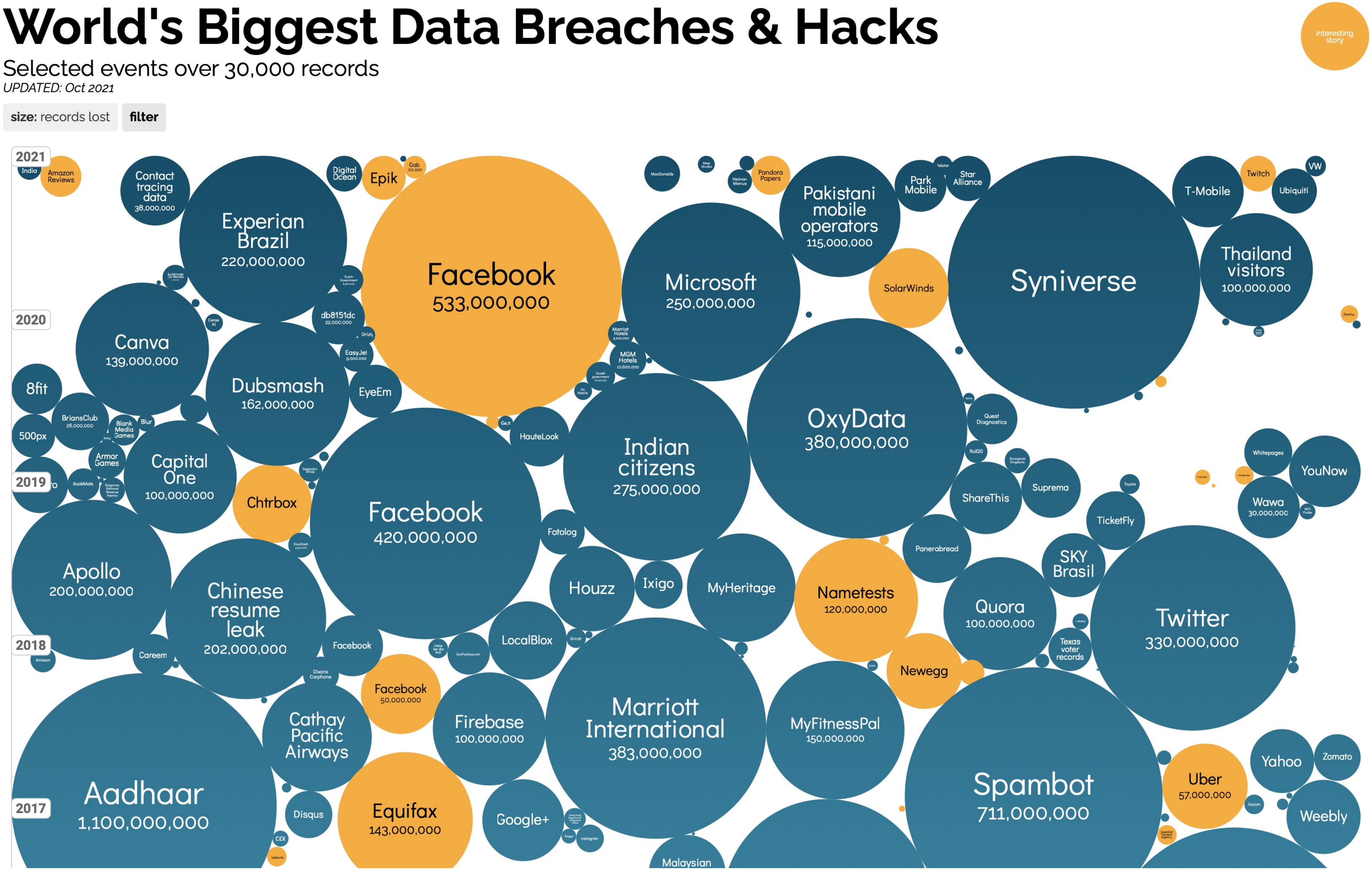
# LOG 8371E
# Software Quality Engineering

## Lecture 08:
## Software Security and Security Testing

Armstrong Foundjem Ph.D. — Winter 2024

# The Heartbleed Bug

- A vulnerability in the popular OpenSSL library
  - Allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of OpenSSL
- Recorded as CVE-2014-0160
- Associated CWE: CWE-119
- Exploits impacted many web applications (e.g., Canada Revenue Agency leaks SINs)



# World's Biggest Data Breaches & Hacks

Selected events over 30,000 records

*UPDATED: Oct 2021*

**size:** records lost **filter**

Source: https://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/

# Agenda

- **Vulnerabilities**
- **Security testing in the SDLC**
- **Static analysis (automated security review)**
- **Penetration tests**

**References:**

OWASP Testing Guide 4.2. 2020

Common Weakness Enumeration: https://cwe.mitre.org

Common Vulnerabilities and Exposures: https://cve.mitre.org/index.html

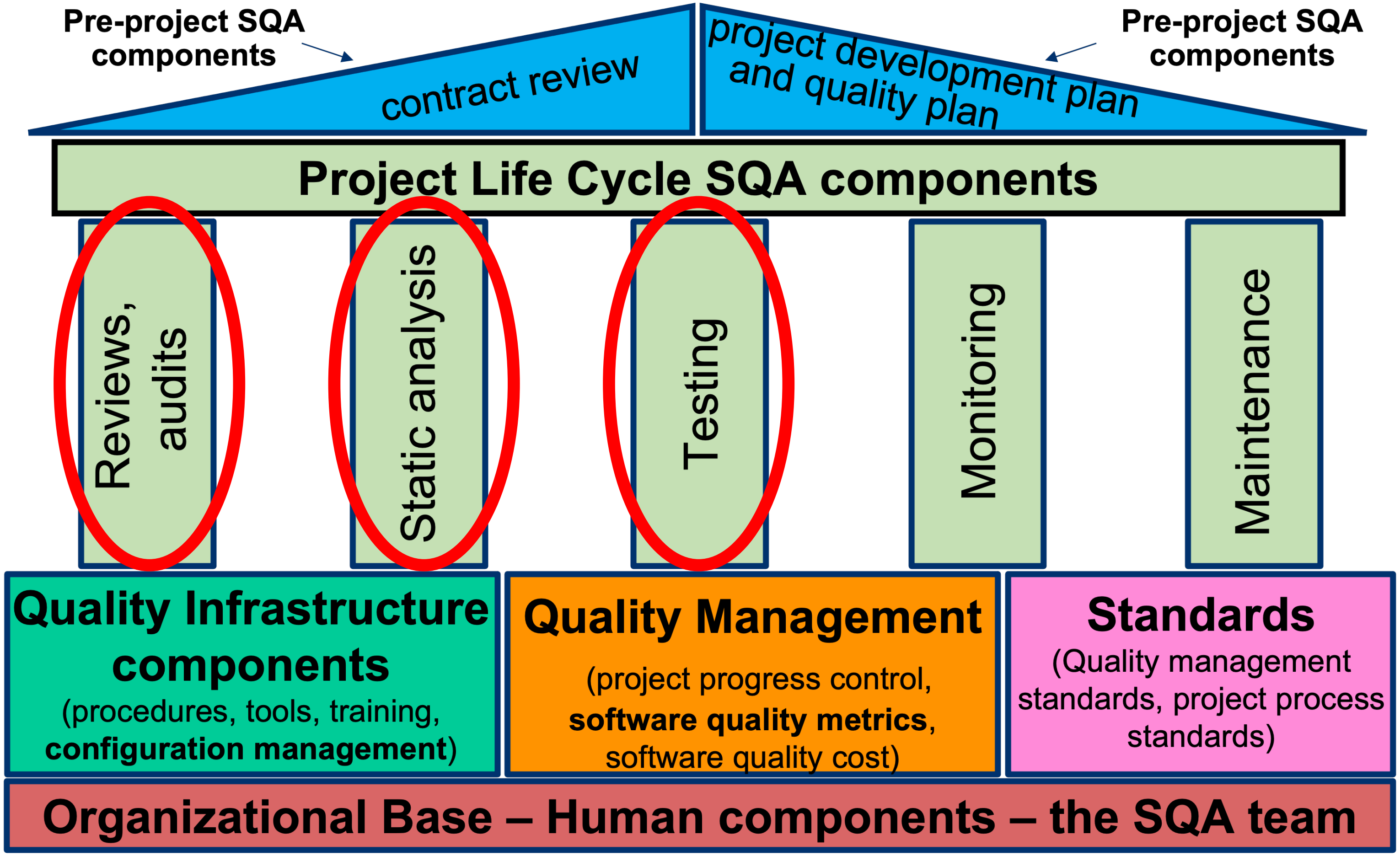
OWASP Top Ten: https://owasp.org/www-project-top-ten/

OWASP ZAP Documentation. https://www.zaproxy.org/docs

SonarQube documentation: https://docs.sonarqube.org

SonarCloud documentation: https://docs.sonarcloud.io

# Topic's topic in the SQA system



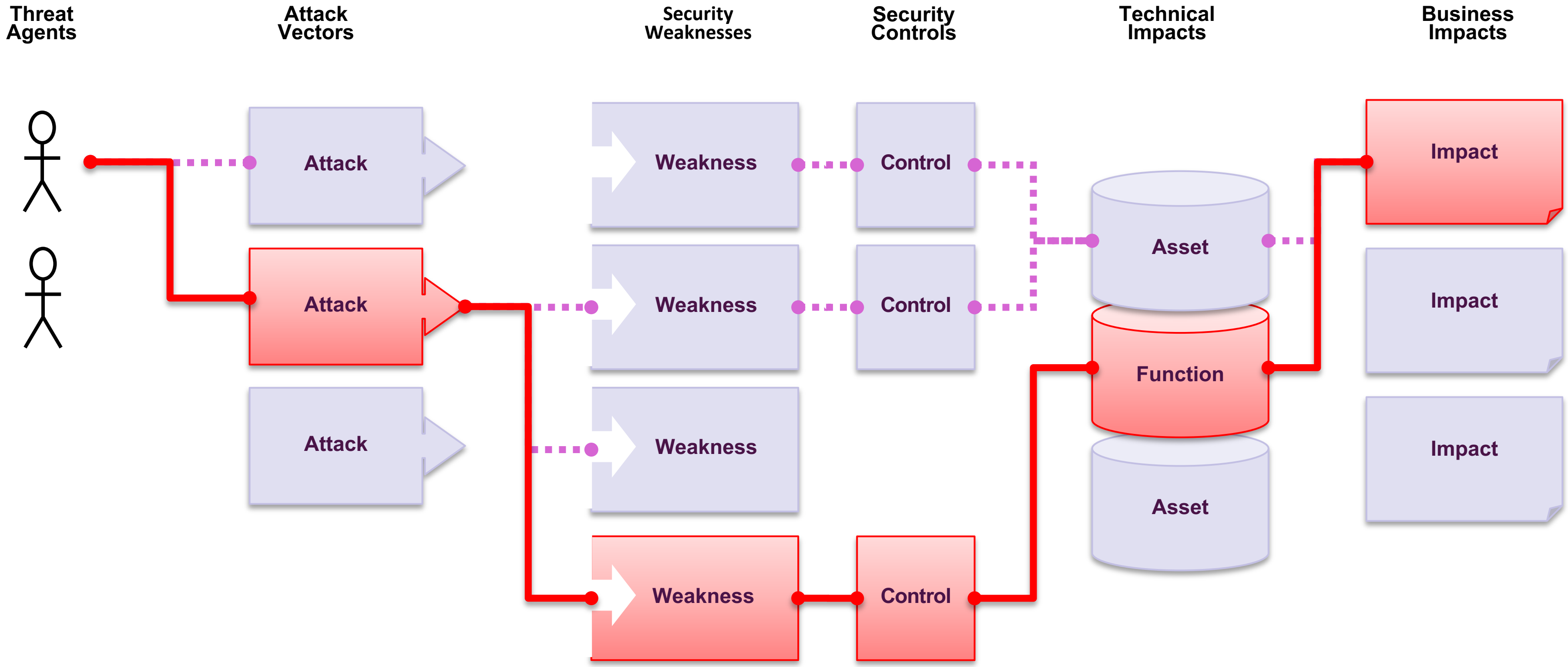
Pre-project SQA components
Pre-project SQA components
contract review
project development plan and quality plan
Project Life Cycle SQA components
Reviews, audits
Static analysis
Testing
Monitoring
Maintenance
Quality Infrastructure components
(procedures, tools, training, **configuration management**)
Quality Management
(project progress control, **software quality metrics**, software quality cost)
Standards
(Quality management standards, project process standards)
Organizational Base – Human components – the SQA team

# Vulnerabilities

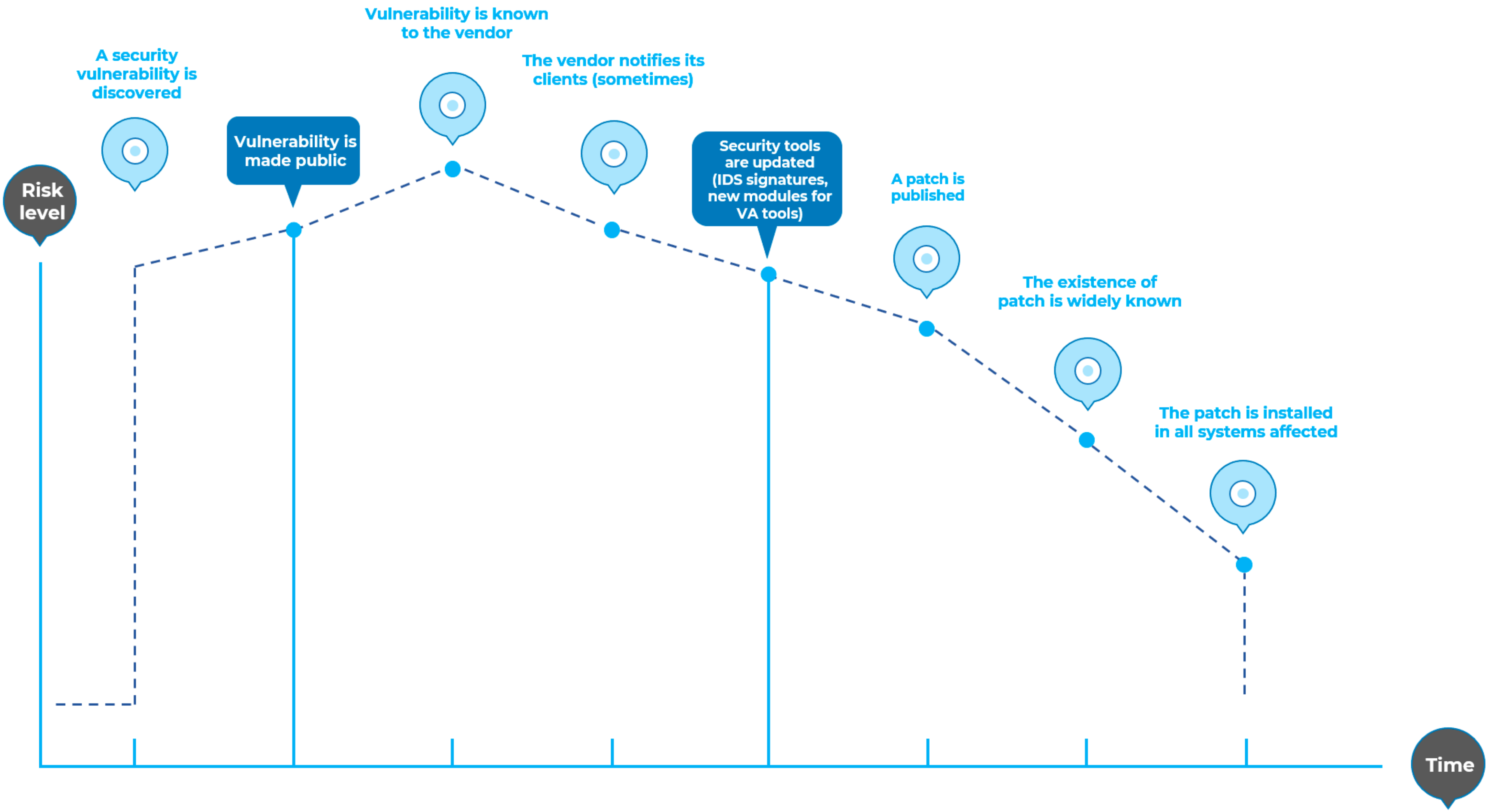
# Software security vulnerabilities

- Vulnerability is a weakness in design, implementation, or operation.
- Vulnerability becomes a problem if it becomes the point of an attack or an "exploit".
- Vulnerabilities and exploits can result in lost money, private data, or life!
- The system can be protected by identifying vulnerabilities through static analysis, penetration testing or code reviews.

# Attackers exploit the weaknesses of the application to do harm to business or organization



# Window of vulnerability


Risk level
A security vulnerability is discovered
Vulnerability is made public
Vulnerability is known to the vendor
The vendor notifies its clients (sometimes)
Security tools are updated (IDS signatures, new modules for VA tools)
A patch is published
The existence of patch is widely known
The patch is installed in all systems affected
Time

# Vulnerabilities causes

1. **Old, unpatched vulnerabilities.**
   - ✓A vulnerability can be identified and described, but developers do not correct it immediately.
2. **Human errors**
   - ✓Those include bugs, weak passwords, private data sharing.
3. **Malware**
   - ✓They cause minor problems but in large quantities and with a lot of variability.
4. **Insider abuse**
   - ✓The most dangerous vulnerabilities and what require internal security measures.
5. **Physical theft**
   - ✓Personal devices that must be physically protected too.

# CWE and CVE

- Common Weakness Enumeration (**CWE**)
  - A community-developed list of software and hardware weakness types;
  - Example: CWE-20 (Improper input validation)
  - A common language, a measuring stick for security tools
  - https://cwe.mitre.org
- Common Vulnerabilities and Exposures (**CVE**)
  - A list of publicly disclosed cybersecurity vulnerabilities
  - Example: CVE-2021-44228 (Apache Log4j2 vulnerability)
  - https://cve.mitre.org/index.html

# OWASP Top 10 Vulnerabilities (2017)

A1 – Injection

A2 – Broken Authentication and Session Management

A3 – Cross-Site Scripting (XSS)

A4 – Broken Access Control

A5 – Security Misconfiguration

A6 – Sensitive Data Exposure

A7 – Insufficient Attack Protection

A8 – Cross-Site Request Forgery (CSRF)

A9 – Using Components with Known Vulnerabilities

A10 – Underprotected APIs

Source: https://owasp.org/www-project-top-ten/2017/Top_10

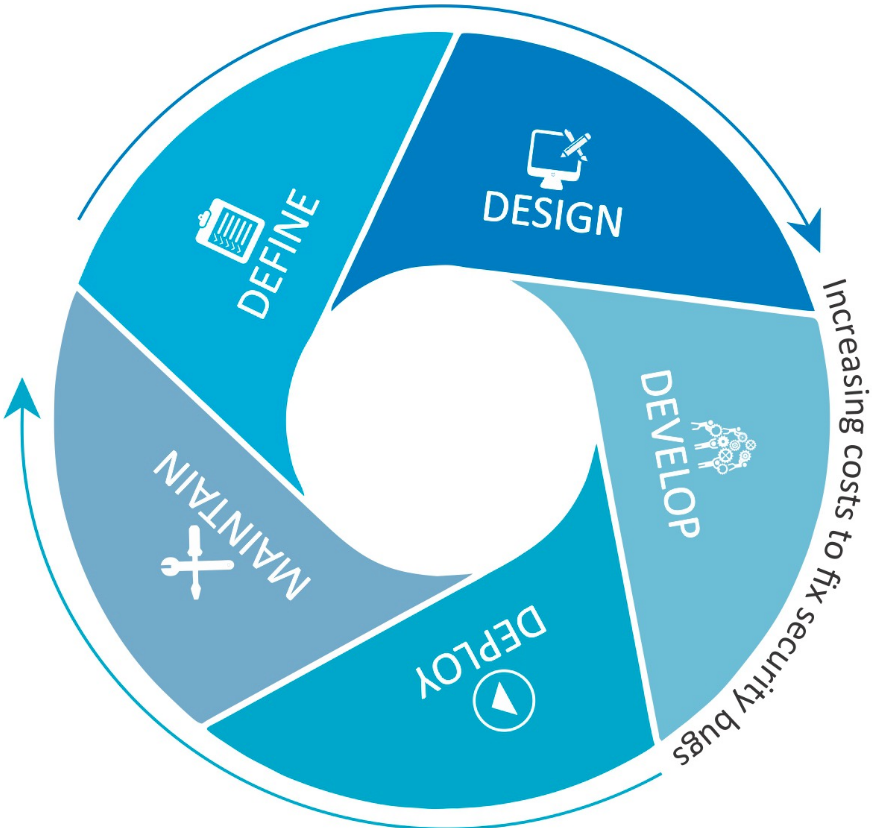
# Security Testing

# Security tests

- Security testing is the process to ensure that mechanisms established to ensure security are not defective.
- The success of security tests does not imply the absence of security defects.
- Security testing controls security requirements.
  - ✓ Confidentiality, integrity, non-repudiation, accountability, authenticity.

# When to test?

- Ensure that security is an integral part of the development process
- Consider security tests in each phase of SDLC
- Test early and test often
  - When a bug is detected early within the SDLC it can be addressed faster and at a lower cost



Software Development Life Cycle (SDLC)

# Testing techniques

- **Manual inspections & reviews**
  - Manually examine the security implications of people, policies, processes, technical decisions (e.g., designs)
- **Threat modeling**
  - Risk assessment for applications
- **Source code review**
  - Manually check the source code for security issues
- **Static analysis (automated code review)**
  - Automatically check the code to find vulnerabilities according to known rules or templates
- **Penetration testing** (ethical hacking)
  - Simulate a malicious attack and identify the degree of sensitivity of the vulnerabilities

# Manual inspections & reviews

- **Definition**
  - Manually examine the security implications of people, policies, processes;
  - Also include inspection of technical decisions (e.g., architectural designs);
  - Usually conducted by analyze documentation or performing interviews with designers or system owners
- **Advantages**
  - Can be applied to a variety of situations;
  - Flexible;
  - Promotes teamwork;
  - Early in the SDLC;
- **Disadvantages**
  - Can be time consuming (analyzing docs, interviews);
  - Supporting material not always available
  - Requires significant human thought and skill to be effective

# Threat modeling

- **Definition**
  - Risk assessment for applications;
  - Help system designers think about the security threats that their systems and applications might face;
  - Enables the designer to develop mitigation strategies for potential vulnerabilities
- **The NIST 800-30 standard approach**
  - **Decomposing the application** – use a process of manual inspection to understand how the application works, its assets, functionality, and connectivity.
  - **Defining and classifying the assets** – classify the assets into tangible and intangible assets and rank them according to business importance.
  - **Exploring potential vulnerabilities -** whether technical, operational, or managerial.
  - **Exploring potential threats** – develop a realistic view of potential attack vectors from an attacker's perspective by using threat scenarios or attack trees.
  - **Creating mitigation strategies** – develop mitigating controls for each of the threats deemed to be realistic.

# Threat modeling

- **Advantages**
  - Practical attacker view of the system
  - Flexible;
  - Early in the SDLC;
- **Disadvantages**
  - Good threat models don’t automatically mean secure software

# Source code review

- **Definition**
  - Manually check the source code for security issues;
  - Almost all security experts agree that there is no substitute for actually looking at the code;
  - Many serious security vulnerabilities are hard to be detected with any other form of analysis or testing, such as concurrency problems, flawed business logic, access control problems, and cryptographic weaknesses.
- **Advantages**
  - Completeness and effectiveness;
  - Accuracy;
  - Fast (for competent reviewers)
- **Disadvantages**
  - Requires highly skilled security aware developers
  - Can miss issues in compiled libraries
  - Cannot detect runtime errors easily

# Static analysis (automated code review)

- **Definition**
  - Using static analysis tools to automatically check the code to find vulnerabilities according to known rules or templates;
  - Useful in determining security issues due to coding errors.
- **Advantages**
  - Fast.
- **Disadvantages**
  - Cannot identify issues due to flaws in the design;
  - False positives (significant manual effort required to validate the results).

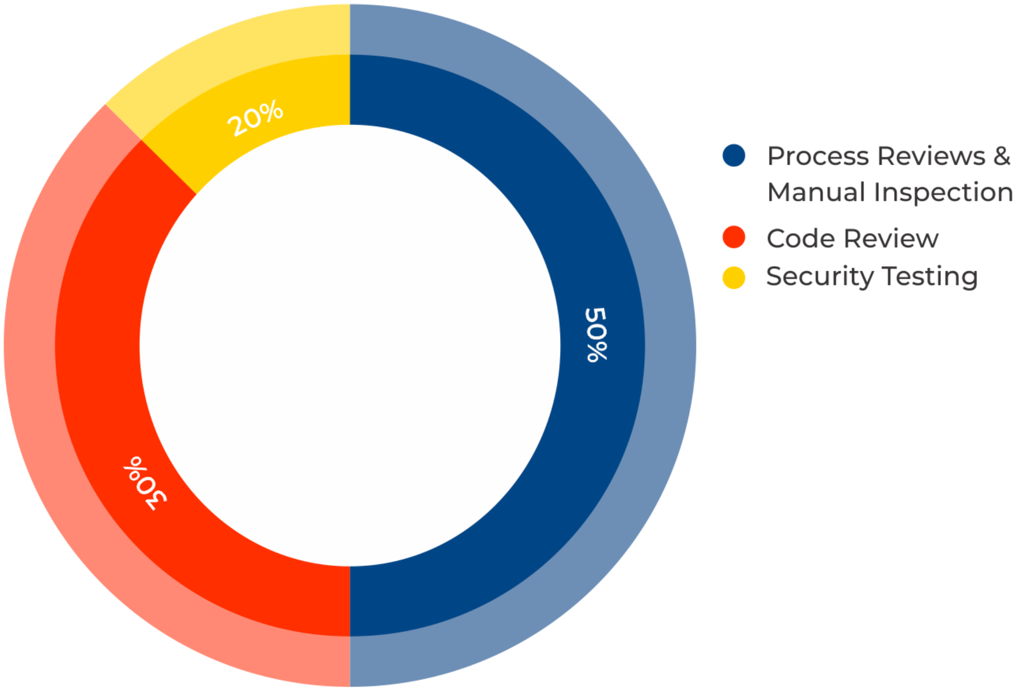
# Penetration testing

- **Definition**
  - Simulate a malicious attack and identify the degree of sensitivity of the vulnerabilities
  - Also known as ethical hacking.
  - Without knowing the inner workings of the target itself (black-box testing).
  - The tester is usually given one or more valid accounts on the system.
- **Advantages**
  - Fast;
  - Lower skill-set requirement than source code review
  - Tests the code that is actually being exposed
- **Disadvantages**
  - Late in the SDLC;
  - Front-impact testing only;
  - Gary McGraw:"*a penetration test can only identify a small representative sample of all possible security risks in a system*".

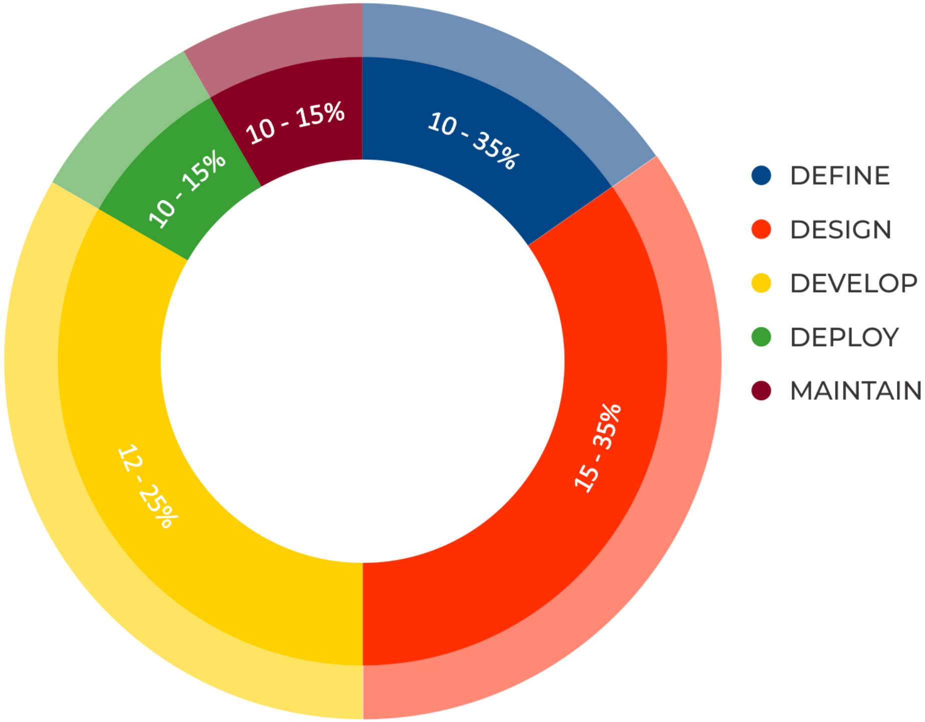
# What techniques to use?

- **No Silver Bullet**
  - No right or wrong answer to the question of exactly which techniques should be used
- Penetration testing is "**too tittle too late**"
  - Historically, many companies have adopted a single approach (often penetration testing)
- A **balanced approach** should cover testing in all SDLC phases
  - Include several techniques, from manual reviews to technical testing

# Balanced representations of testing effort



Proportion of test effort in SDLC



Proportion of test effort by test technique

# Why automated black-box testing may not be effective?

## Example: Bad Cryptography

- Imagine that a developer decided to write a simple cryptography algorithm to sign a user in from site A to site B automatically. In their wisdom, the developer decides that if a user is logged into site A, then they will generate a key using an MD5 hash function that comprises: Hash {username : date}.
- When a user is passed to site B, they will send the key on the query string to site B in an HTTP redirect. Site B independently computes the hash, and compares it to the hash passed on the request. If they match, site B signs the user in as the user they claim to be.
- Is there a security issue?
- Can automated black-box testing detect the issue? If not, what approaches can?

# Derive and elicit security requirements

High-level testing objectives:

- proving confidentiality, integrity, and availability of the data as well as the service;
- validate that security controls are implemented with few or no vulnerabilities

# Derive and elicit security requirements

Two types of security requirements:

- **Positive (functional)** Requirements: These describe the appropriate functionality of the system. Examples:
  - ✓The application will lockout the user after six failed log on attempts
  - ✓Do not show specific validation errors to the user as a result of a failed log on
- **Negative (risk-oriented)** Requirements: These describe what the application should not do. Example:
  - ✓The application should not allow for the data to be altered or destroyed.
  - ✓The application should not be compromised or misused for unauthorized financial transactions by a malicious user.
- **Which is more difficult?**

# Derive security requirements from use and misuse cases

- **Use cases** (from a normal user point of view)
  - Describe what the application is supposed to do and how
  - Show the interactions of actors and their relations
- **Misuse or abuse cases** (from an attacker point of view)
  - Describe unintended and malicious use scenarios
  - Describe scenarios of how an attacker could misuse and abuse the application

# Derive security requirements from use and misuse cases

Example: for the case of authentication

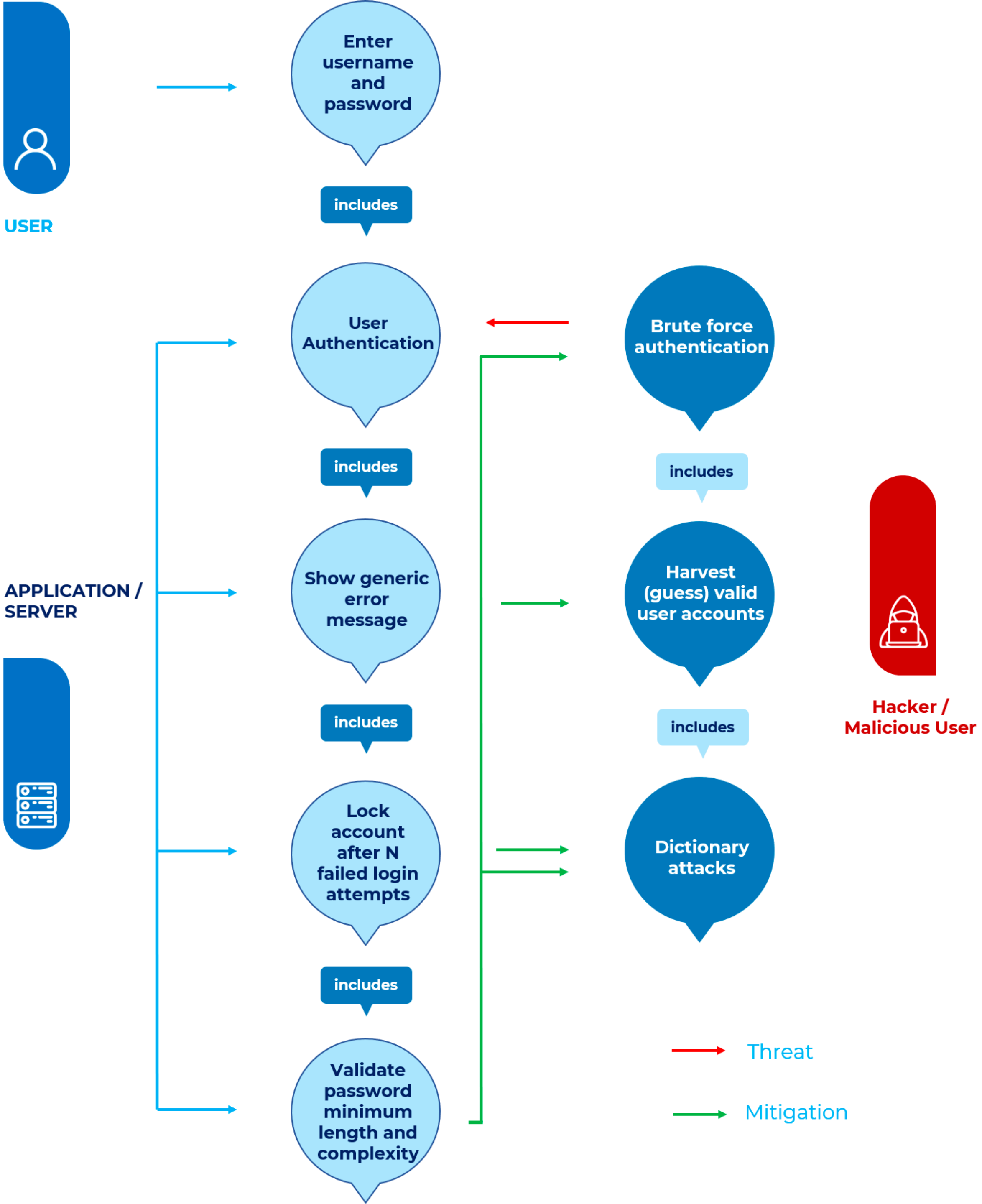
- **Step 1: Describe the functional scenario**
  - User authenticates by supplying a username and password.
  - The application grants access to users based upon authentication of user credentials by the application.
  - The application provides specific errors to the user when validation fails.
- **Step 2: Describe the negative scenario**
  - Attacker breaks the authentication through a brute force or dictionary attack of passwords and account harvesting vulnerabilities in the application.
  - The validation errors provide specific information to an attacker that is used to guess which accounts are valid registered accounts (usernames).
  - The attacker then attempts to brute force the password for a valid account.

# Derive security requirements from use and misuse cases

Example: for the case of authentication

- **Step 3: Describe functional and negative scenarios with use and misuse case**
  - The functional scenario consists of the user actions (entering a username and password) and the application actions (authenticating the user and providing an error message if validation fails).
  - The misuse case consists of the attacker actions, i.e. trying to break authentication by brute forcing the password via a dictionary attack and by guessing the valid usernames from error messages.
  - By graphically representing the threats to the user actions (misuses), it is possible to derive the countermeasures as the application actions that mitigate such threats.

# Use and misuse case (authentication)



USER

Enter username and password

includes

User Authentication

includes

Show generic error message

includes

Lock account after N failed login attempts

includes

Validate password minimum length and complexity

APPLICATION / SERVER

Brute force authentication

includes

Harvest (guess) valid user accounts

includes

Dictionary attacks

Hacker / Malicious User

Threat

Mitigation

# Derive security requirements from use and misuse cases

Example: for the case of authentication

- **Step 4: Elicit security requirements**
  - Passwords requirements must be aligned with the current standards for sufficient complexity.
  - Accounts must be to locked out after five unsuccessful log in attempts.
  - Log in error messages must be generic.

# Integrating security tests in development and testing workflows

- Security testing in the development workflow
  - Source code analysis (static)
    - Potential vulnerabilities
    - Compliance with secure coding standards
  - Security unit tests (dynamic)
    - Components function as expected
  - Secure code review
    - Usually led by senior developers

# Integrating security tests in development and testing workflows

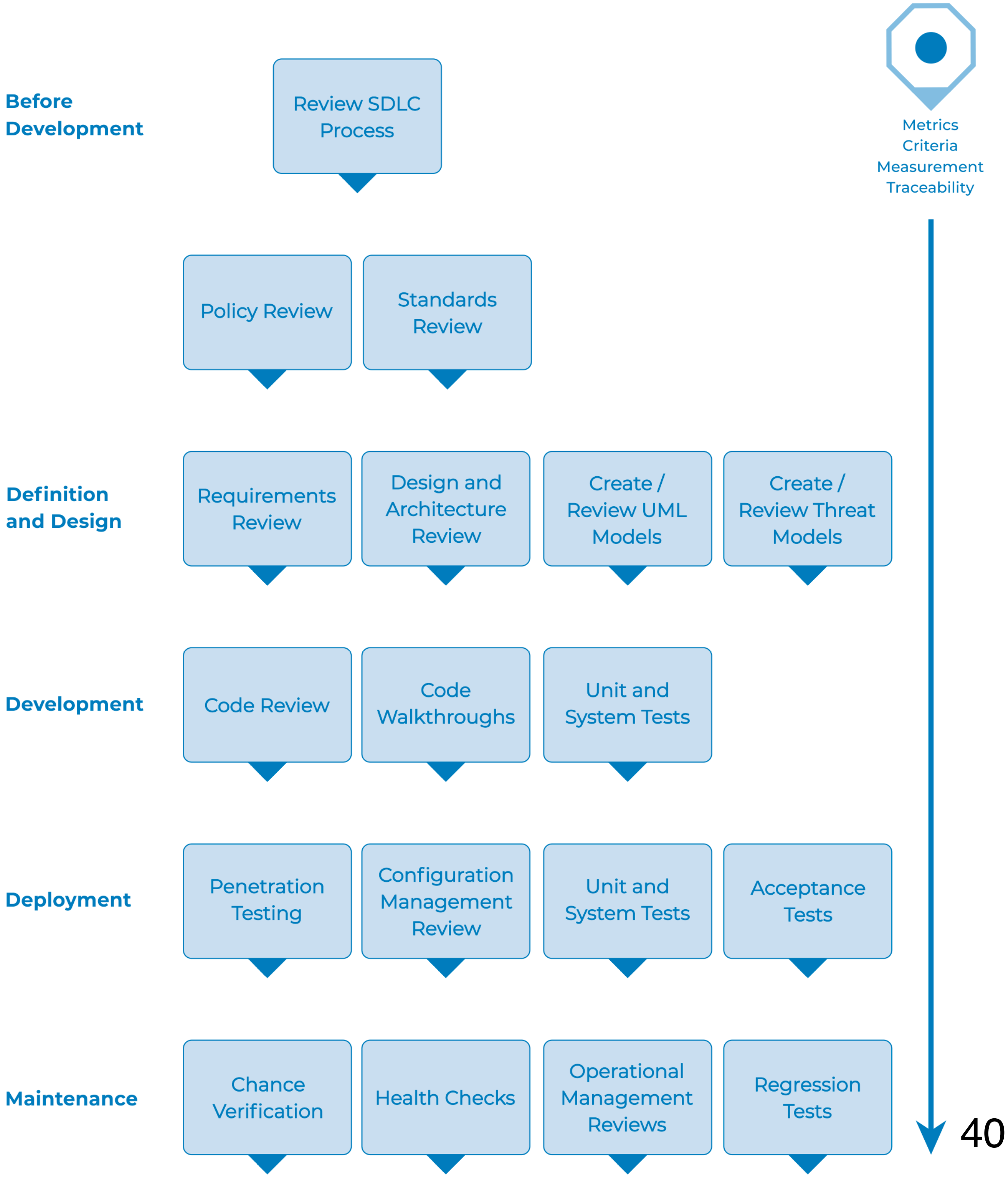
- Security testing in the test workflow
  - Integration test
    - identifying vulnerabilities due to integration of components
  - System test, user acceptance test, and operation tests
    - Most representative of the deployment configuration

# Security test data analysis and reporting

- **Defining security testing metrics**
  - E.g., the number of vulnerabilities found with security tests
  - Used for risk analysis and management processes, compare against a baseline.
- **Reporting testing results**
  - a categorization of each vulnerability by type;
  - the security threat that each issue is exposed to;
  - the root cause of each security issue, such as the bug or flaw;
  - each testing technique used to find the issues;
  - the remediation, or countermeasure, for each vulnerability; and
  - the severity rating of each vulnerability (e.g., high, medium, low, or CVSS score).

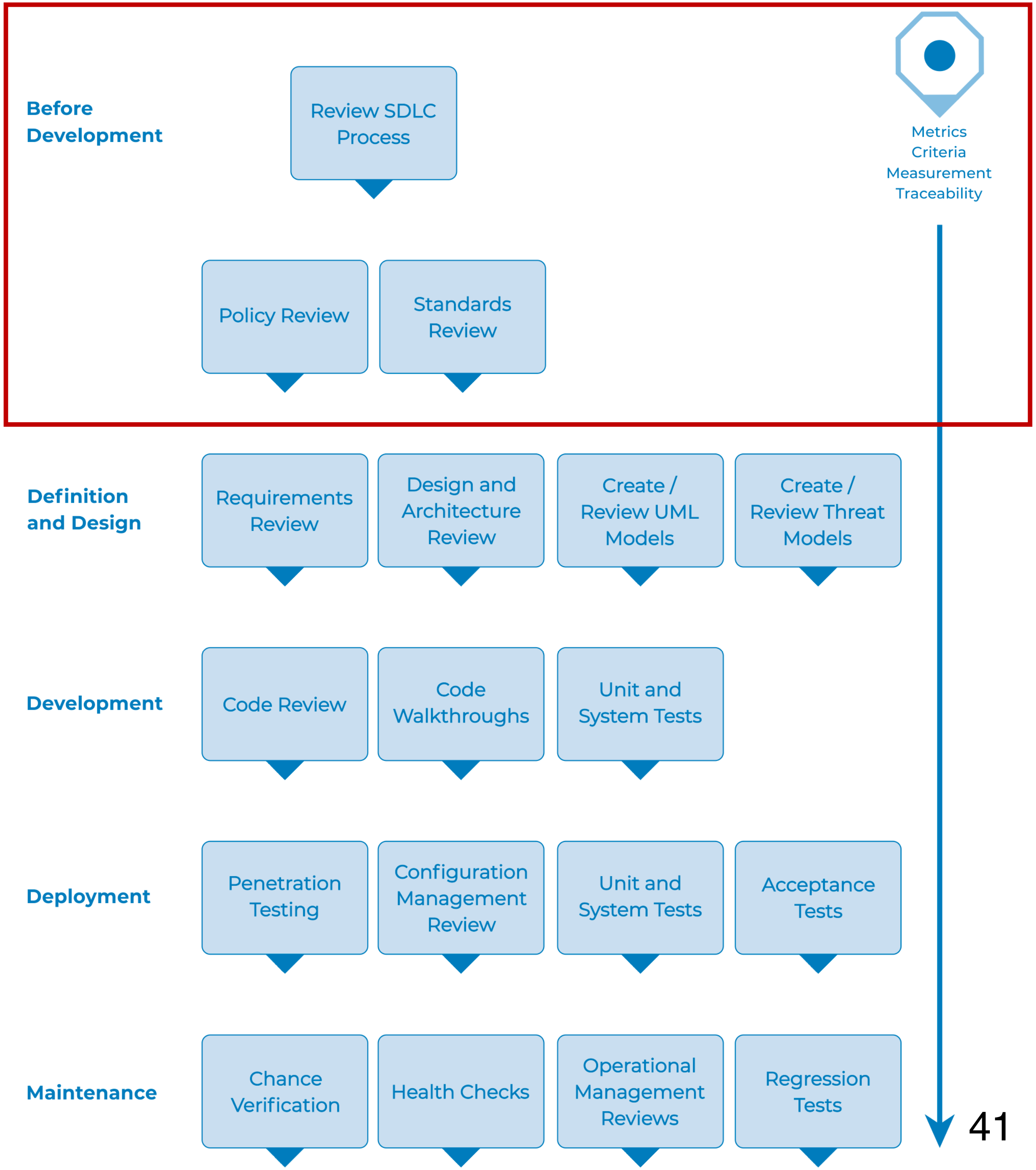
# OWASP Testing Framework

A reference framework that comprises techniques and tasks that are appropriate at various phases of the SDLC



# OWASP Testing Framework

A reference framework that comprises techniques and tasks that are appropriate at various phases of the SDLC

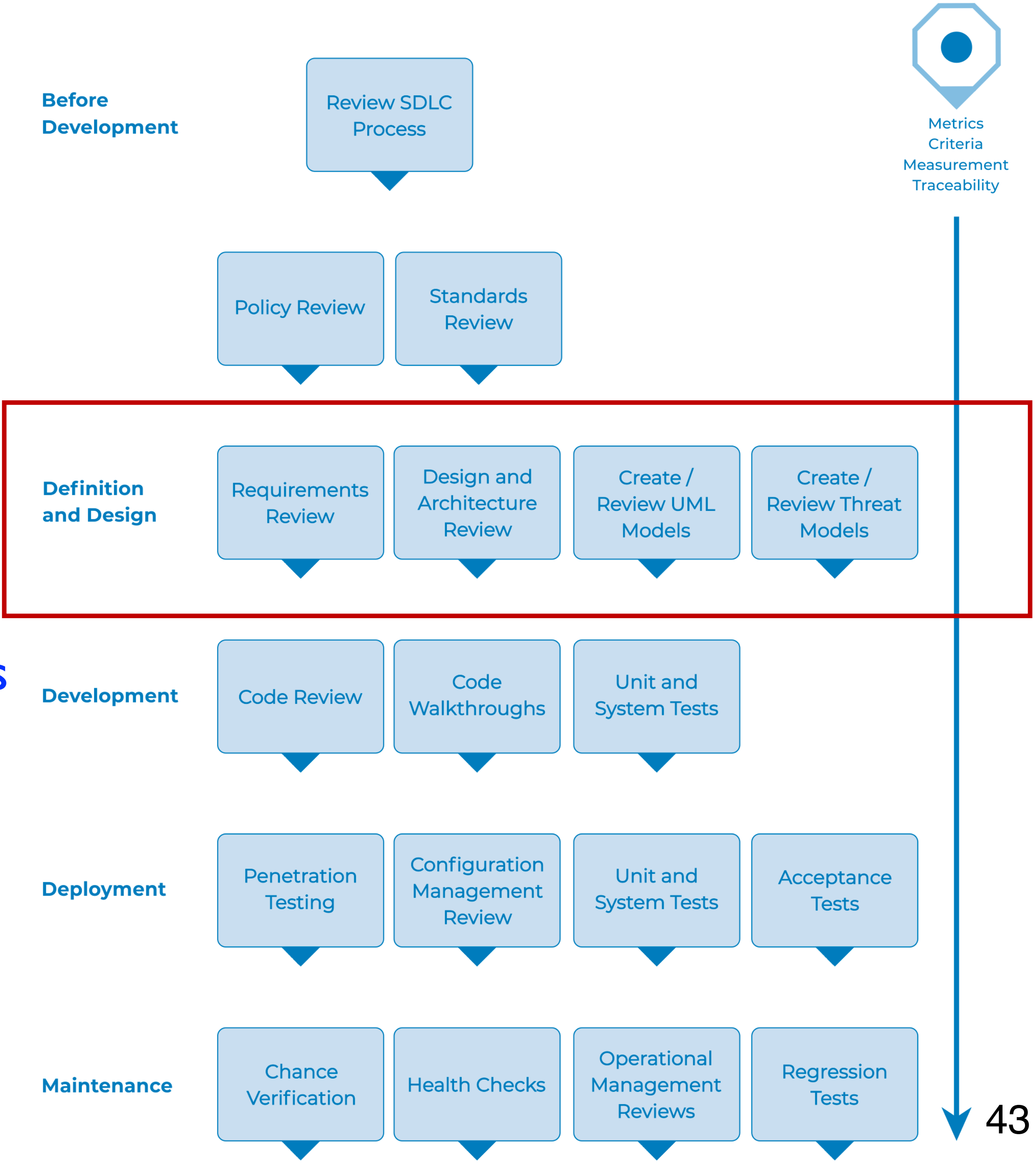


# Phase 1: Before development begins

- **Define a SDLC**
  - An adequate SDLC must be defined where security is inherent at each stage
- **Review policies and standards**
  - Ensure that there are appropriate security policies, standards, and documentation in place
- **Develop measurement and metrics criteria and ensure traceability**
  - Defining criteria that need to be measured, which provides visibility into defects in both the process and product.

# OWASP Testing Framework

A reference framework that comprises techniques and tasks that are appropriate at various phases of the SDLC



Metrics
Criteria
Measurement
Traceability

**Before Development**
- Review SDLC Process
- Policy Review
- Standards Review

**Definition and Design**
- Requirements Review
- Design and Architecture Review
- Create / Review UML Models
- Create / Review Threat Models

**Development**
- Code Review
- Code Walkthroughs
- Unit and System Tests

**Deployment**
- Penetration Testing
- Configuration Management Review
- Unit and System Tests
- Acceptance Tests

**Maintenance**
- Chance Verification
- Health Checks
- Operational Management Reviews
- Regression Tests

# Phase 2: During definition and design

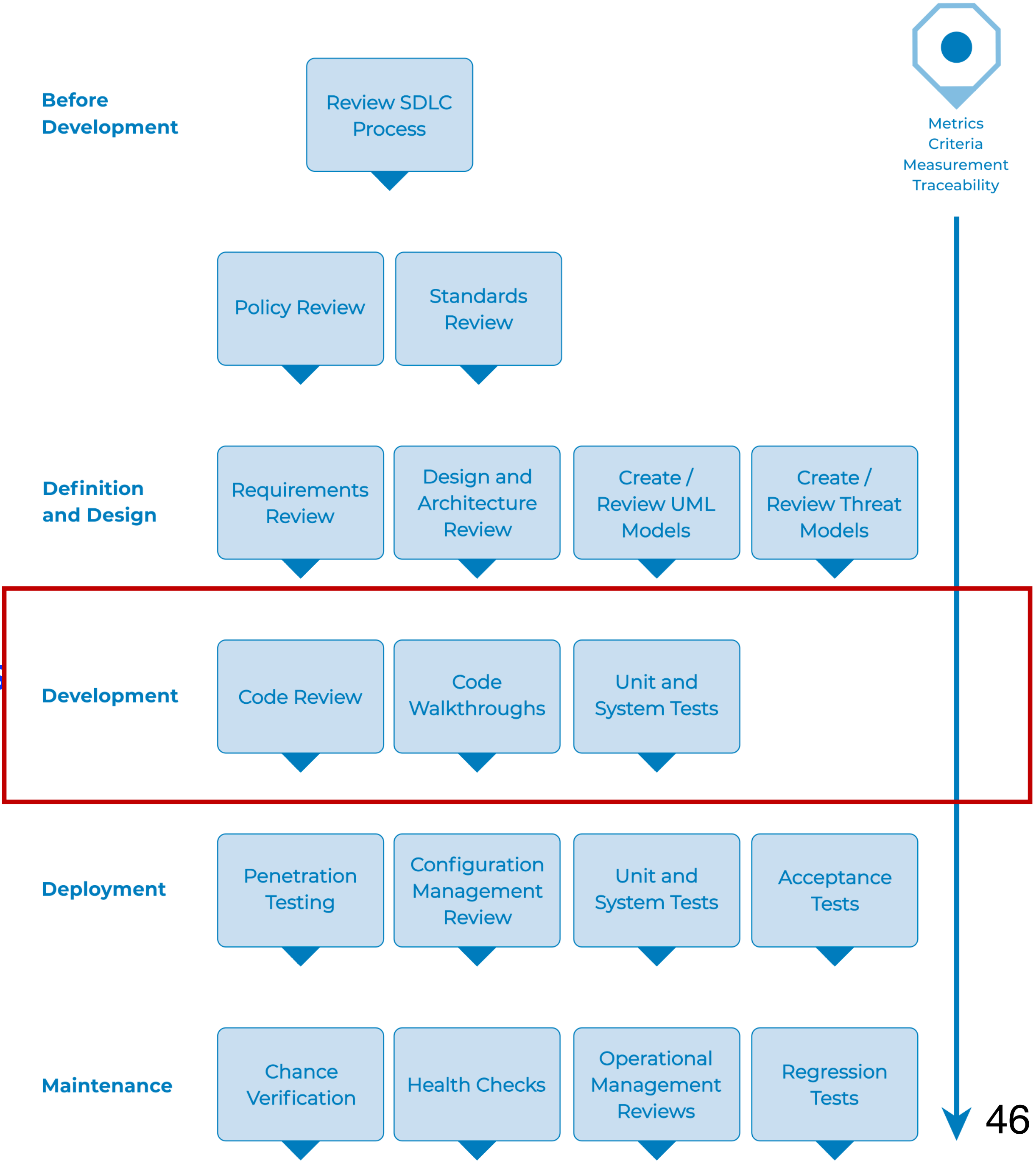
- **Review security requirements**
  - Security requirements define how an application works from a security perspective.
  - Security requirements should be unambiguous
    - “Users must be registered before they can get access to the whitepapers section of a website.”
- **Review design and architecture**
  - Ensure that the design and architecture enforce the appropriate level of security as defined in the requirements.
  - Cost-efficient to identify flaws and make changes.

# Phase 2: During definition and design (cont'd)

- **Create and review UML models**
  - Build UML models that describe how the application works.
  - UML models can help identify weaknesses.
- **Create and review threat models**
  - Develop realistic threat scenarios.
  - Analyze the design and architecture to ensure that these threats have been mitigated.
  - When identified threats have no mitigation strategies, revisit and modify the design and architecture.

# OWASP Testing Framework

A reference framework that comprises techniques and tasks that are appropriate at various phases of the SDLC

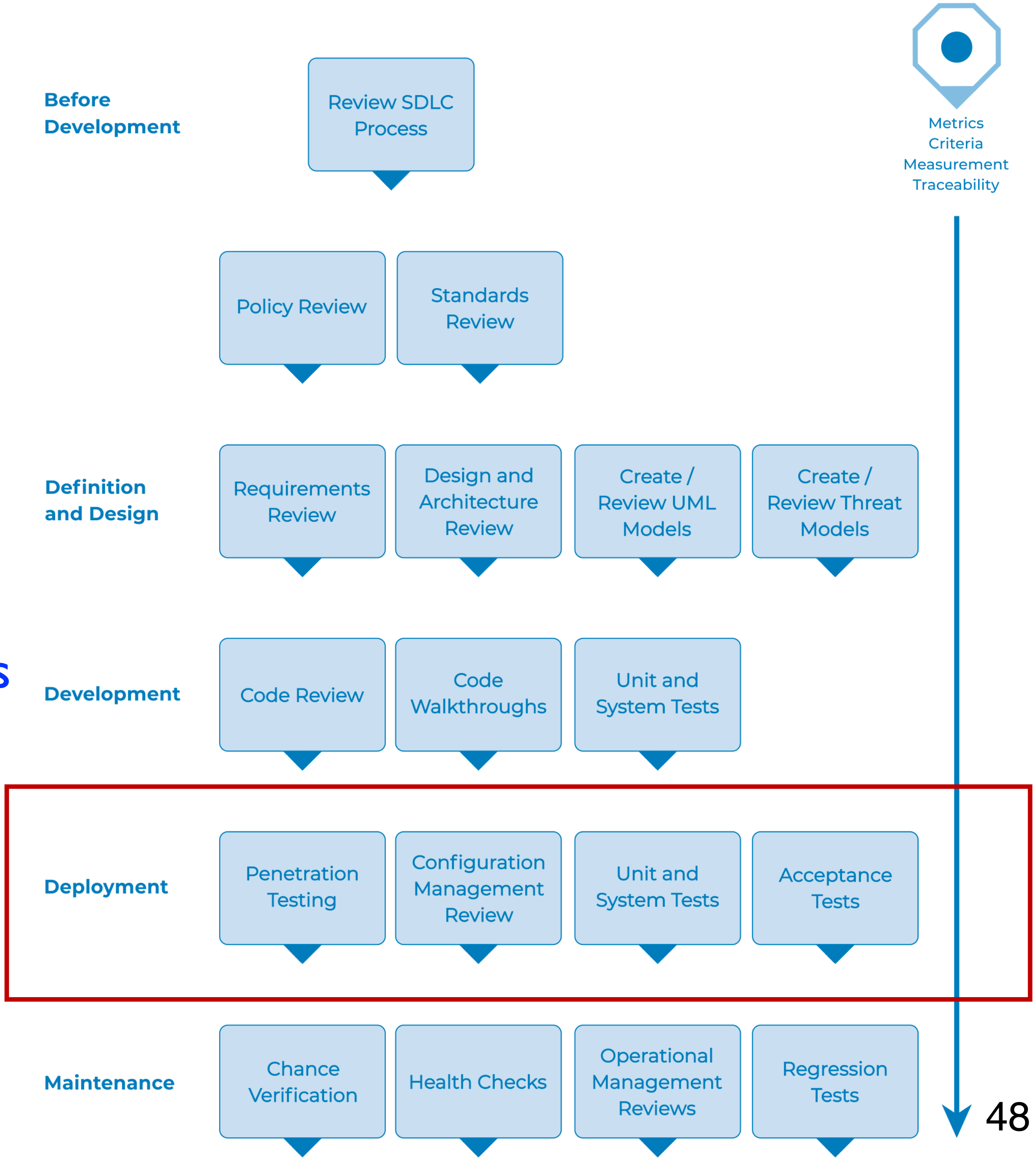


# Phase 3: During development

- **Code walkthrough**
  - The security experts perform a code walkthrough with the developers.
  - A code walkthrough is a high-level look at the code during which the developers explain the logic and flow of the implemented code.
- **Code reviews**
  - The security experts examine the actual code for security defects:
    - Business requirements for availability, confidentiality, integrity
    - Standard vulnerabilities (e.g., OWASP 10 10 checklists)
    - Language-specific issues (e.g., Microsoft security coding checklist for ASP.NET)
    - Industry-specific requirements (e.g., Sarbanes-Oxley 404)
- **Unit and system tests**

# OWASP Testing Framework

A reference framework that comprises techniques and tasks that are appropriate at various phases of the SDLC



Metrics
Criteria
Measurement
Traceability

**Before Development**

- Review SDLC Process
- Policy Review
- Standards Review

**Definition and Design**

- Requirements Review
- Design and Architecture Review
- Create / Review UML Models
- Create / Review Threat Models

**Development**

- Code Review
- Code Walkthroughs
- Unit and System Tests

**Deployment**

- Penetration Testing
- Configuration Management Review
- Unit and System Tests
- Acceptance Tests

**Maintenance**

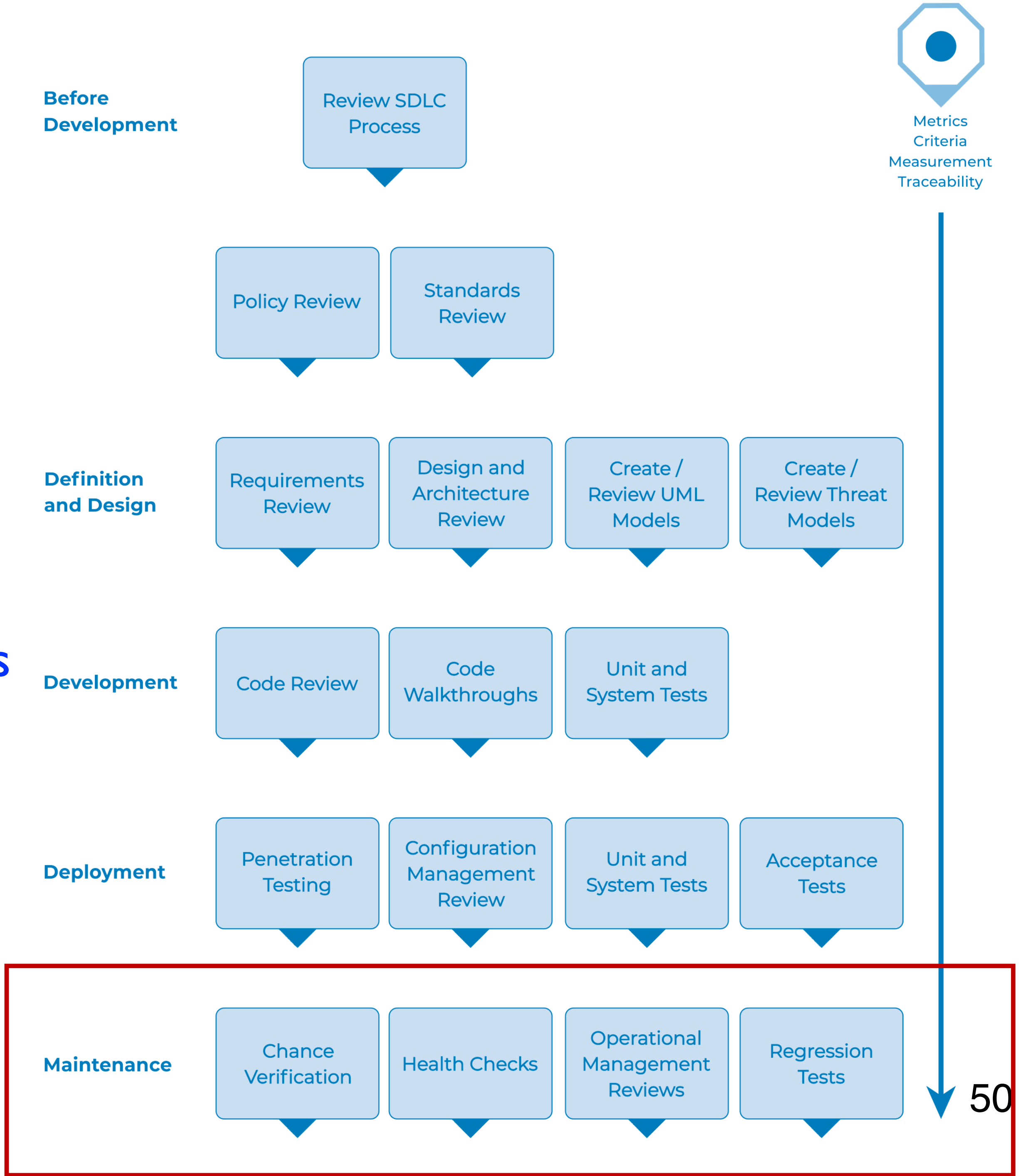
- Chance Verification
- Health Checks
- Operational Management Reviews
- Regression Tests

# Phase 4: During deployment

- **Application penetration testing**
  - Conducted after the application has been deployed.
  - Provides an additional check to ensure that nothing has been missed.
- **Configuration management testing**
  - Examination of how the infrastructure was deployed and secured.

# OWASP Testing Framework

A reference framework that comprises techniques and tasks that are appropriate at various phases of the SDLC



Metrics
Criteria
Measurement
Traceability

**Before Development**

Review SDLC Process

Policy Review

Standards Review

**Definition and Design**

Requirements Review

Design and Architecture Review

Create / Review UML Models

Create / Review Threat Models

**Development**

Code Review

Code Walkthroughs

Unit and System Tests

**Deployment**

Penetration Testing

Configuration Management Review

Unit and System Tests

Acceptance Tests

**Maintenance**

Chance Verification

Health Checks

Operational Management Reviews

Regression Tests

# Phase 5: During maintenance and operations

- **Conduct operational management reviews**
  - Review how the operational side of both the application and infrastructure is managed.
- **Conduct periodic health checks**
  - Monthly or quarterly health checks should be performed on both the application and infrastructure to ensure no new security risks have been introduced and that the level of security is still intact
- **Ensure change verification**
  - Each change is checked to ensure that the level of security has not been affected by the change

# Static analysis (automated security review)

# Static analysis

- Code scanning before deployment and execution to find vulnerabilities (and other bugs).
- Identifying vulnerabilities in the early stages of development is ideal.
- Techniques :
  - ✓ Data flow analysis: The collection of dynamic information in a static way; how the data is processed by the instructions.
  - ✓ Control Flow Graph: An abstract representation of the software that shows dependencies between instructions and code blocks.
  - ✓ Taint Analysis: Identifying variables "polluted" by users and plotting them to potential points of vulnerabilities.

# Static analysis

## Limitations

- **False positives**: The tool detects a vulnerability that is not really a problem.
- **False negatives**: A vulnerability exists, but it is not detected by the tool.

## Selecting tools

- Language and technology of the analyzed software.
- Types of vulnerabilities detected by the tool.
- Can it resolve dependencies automatically?
- Does it require source files or binary files?
- Can it be integrated with IDEs?
- Is it accurate?
- Is it easy to install / configure / use?

# Tutorial for Static Analysis

# Penetration Tests

# Penetration tests (PT)

- PT is a simulated and authorized attack to reveal possible exploits and their severity due to vulnerabilities.
- The tests can reveal the vulnerabilities but also the power of the system to defend itself.
- The tests can be performed in the form of a "black box” (typically) or in the form of a "white box".
- They can be performed internally by the organization or by external authorities or experts.

# PT: Phases

- **Recognition**
  - ✓ The activity acquires important information for the target system. The information will improve the attack.
- **Scanning**
  - ✓ Using automatic tools, the attacker gains more information. For example, we can find which ports are open or which input fields are vulnerable.
- **Get access**
  - ✓ Exploit identified vulnerabilities to attack the system or increase the level of authorization.
- **Maintain access**
  - ✓ Activities to be able to stay in the system and get as much information as possible.
- **Cover the tracks**
  - ✓ For the attack to succeed, the attacker must erase all traces that the software has been compromised.
- **Climbing**
  - ✓ After a resource (a machine or a component of the software) is compromised, the attacker can continue the attack to the other resources with the information obtained by the first step of the attack.

# Scenario

- The initial discovery revealed a misconfigured DNS server that gave access to a list of hosts.
- One of these hosts contained an interface of an administrative server protected by a password.
- The attacker was able to find the password with a brute force attack.
- The administrative interface has been vulnerable to remote code injections.
- The attacker got access to the operating system.
- An attack on a Java applet gave access to all machines used by administrators.
- The attack managed to compromise the entire system.

# Results - Reports

- Vulnerability: "Username and password weak or default (eg admin-admin). "
  - ✓ Risk: High
- Vulnerability: "Password reused. "
  - ✓ Risk: High
- Vulnerability: "Shared local administrator password (multiple hosts had the same passwords). "
  - ✓ Risk: High
- Vulnerability: "Patch management (multiple hosts have not been updated). "
  - ✓ Risk: High
- Vulnerability: "DNS zone transfer (DNS server incorrectly configured). "
  - ✓ Risk: Low
- Vulnerability: "Apache file by default. "
  - ✓ Risk: Low

# Penetration test vs Static analysis

| **Penetration test** | **Static analysis** |
|---|---|
| ▪ After the deployment. | ▪ During the development. |
| ▪ Must combine with manual efforts. | ▪ Mostly automatic, but with the risk of false positives. |
| ▪ Increase the certainty of the risks. | ▪ Capable of early identification and minimization of correction costs. |

# Tutorial for Penetration Tests

# TP3 - Security

- **Static analysis**
- **Penetration testing**
- **Due on December 1st**