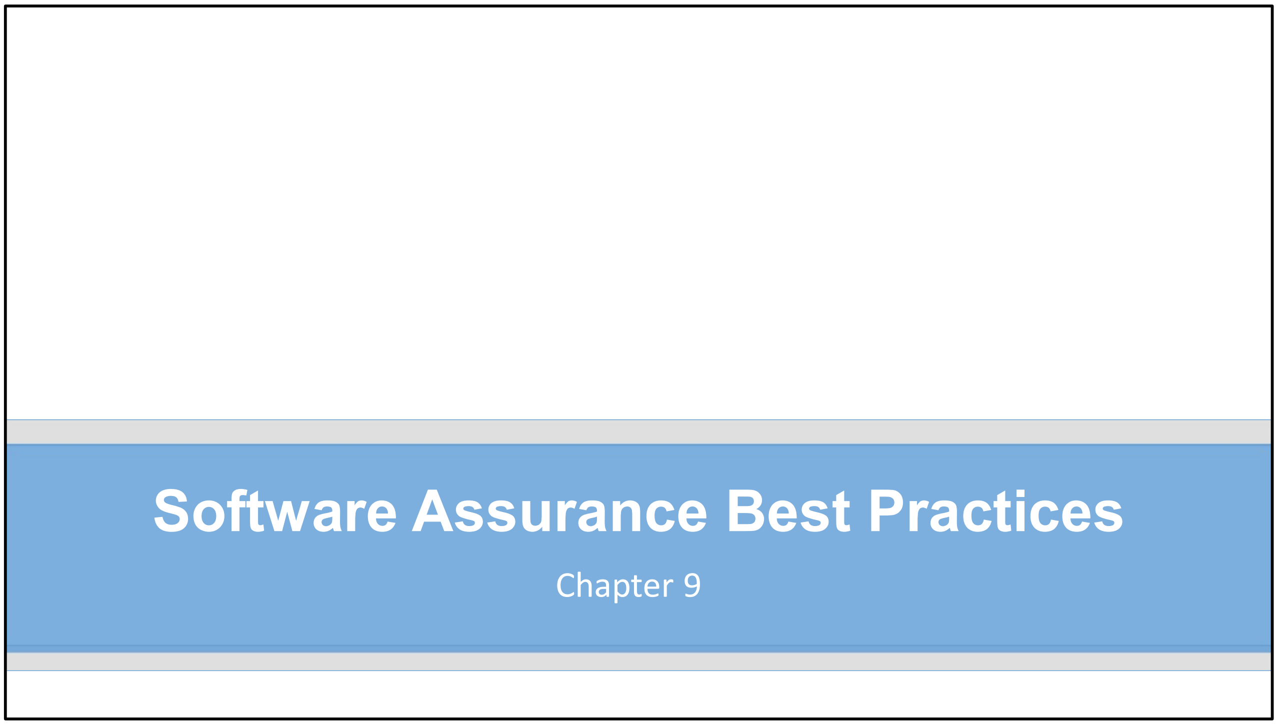


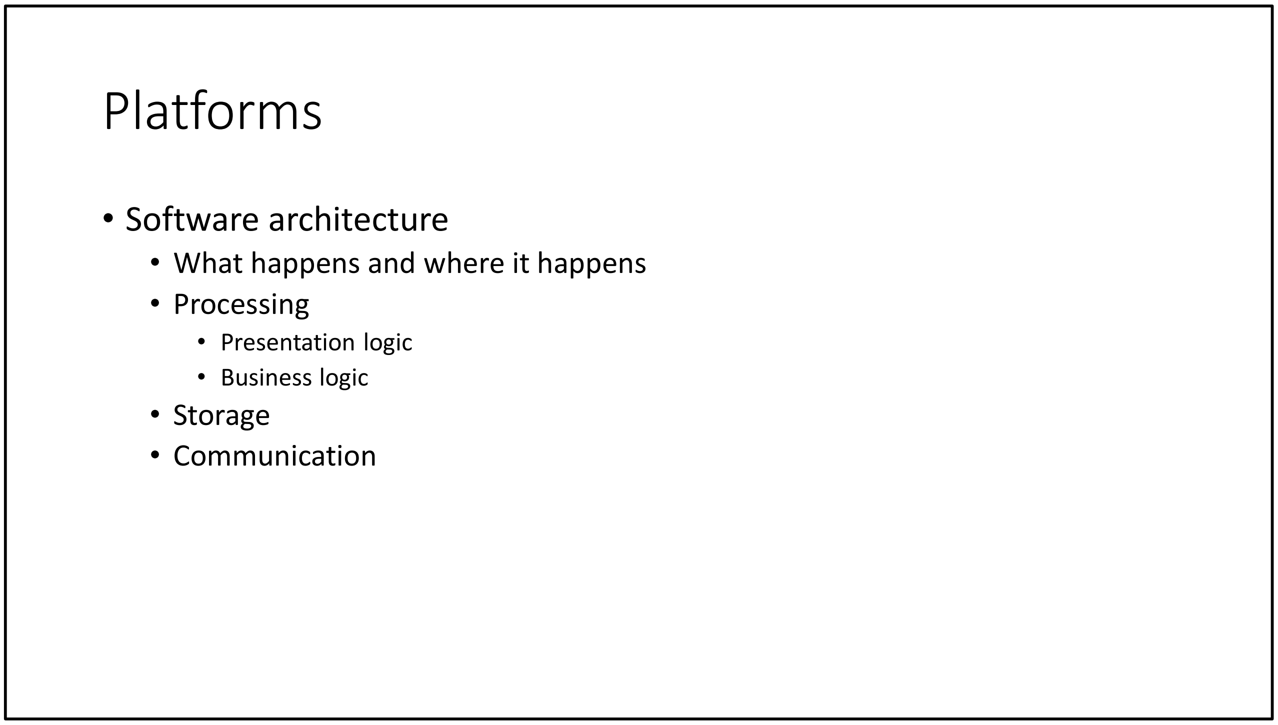
Software Assurance Best Practices

Chapter 9

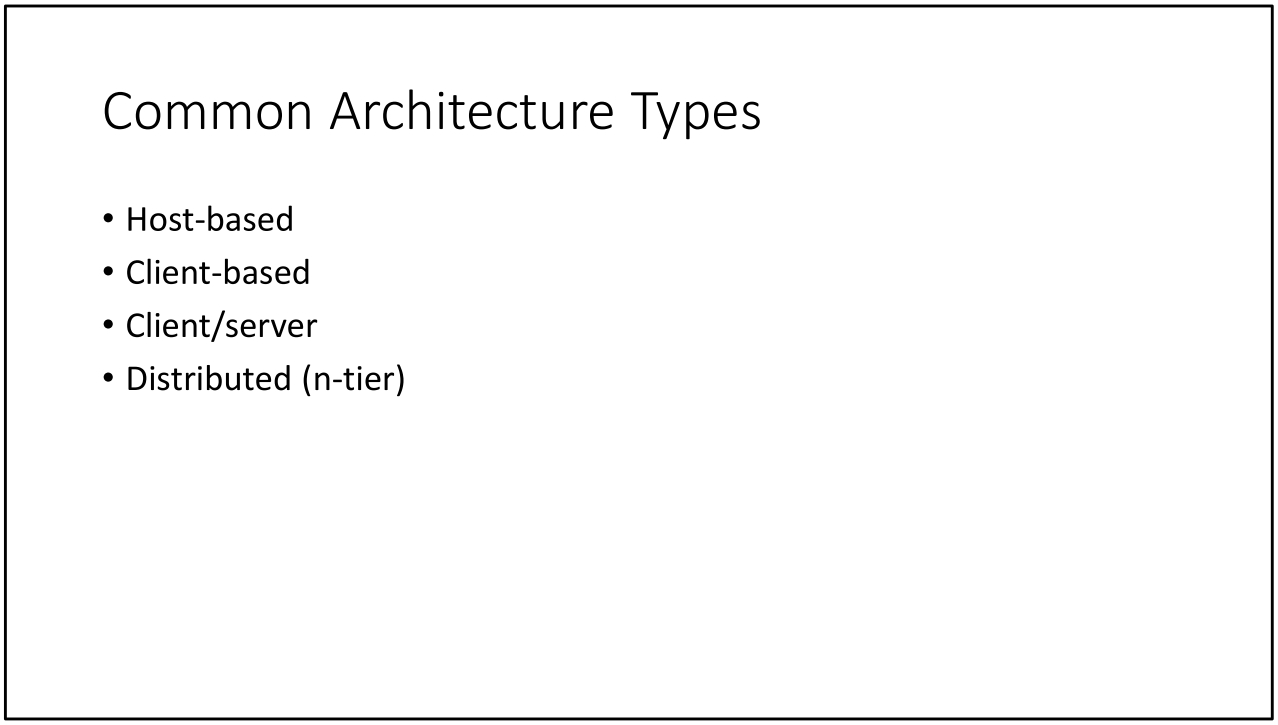
Episode 9.01

Platforms

Objective 2.2 Explain software assurance best practices.
• Platforms
- Mobile
- Web application
- Client/server
- Embedded
- System-on-chip (SoC)
- Firmware

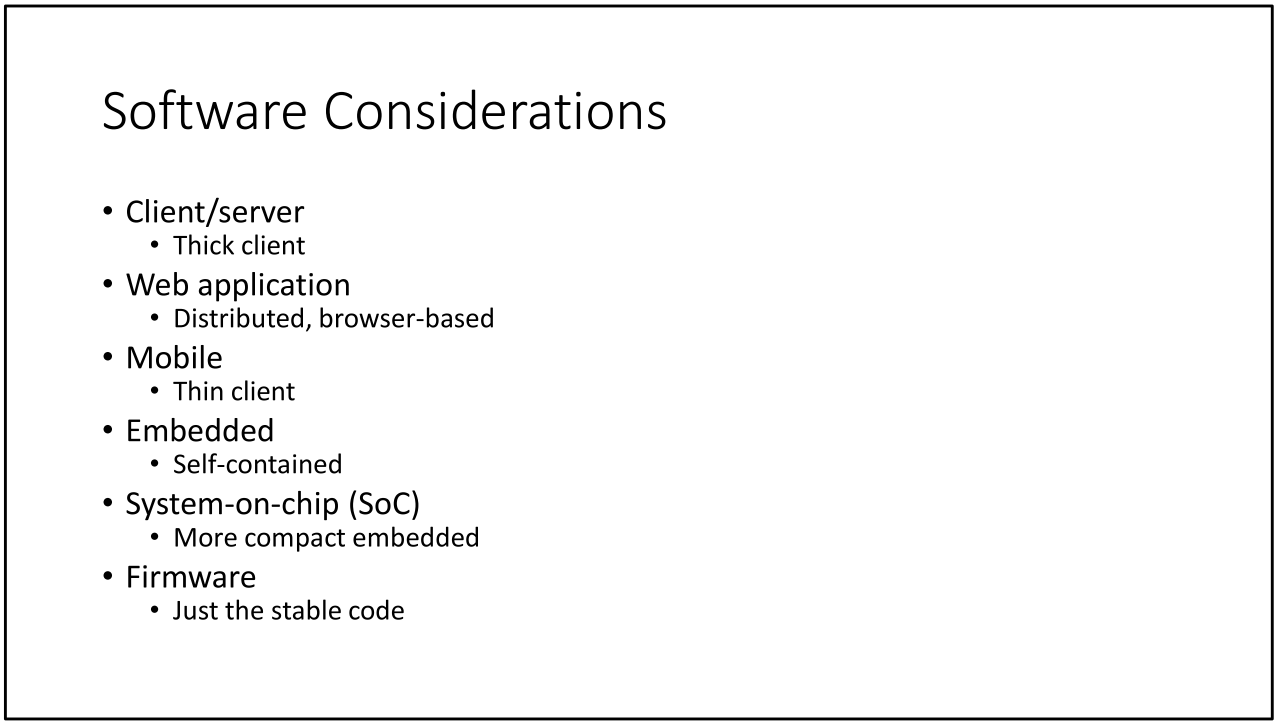

Platforms

- Software architecture
  - What happens and where it happens
  - Processing
    - Presentation logic
    - Business logic
  - Storage
  - Communication



Common Architecture Types

- Host-based
- Client-based
- Client/server
- Distributed (n-tier)



Software Considerations

- Client/server
  - Thick client
- Web application
  - Distributed, browser-based
- Mobile
  - Thin client
- Embedded
  - Self-contained
- System-on-chip (SoC)
  - More compact embedded
- Firmware
  - Just the stable code

Episode 9.02

SOA and DevSecOps

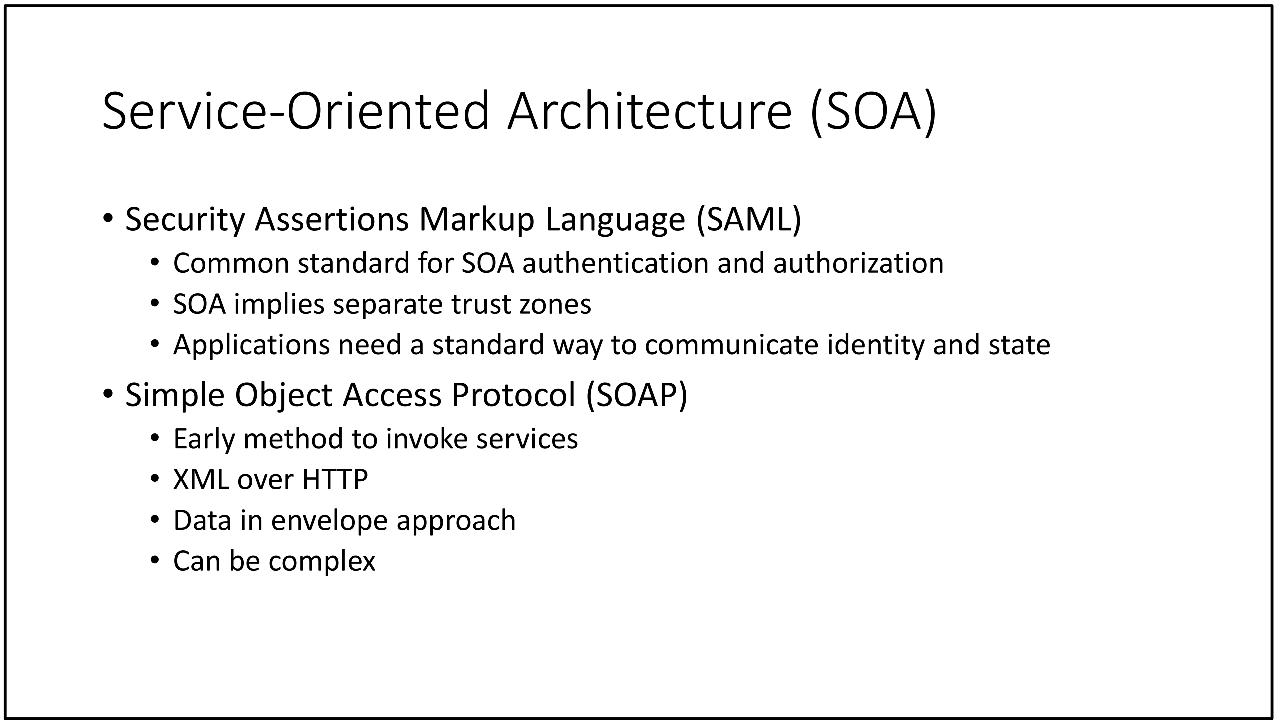
Objective 2.2 Explain software assurance best practices.

- DevSecOps
- Service-oriented architecture

- Security Assertions Markup Language (SAML)
- Simple Object Access Protocol (SOAP)
- Representational State Transfer (REST)
- Microservices

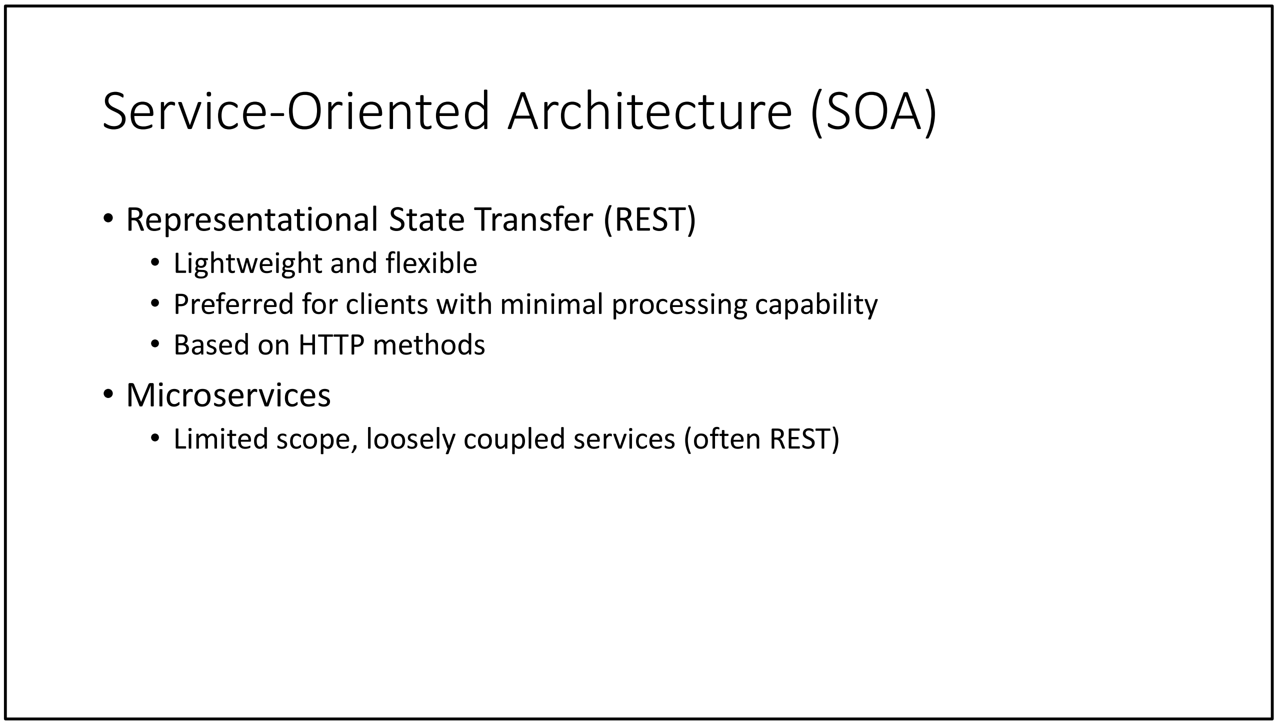
Service-Oriented Architecture (SOA)

- SOA clients consume services via APIs
- Usually based on web approach
  - Loosely-coupled connection between client and server
- Two types of connections
  - Persistent
  - Non-persistent
- SOAs can be open or closed
- Cloud computing makes SOA far easier to scale



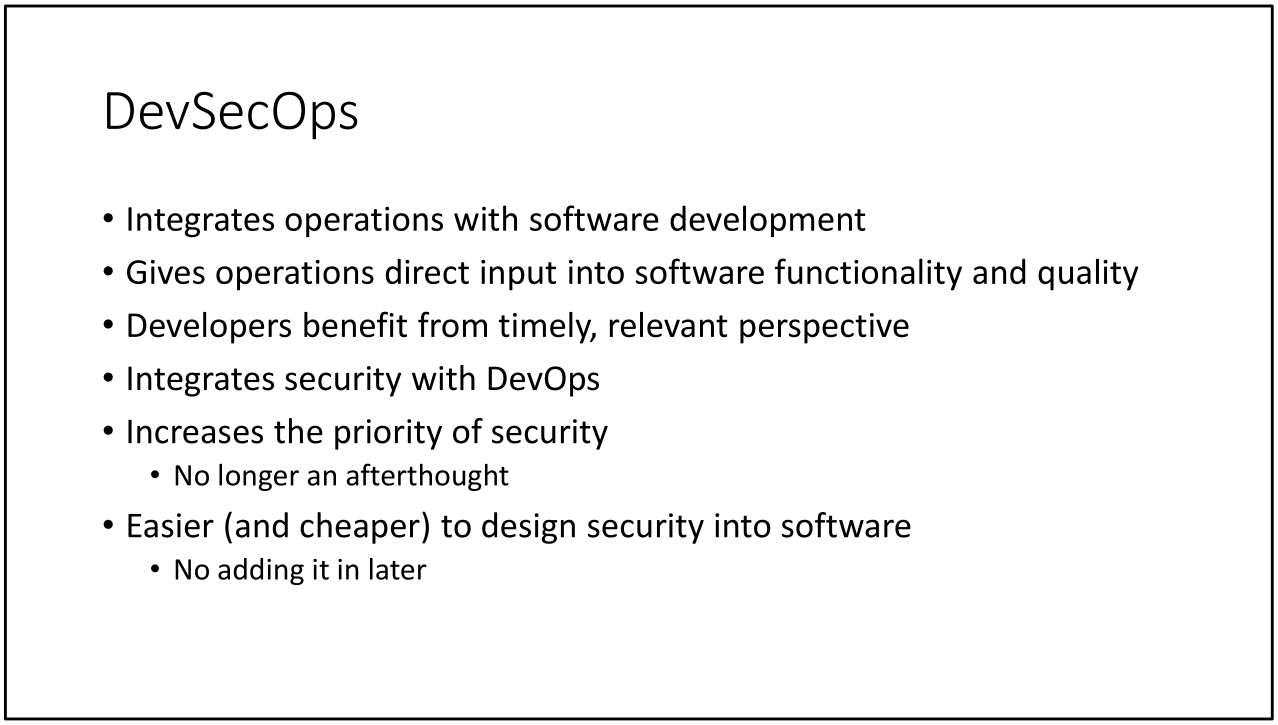
Service-Oriented Architecture (SOA)

- Security Assertions Markup Language (SAML)
  - Common standard for SOA authentication and authorization
  - SOA implies separate trust zones
  - Applications need a standard way to communicate identity and state
- Simple Object Access Protocol (SOAP)
  - Early method to invoke services
  - XML over HTTP
  - Data in envelope approach
  - Can be complex



Service-Oriented Architecture (SOA)

- Representational State Transfer (REST)
  - Lightweight and flexible
  - Preferred for clients with minimal processing capability
  - Based on HTTP methods
- Microservices
  - Limited scope, loosely coupled services (often REST)



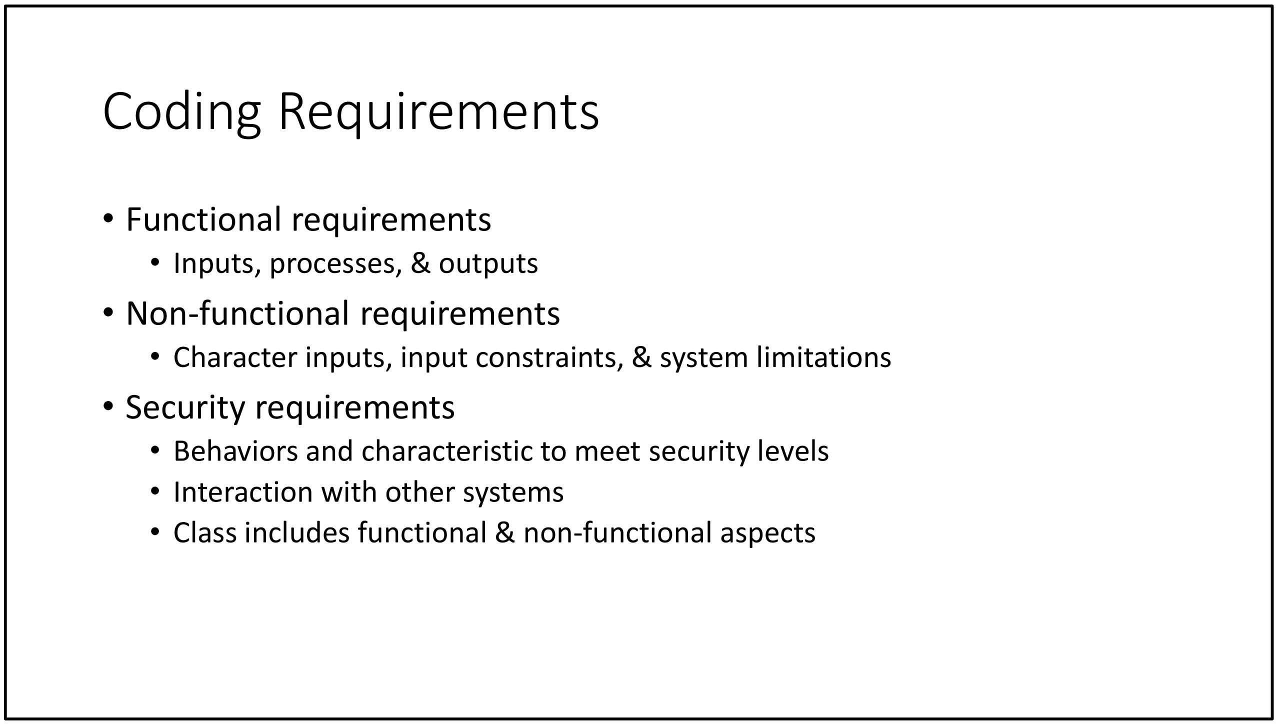
DevSecOps

- Integrates operations with software development
- Gives operations direct input into software functionality and quality
- Developers benefit from timely, relevant perspective
- Integrates security with DevOps
- Increases the priority of security
  - No longer an afterthought
- Easier (and cheaper) to design security into software
  - No adding it in later

Episode 9.03

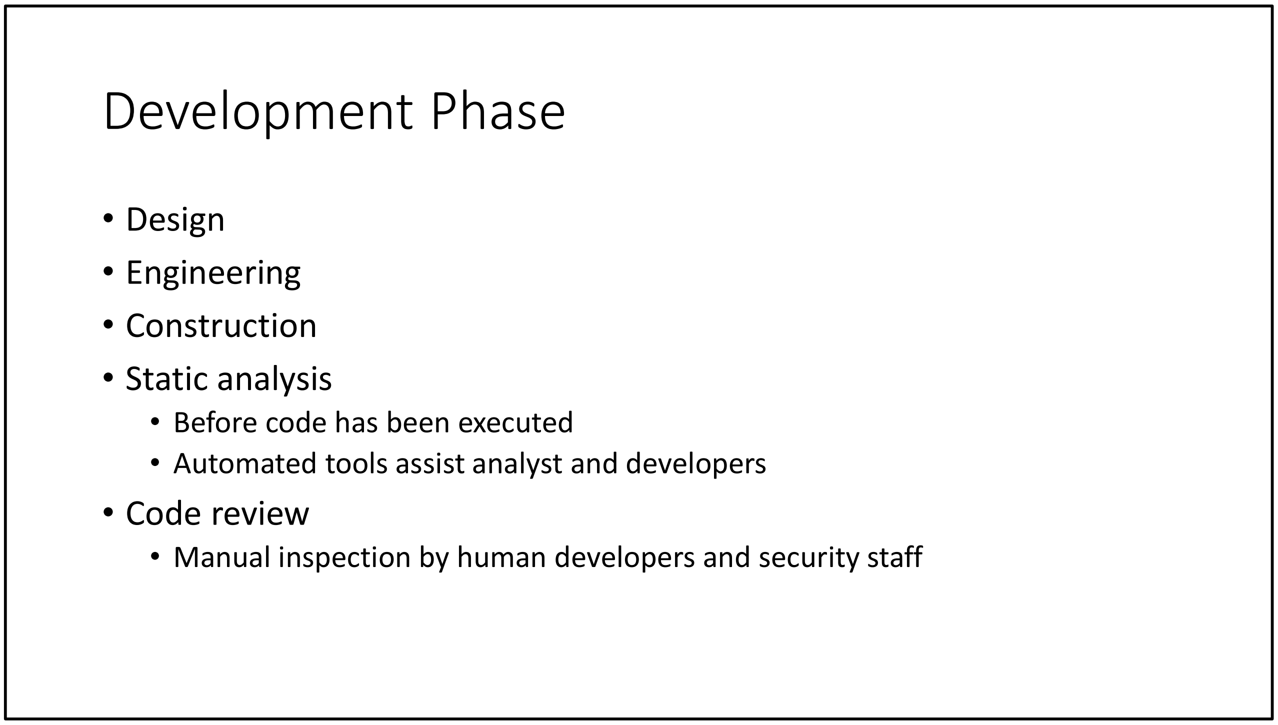
Secure Software Development

Given a scenario, analyze the output from common vulnerability assessment tools.
• Software assessment tools and techniques
- Static analysis
- Fuzzing
Objective 2.2 Explain software assurance best practices.
• Software development life cycle (SDLC) integration
• Software assessment methods
- User acceptance testing
- Stress test application
- Security regression testing
- Code review
• Static analysis tools



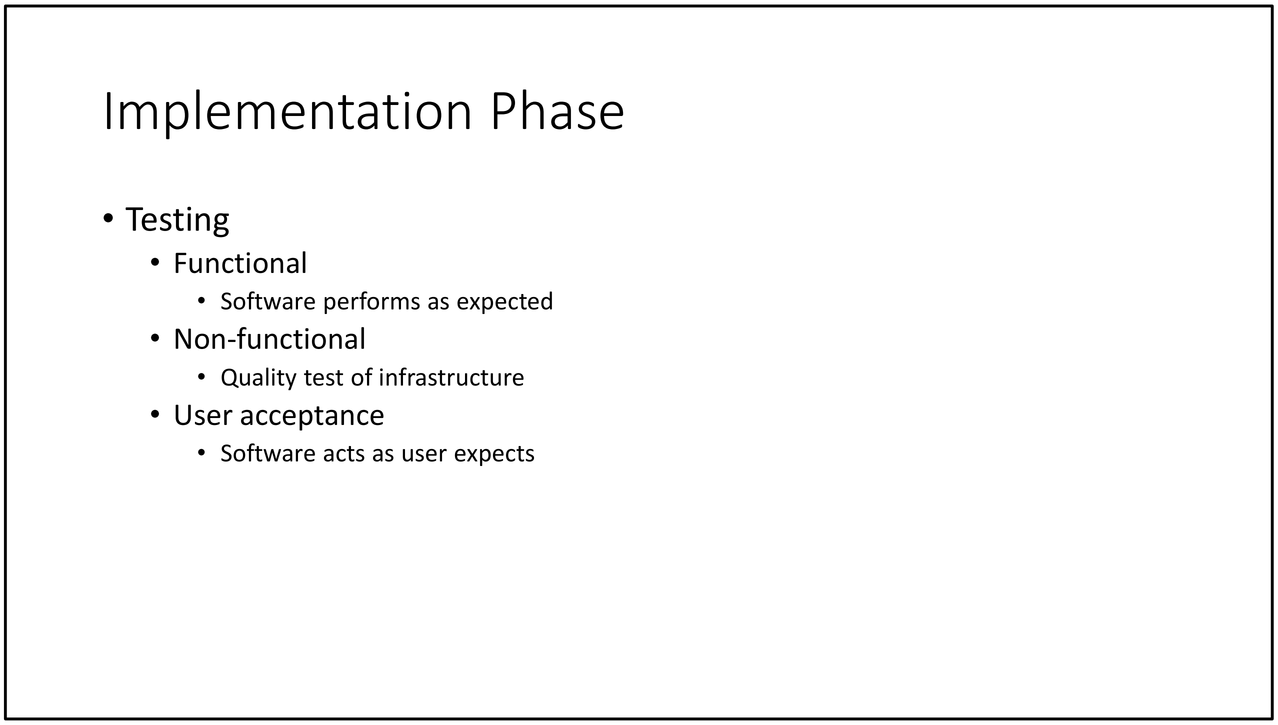
Coding Requirements

- Functional requirements
  - Inputs, processes, & outputs
- Non-functional requirements
  - Character inputs, input constraints, & system limitations
- Security requirements
  - Behaviors and characteristic to meet security levels
  - Interaction with other systems
  - Class includes functional & non-functional aspects



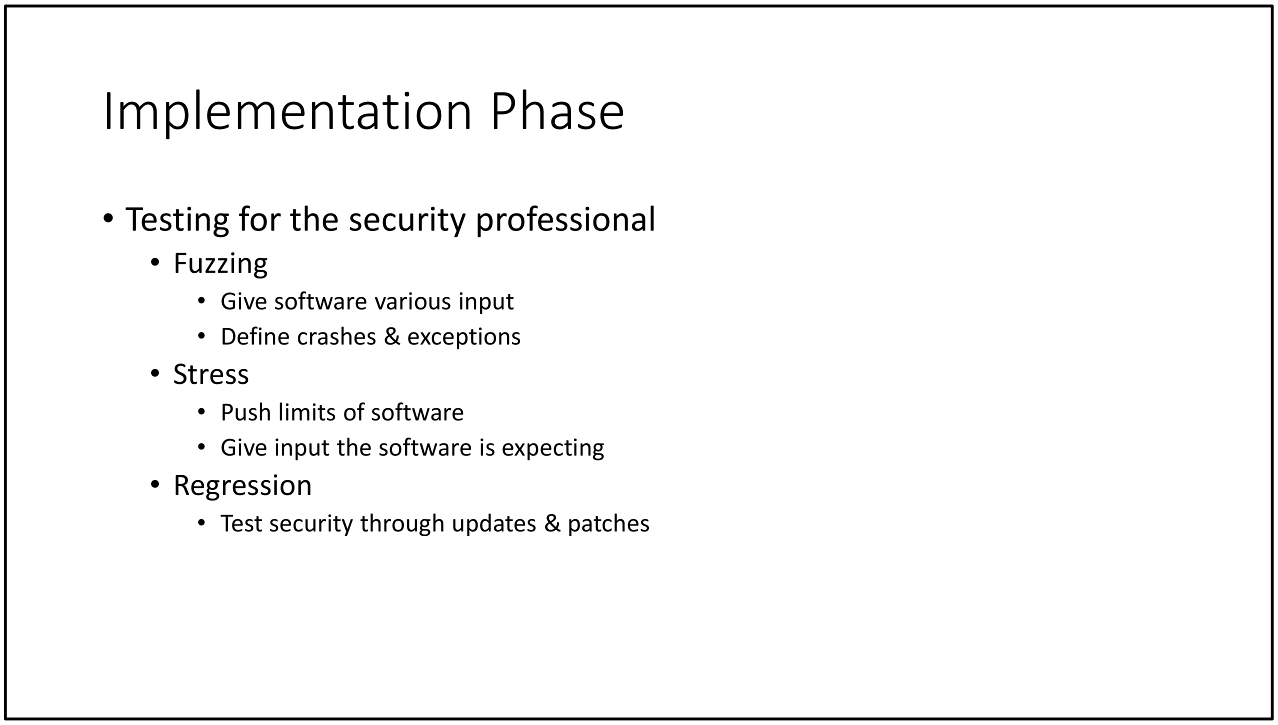
Development Phase

- Design
- Engineering
- Construction
- Static analysis
  - Before code has been executed
  - Automated tools assist analyst and developers
- Code review
  - Manual inspection by human developers and security staff



Implementation Phase

- Testing
  - Functional
    - Software performs as expected
  - Non-functional
    - Quality test of infrastructure
  - User acceptance
    - Software acts as user expects



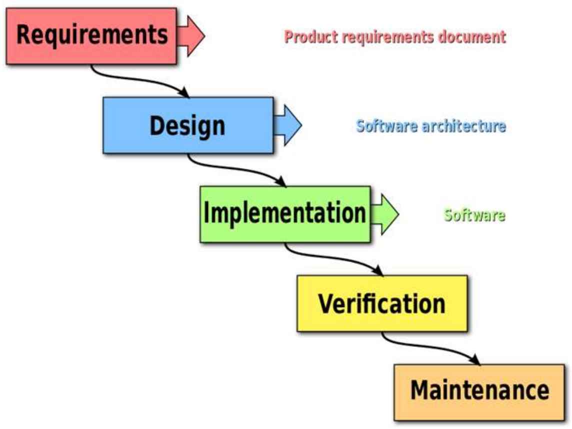
Implementation Phase

- Testing for the security professional
  - Fuzzing
    - Give software various input
    - Define crashes & exceptions
  - Stress
    - Push limits of software
    - Give input the software is expecting
  - Regression
    - Test security through updates & patches

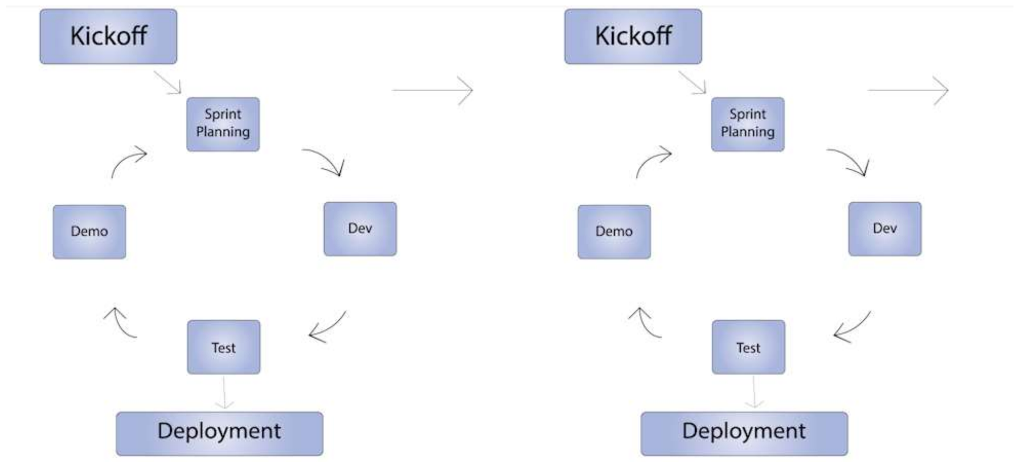
Operations & Maintenance Phase

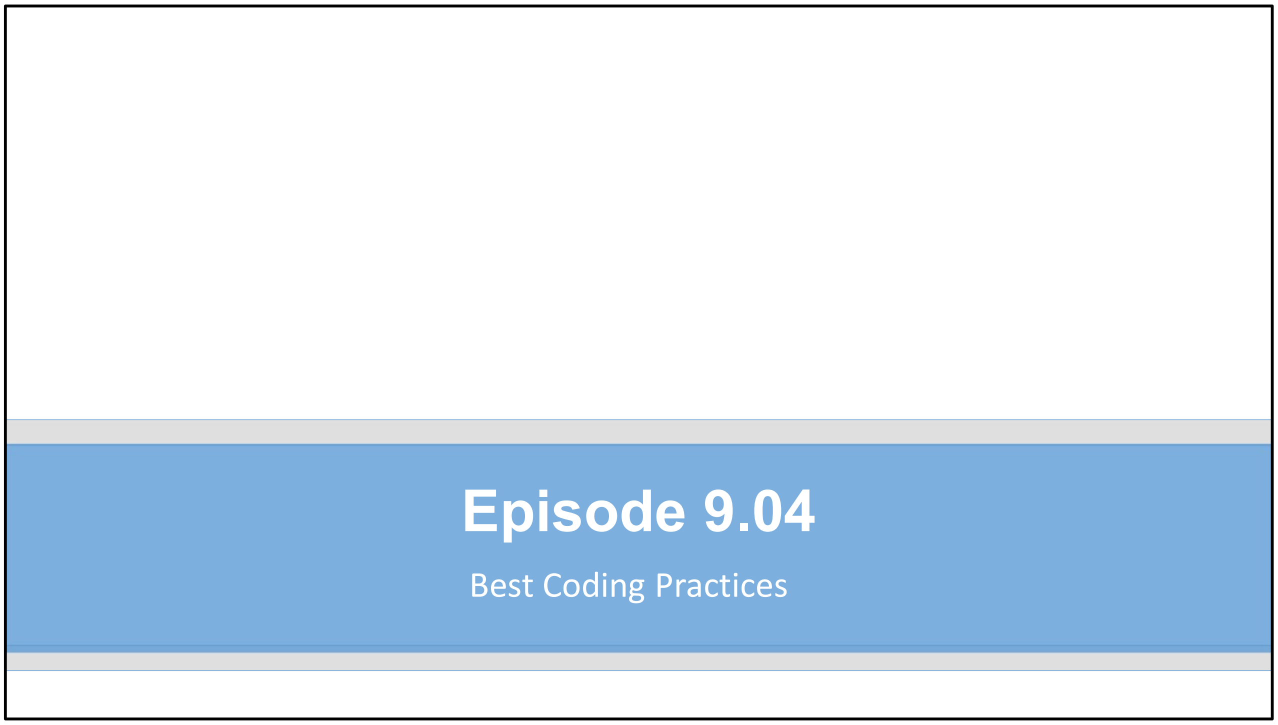
- Maintenance responsibility
  - Outline software security requirements for developers & implementers
  - Patches – how often, under what conditions, who is responsible
- Maintenance phase length
  - All software eventually gets retired
  - Define end-of-life

Waterfall Model



Agile Model

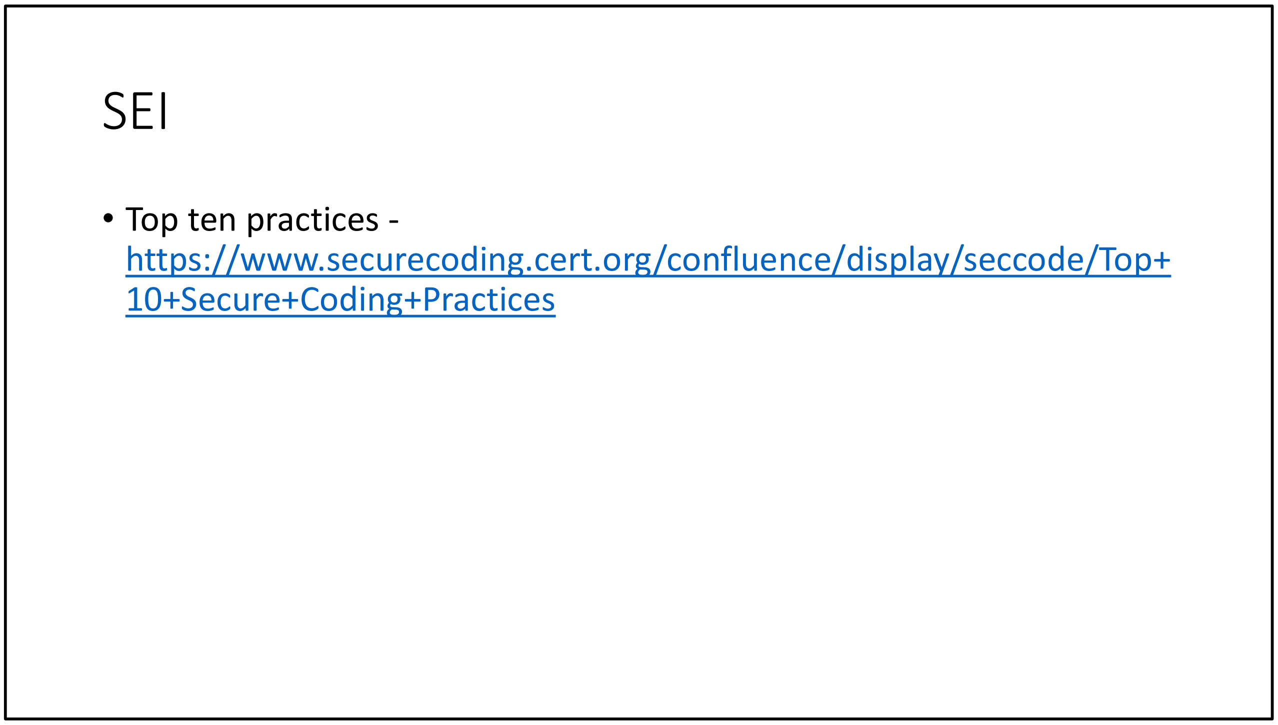

Kickoff
Sprint Planning
Dev
Test
Demo
Deployment
Kickoff
Sprint Planning
Dev
Test
Demo
Deployment



Episode 9.04

Best Coding Practices

Objective 2.2 Explain software assurance best practices.
- • Secure coding best practices
- - Input validation
- - Output encoding
- - Session management
- - Authentication
- - Data protection
- - Parameterized queries
- • Formal methods for verification of critical software



SEI

- Top ten practices - https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices

OWASP

- https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf
- TOC
- Input validation section