, and scaling of containerized applications.

Managed Kubernetes

Container hosts are cloud-based virtual machines (VM). This is where the containers run

Most CSPs offer **hosted Kubernetes service**, handles critical tasks like health monitoring and maintenance for you. Platform-as-a-Service

You pay only for the agent nodes within your clusters, not for the management cluster.

Major CSPs also offer a monitoring solution that will identify at least some potential security concerns

EXAMPLES: AKS (MSFT), EKS (AWS), GKE (GCP)

CLOUD ORCHESTRATION

cloud orchestration allows a customer to manage their cloud resources centrally in an efficient and cost-effective manner.

This is especially important in a multi-cloud environment.

Management of the complexity of corporate cloud needs will only increase as more computing workloads move to the cloud.

Allows the automation of workflows, management of accounts in addition to the deployment of cloud and containerized applications.

Implements automation in a way that manages cost and enforces corporate policy in and across clouds.

Major CSPs offer orchestration tools that work on their platform and third parties offer multi-cloud orchestration solutions

4. CLOUD APPLICATION SECURITY

4.7 Design Appropriate Identity and Access Management (IAM) Solutions

Federated Identity

Identity Providers (IdP)

Single Sign-On (SSO)

Multi-Factor Authentication (MFA)

Cloud Access Security Broker

Secrets Management

DESCRIBE THE CONCEPT OF FEDERATED SERVICES

Federation is a collection of domains that have **established trust.**

The level of trust may vary, but typically includes authentication and almost always includes authorization.

Often includes a number of organizations that have established trust for shared access to a set of resources.

Example

You can federate your on-premises environment with Azure AD and use this federation for authentication and authorization.

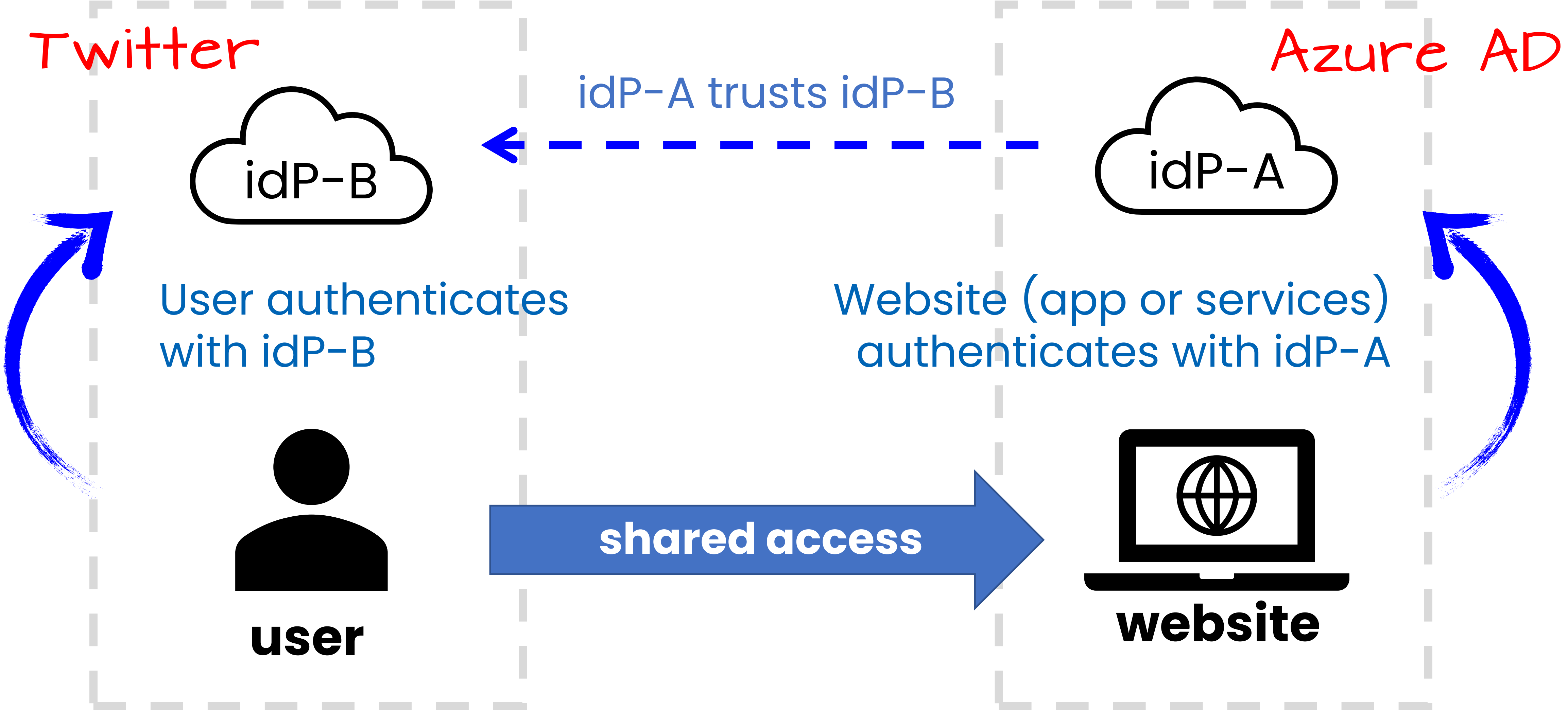
This sign-in method ensures that all user authentication occurs on-premises.

Allows administrators to implement more rigorous levels of access control.

Certificate authentication, key fob, card token ←

IDENTITY FEDERATION (EXAMPLE)

may be cloud or on-premises



Twitter
Azure AD
idP-A trusts idP-B
idP-B
idP-A
User authenticates with idP-B
Website (app or services) authenticates with idP-A
user
shared access
website

trust is not always bi-directional

IDENTITY PROVIDERS

Identity Providers

Creates, maintains, and manages identity information while providing authentication services to applications.

For example, Azure Active Directory is the identity provider for Office 365

Other IDaaS options include OKTA and DUO

Social identity providers that support OAuth, like Google, Facebook, and Apple are common in federation scenarios

AUTHENTICATION/AUTHORIZATION

Single Sign-on (SSO)

Single sign-on means a user doesn't have to sign into every application they use.

AUTHENTICATION/AUTHORIZATION

Single Sign-on (SSO)

Single sign-on means a user doesn't have to sign into every application they use.

The user logs in once and that credential is used for multiple apps.

AUTHENTICATION/AUTHORIZATION

Single Sign-on (SSO)

Single sign-on means a user doesn't have to sign into every application they use.

The user logs in once and that credential is used for multiple apps.

Single sign-on based authentication systems are often called "**modern authentication**".

This is a common user experience issue in enterprise desktop scenarios

MFA ATTACK PREVENTION

Multi-factor Authentication

Something you **know** (pin or password)

Something you **have** (trusted device)

Something you **are** (biometric)

MFA is a preventative security control for multiple attacks

PREVENTS:

These attacks are all used in credential theft

- Phishing
- Spear phishing
- Keyloggers
- Credential stuffing
- Brute force and reverse brute force attacks
- Man-in-the-middle (MITM) attacks

IAM SOLUTIONS

CASB

Cloud Access Security Broker

Enforces the company’s data security policies between on-premises and the cloud.

Can detect (and optionally, prevent) data access with unauthorized apps and data storage in unauthorized locations.

Combines the ability to control use of services with data loss prevention and threat management features

There are on-premises, hybrid, and cloud hosted models

Often used in enterprise scenarios where high levels of control and assurance in cloud usage are necessary

CLOUD SECURITY CONTROLS

Secrets management

CSPs offer a cloud service for centralized secure storage and access for application secrets

A secret is anything that you want to control access to, such as **API keys**, **passwords**, **certificates**, **tokens**, or **cryptographic keys.**

Service will typically offer programmatic access via API to support DevOps and continuous integration/continuous deployment (CI/CD)

Access control at vault instance-level and to secrets stored within.

Your CI/CD pipelines should leverage centralized storage of secrets rather than hard-coded values or storage on disk

INSIDE CLOUD

AND SECURITY

THANKS

FOR WATCHING!