ESOF 422:

Advanced Software Engineering: Cyber Practices

Secure by Design (Part 3)

Software Development Lifecycle, Testing

Reese Pearsall

Spring 2025

Announcements

Third exam will take place during finals week

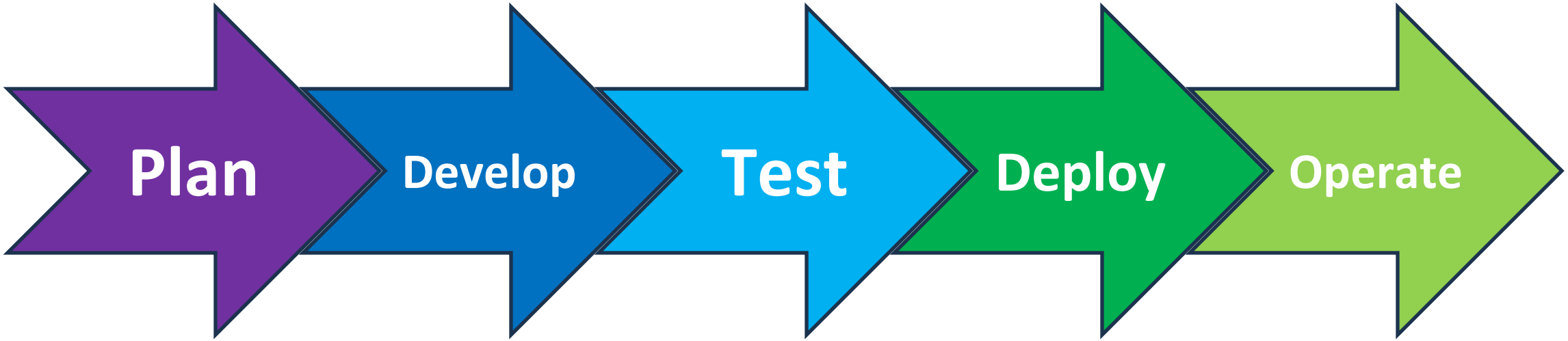
This exam is **optional**
If you don't take it, the average of your first two exam scores will be used instead

- The exam will largely cover the second half of the semester, but there may be material from earlier this semester

Next homework posted. Due Friday April 4th
→ Coding-based project

HW 4

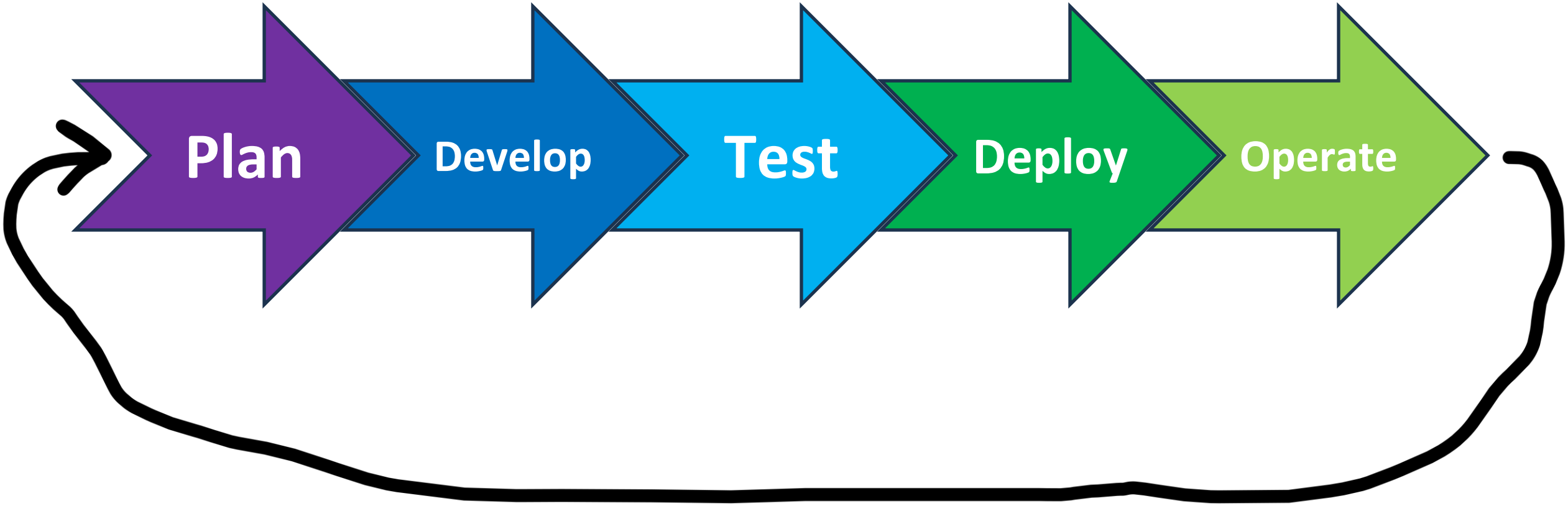
Software Development Lifecycle

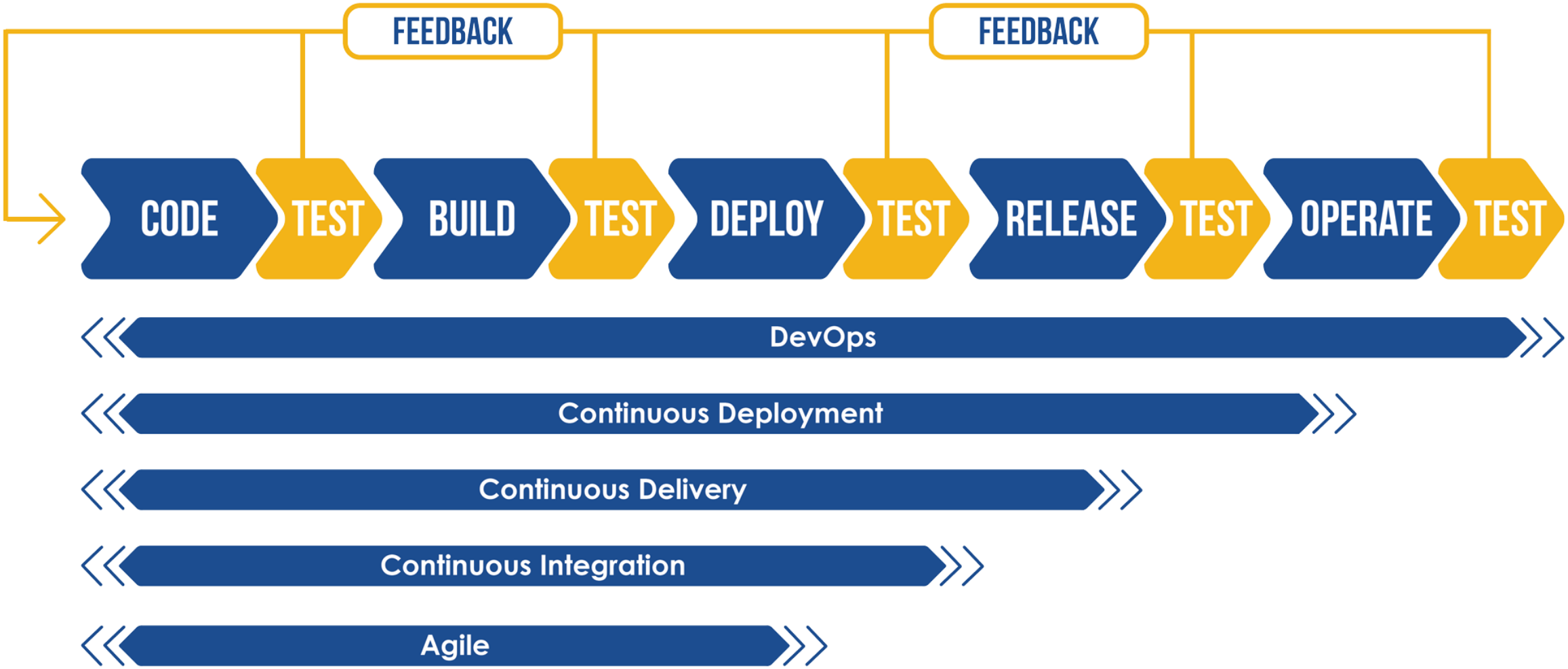


~~Software Development Lifecycle~~

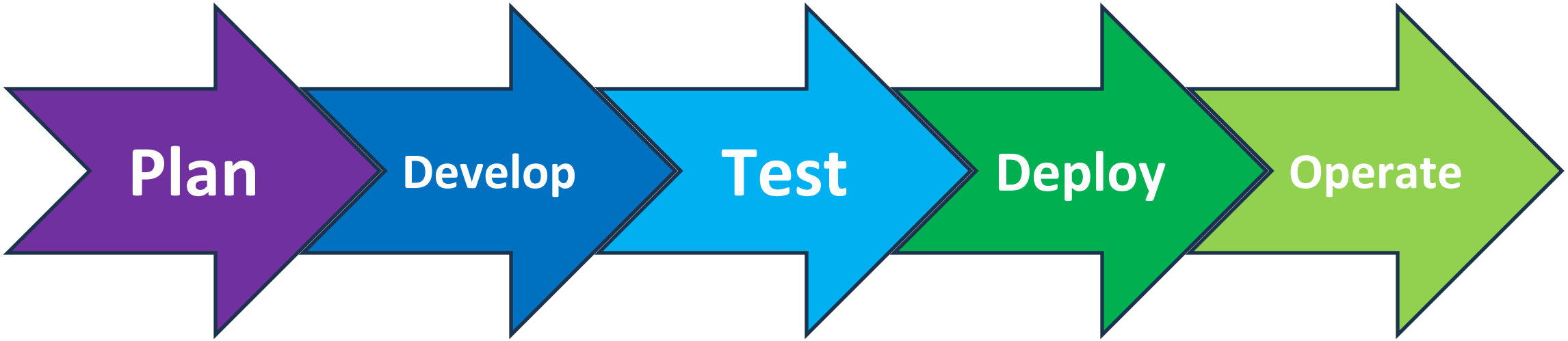
"Continuous Delivery Pipeline"

CI/CD

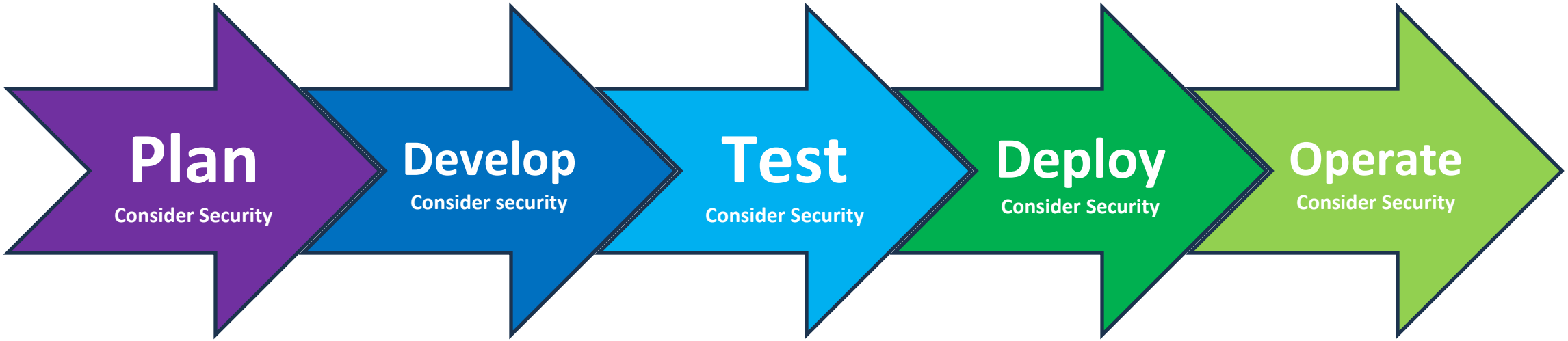



FEEDBACK
FEEDBACK
CODE
TEST
BUILD
TEST
DEPLOY
TEST
RELEASE
TEST
OPERATE
TEST
DevOps
Continuous Deployment
Continuous Delivery
Continuous Integration
Agile

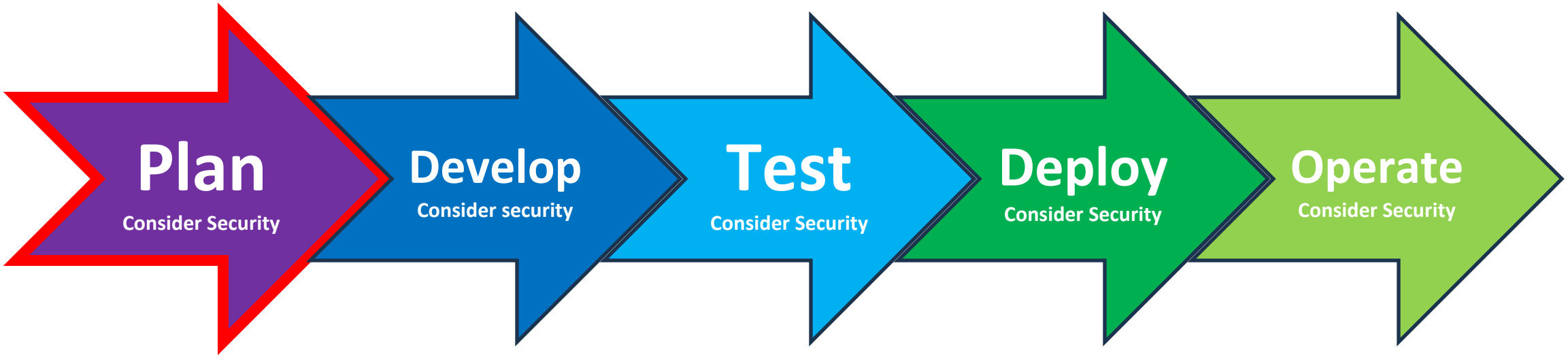
Software Development Lifecycle



Secure Development Lifecycle (SDL)



Secure Development Lifecycle (SDL)



- Clearly Define the **requirements**

  Many software issues and vulnerabilities can be linked to poor requirement gathering

Compliance- requirement to adhere to specific standards and regulations to ensure CIA

Gaps in security

How should encryption and authentication be handled?

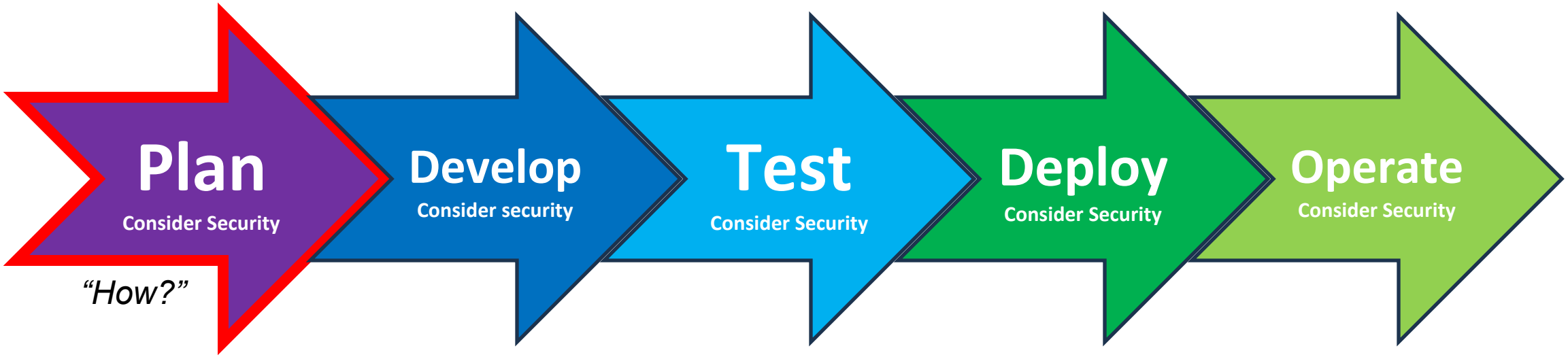
Performance Issues

Estimated user load?
Physical servers and database requirements?

Compliance Regulations

Healthcare data must be HIPPA compliant
Credit card data must be PCI compliant

Secure Development Lifecycle (SDL)



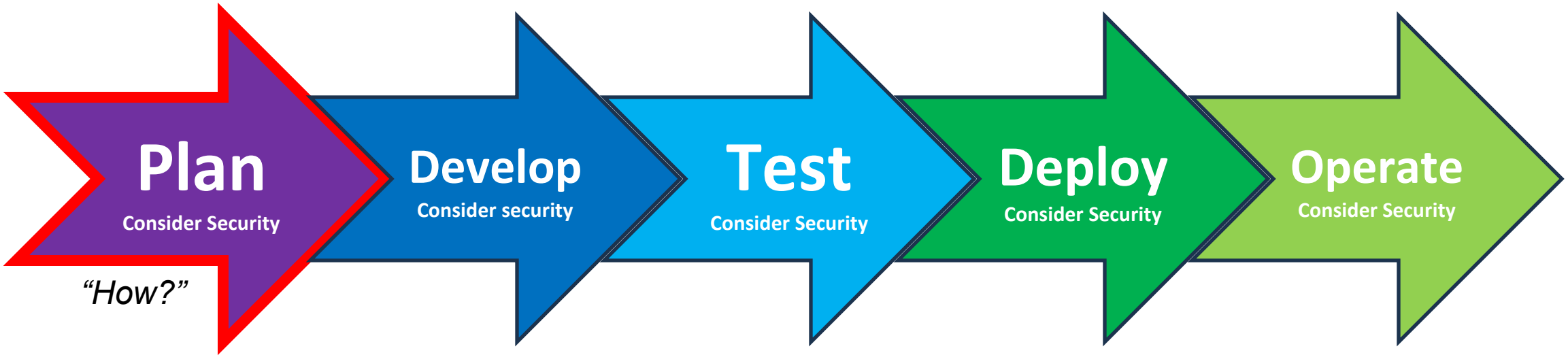
Generate UML Class Diagrams, Sequence Diagrams

- Are there any design patterns?
- What external components are there?

Secure Architecture- Apply security principles when designing the system

Principle of least privilege: Giver users the minimum level of access necessary to perform their tasks.

Secure Development Lifecycle (SDL)



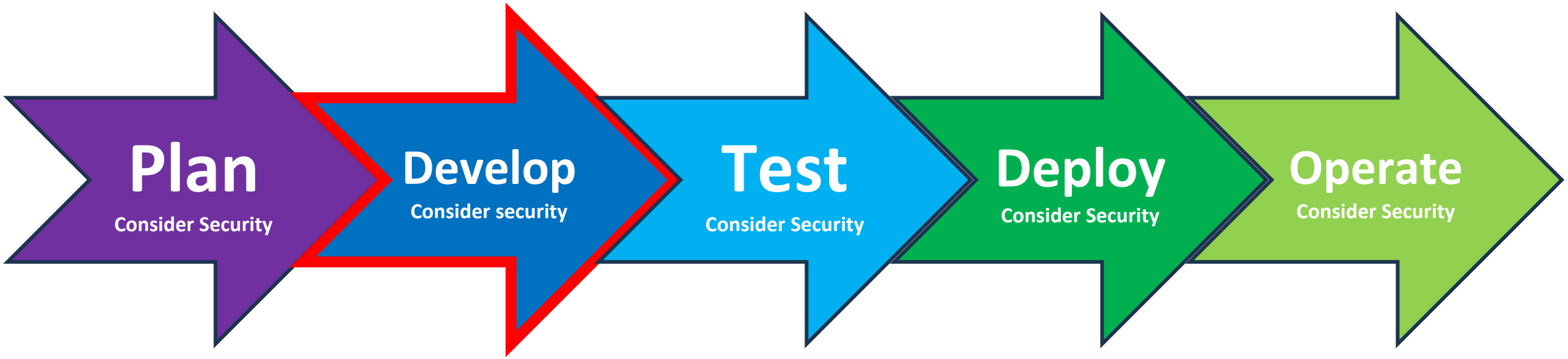
Generate UML Class Diagrams, Sequence Diagrams

- Are there any design patterns?
- What external components are there?

Secure Architecture- Apply security principles when designing the system

Principle of secure defaults: The default configurations should favor security rather than convenience

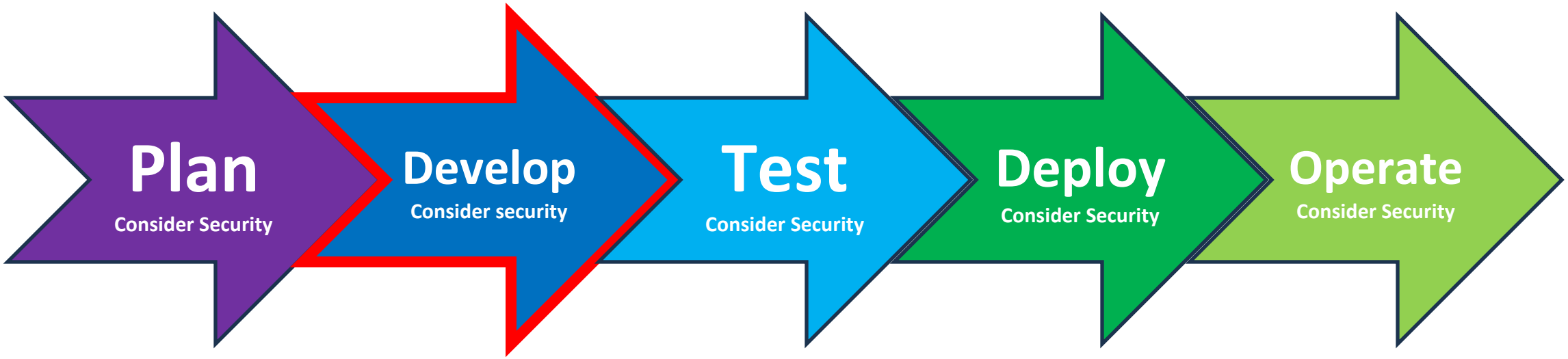
Secure Development Lifecycle (SDL)



- Code Responsibly
- Try to step in the shoes of an attacker

Bug vs vulnerability ?

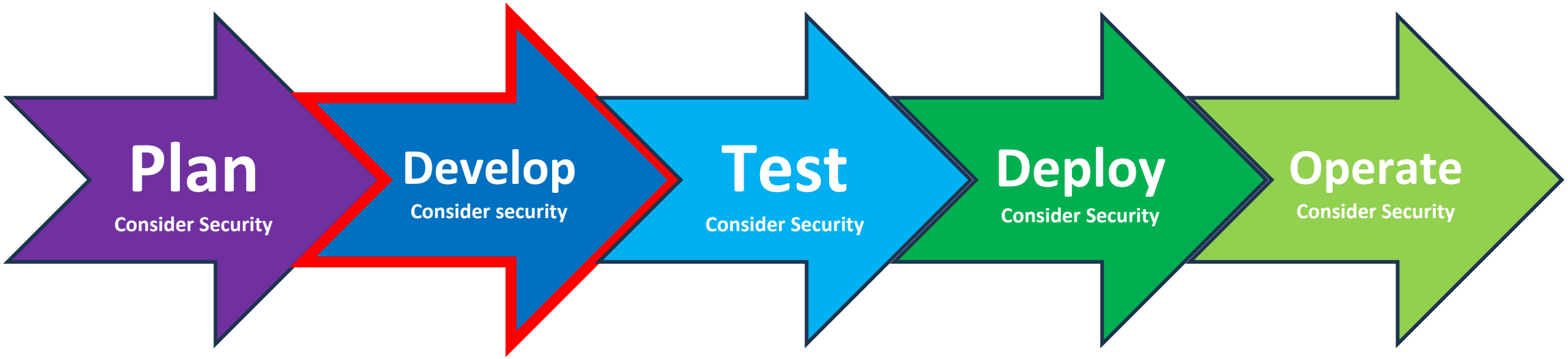
Secure Development Lifecycle (SDL)



- Code Responsibly
- Try to step in the shoes of an attacker

Bug- an *unintended* flaw in code that cause incorrect or unexpected behavior
Vulnerability- a bug that creates some weakness *that can be exploited*

Secure Development Lifecycle (SDL)

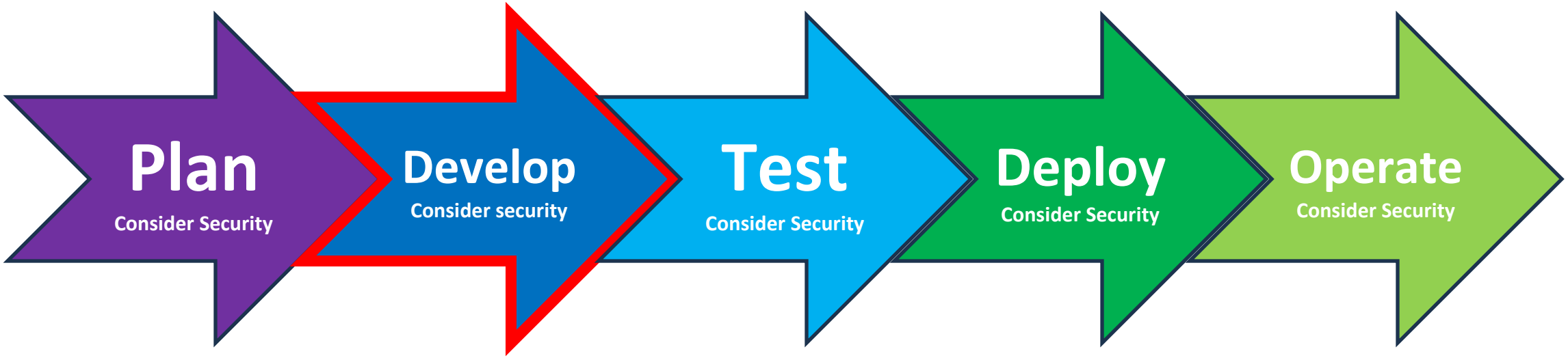


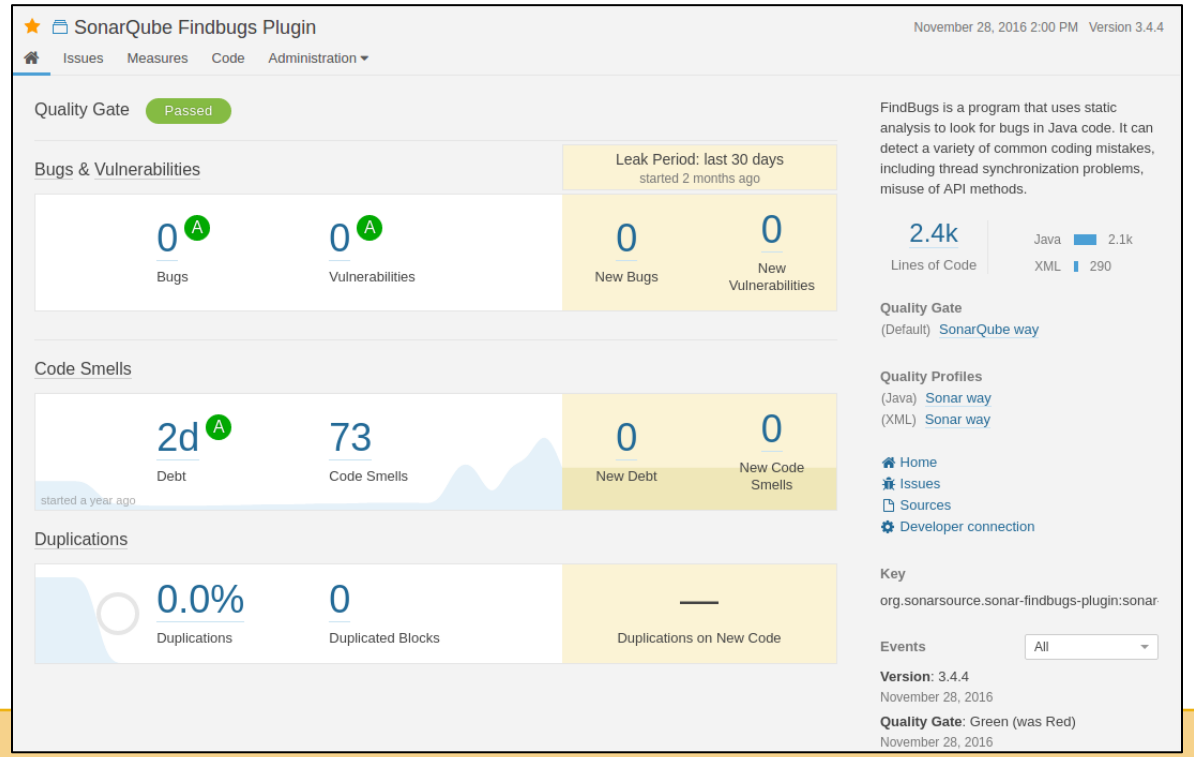
- Code Responsibly
- Try to step in the shoes of an attacker

- Secure coding practices (NIST, OWASP)

- Static code analysis tools

Secure Development Lifecycle (SDL)



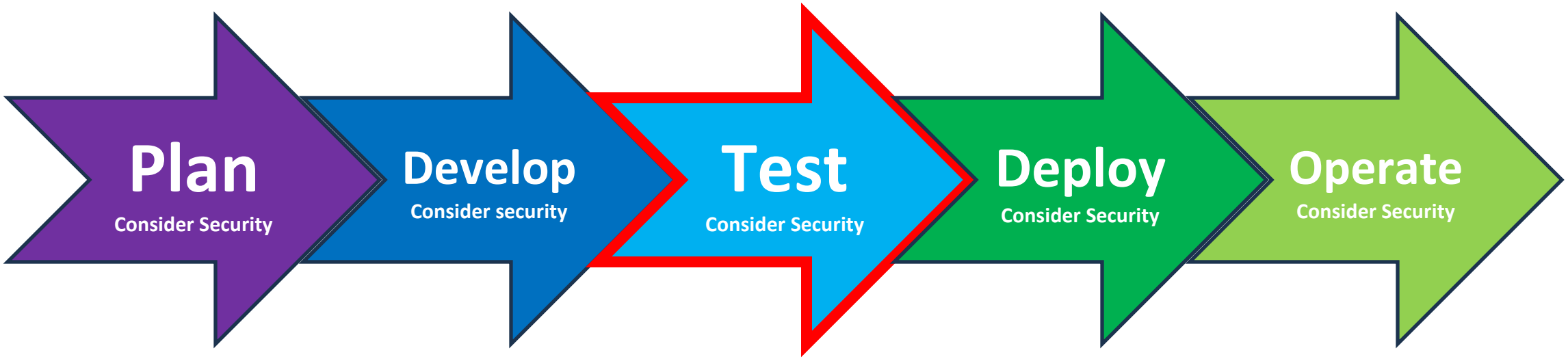


Sonar Qube is an excellent static analysis tools that can automatically scan for bugs and vulnerabilities in source code,

Code Smells- poor coding practices that don't necessarily cause errors

Technical Debt- the cost of taking shortcuts during the software development process

Secure Development Lifecycle (SDL)

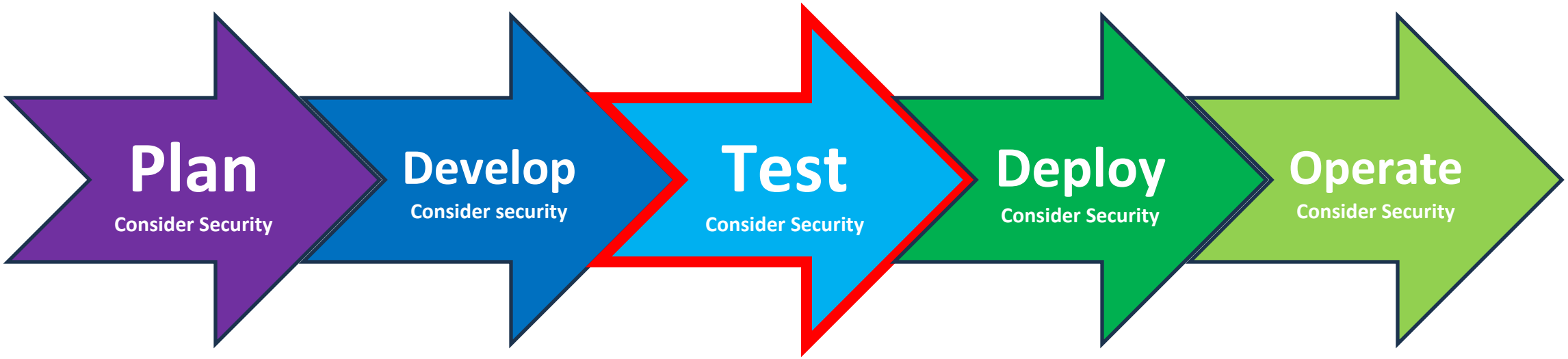


Software tests and security tests are not the same thing

- Software tests focus on software failure and intended design
- Software security tests adds a focused adversary

Security flaws are more difficult and more expensive to six later in the development lifecycle

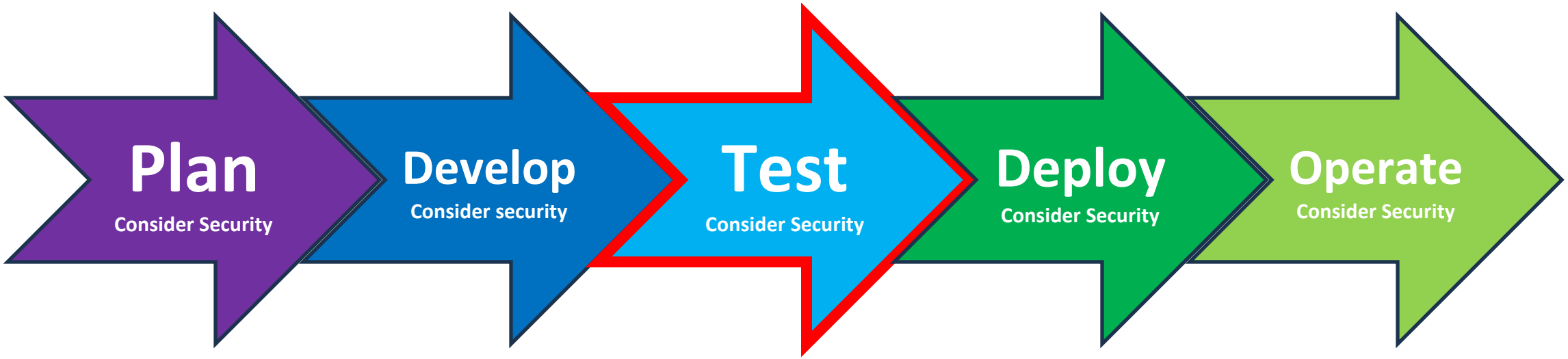
Secure Development Lifecycle (SDL)



- **White Box**
  - Tester knows all information about system.
  - Including source code, design, requirements.
  - Most efficient technique.

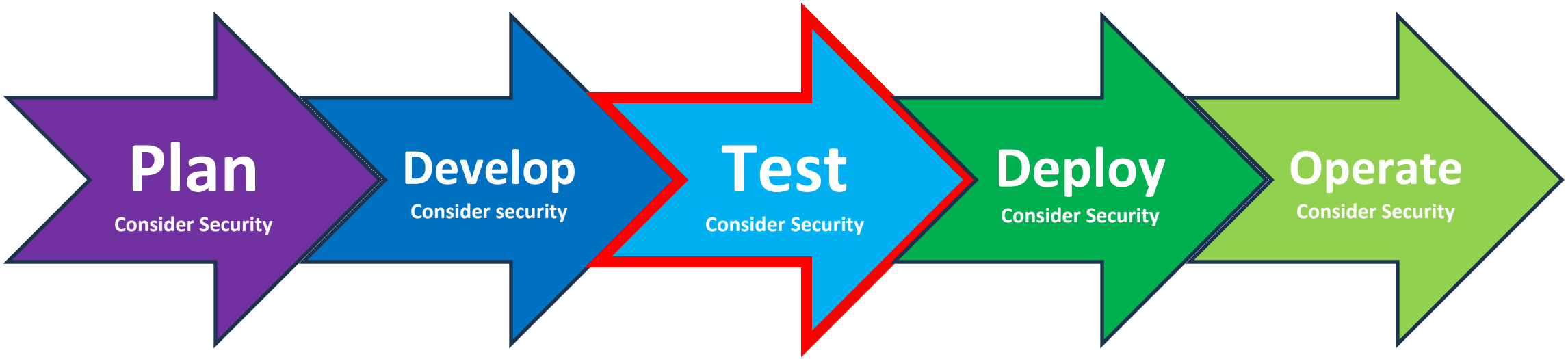
- **Black Box**
  - Examines system as an outsider world
  - Tester builds understanding of attack surface and system internals during test process
  - Can use to evaluate effort required to attack system

Secure Development Lifecycle (SDL)



Normal Input Testing	**Boundary Input Testing**	**Invalid Input Testing**	**Extreme Input Testing**
Verifies that the design accepts input that clearly passes the domain rules, ensuring that the code handles *vanilla* input correctly	Verifies that only structurally correct input is accepted. Examples of boundary checks include length, size, and quantity.	Verifies that the design doesn't break when invalid input is handled. Empty data structures, `null`, and strange characters are often considered invalid input.	Verifies that design doesn't break when extreme inputs are handled. For example, such input may include a string of 40 million characters.

Secure Development Lifecycle (SDL)



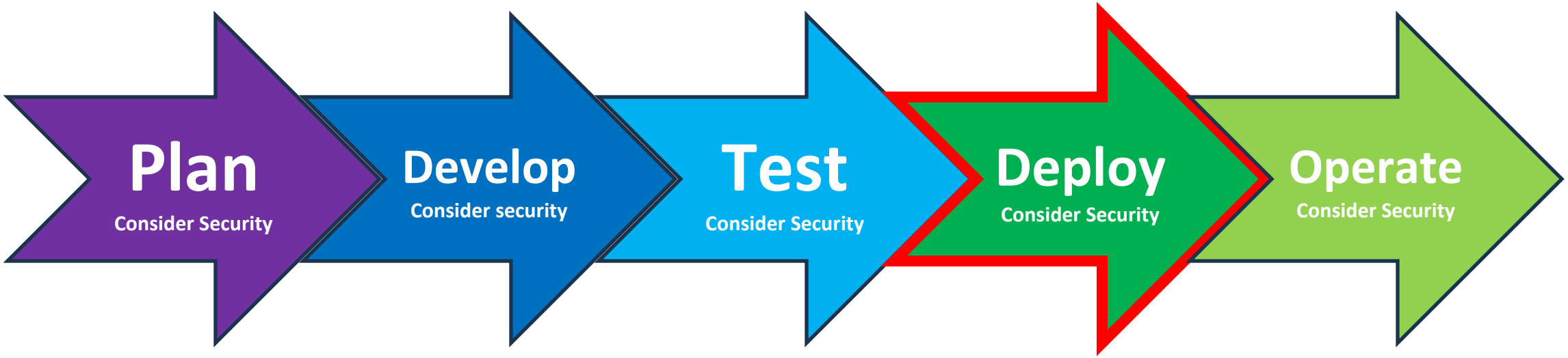
Availability of system is important. However, unit tests cannot really be written to taest for availability

Bees with machine guns is a command line utility

> **Bees with Machine Guns!**
>
> A utility for arming (creating) many bees (micro EC2 instances) to attack (load test) targets (web applications).
>
> Also, retribution for this shameful act against a proud hive.

For load testing web applications for scenarios such as DDOS attacks

Secure Development Lifecycle (SDL)



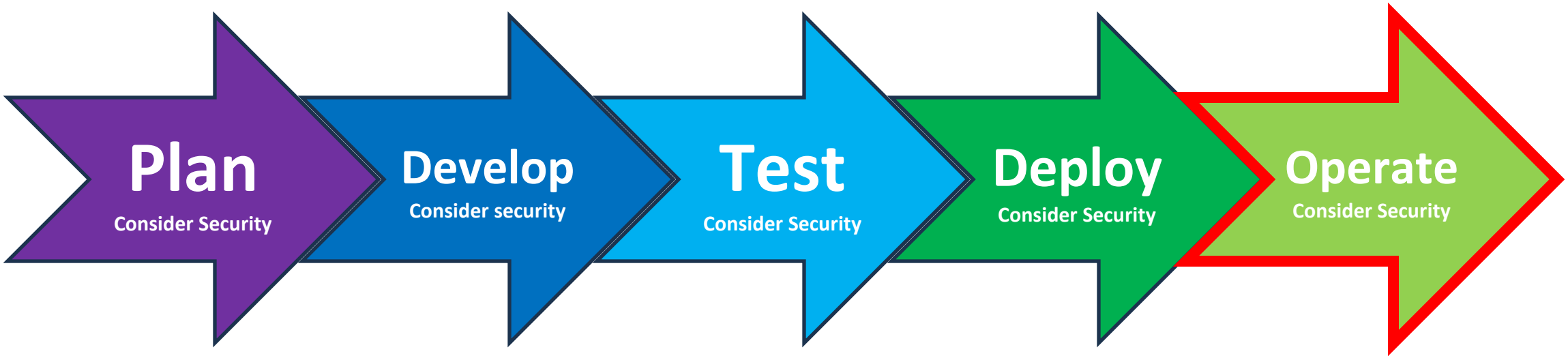
Ensure cloud instance is configured correctly

Enable all necessary services, disable all unnecessary services

Accumulate all third-party security components in the supply chain

- Libraries
- API
- Package Managers

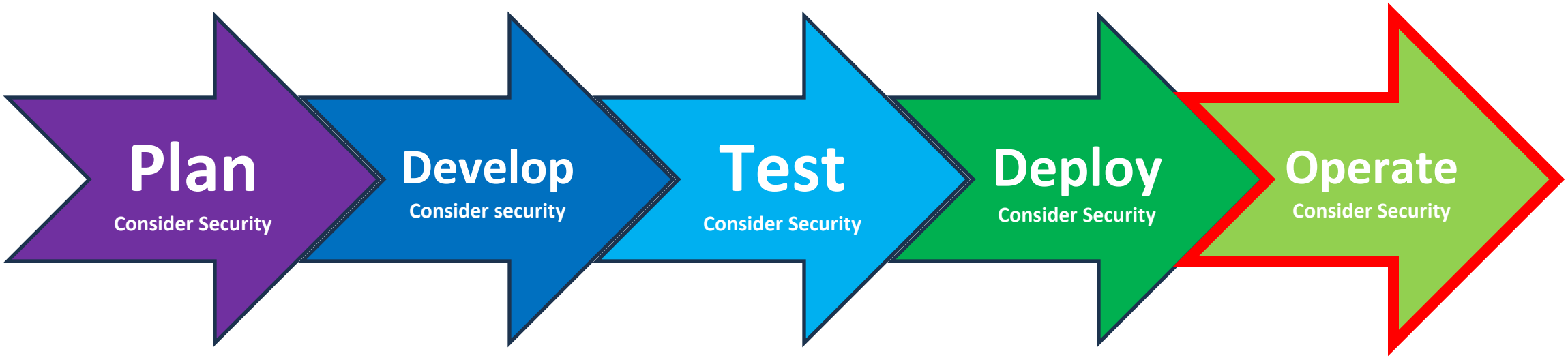
Secure Development Lifecycle (SDL)



Log and **monitor** traffic to detect anomalies and potential threats

A **security operations center** (**SOC**) is an team that monitors, detects, analyzes, and respond to cyberthreats to protect an organization's system

Secure Development Lifecycle (SDL)

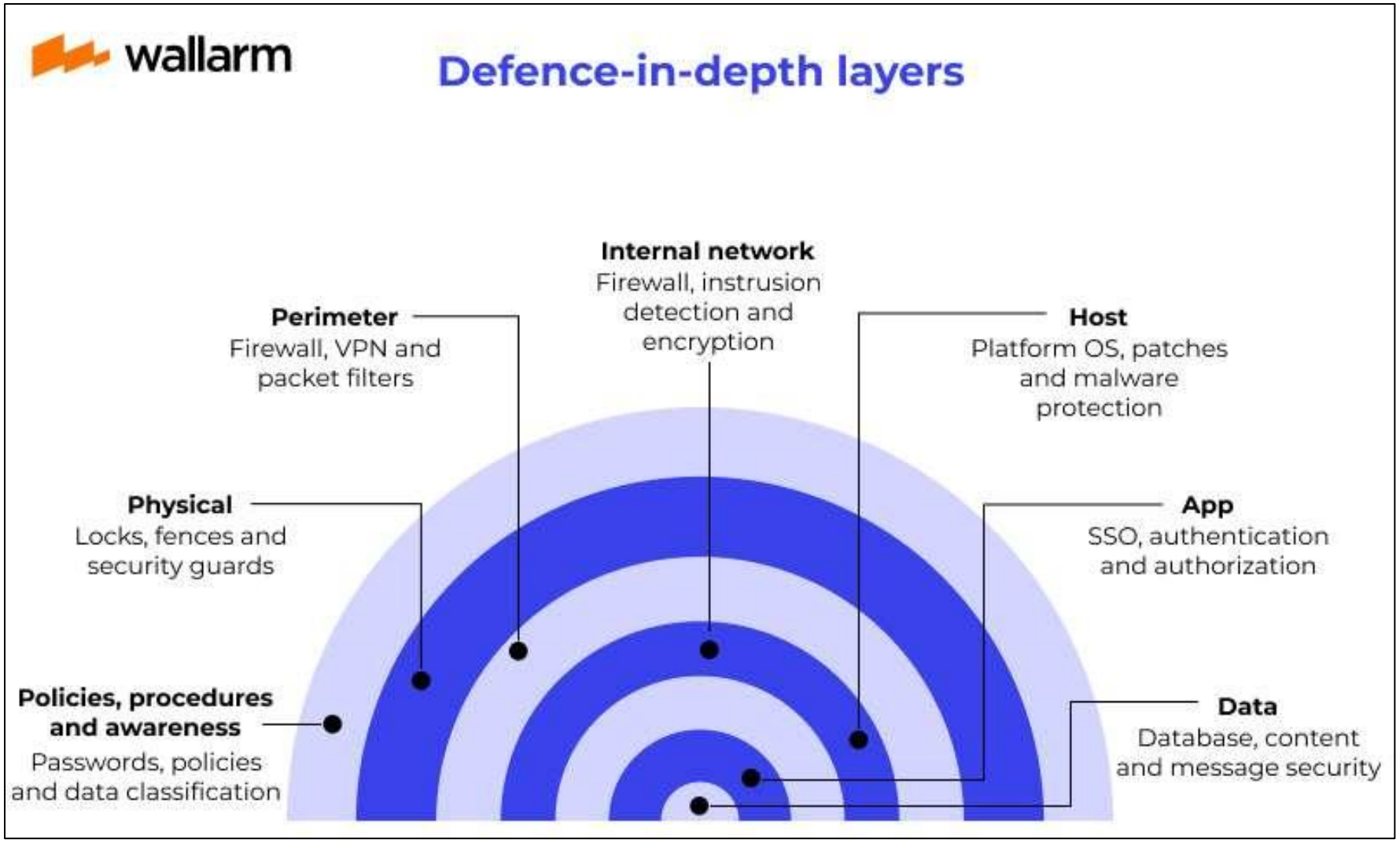


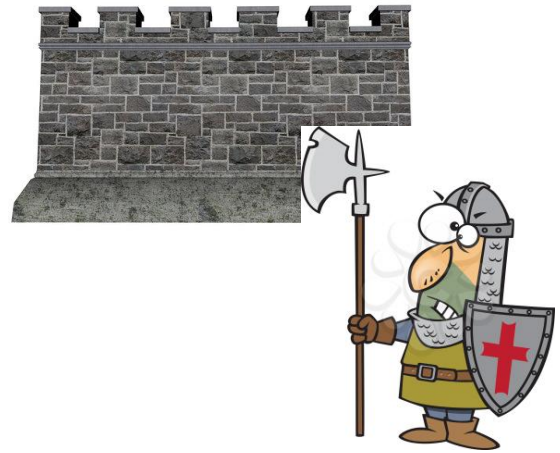
Log and **monitor** traffic to detect anomalies and potential threats

Incidence Response Plan

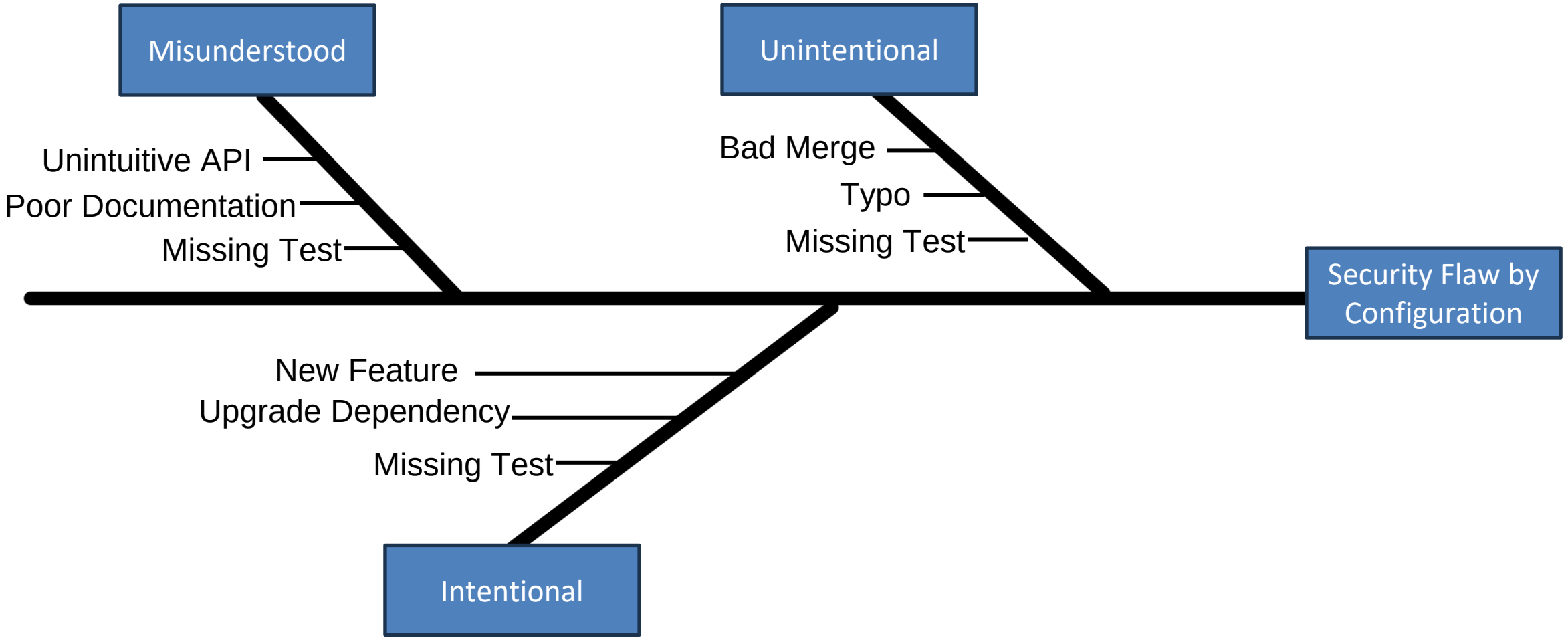
Application maintenance and patching

Defense-in-Depth is a principle that states multiple layers of security should be implemented in case one layer fails





Security flaws can often come from *configuration-related* issues



Software Development Lifecycle