ShiftLeft

APPSEC FUNDAMENTALS FOR MODERN DEVOPS

Suchakra Sharma
Staff Scientist
ShiftLeft

Vickie Li
Developer Evangelist
ShiftLeft

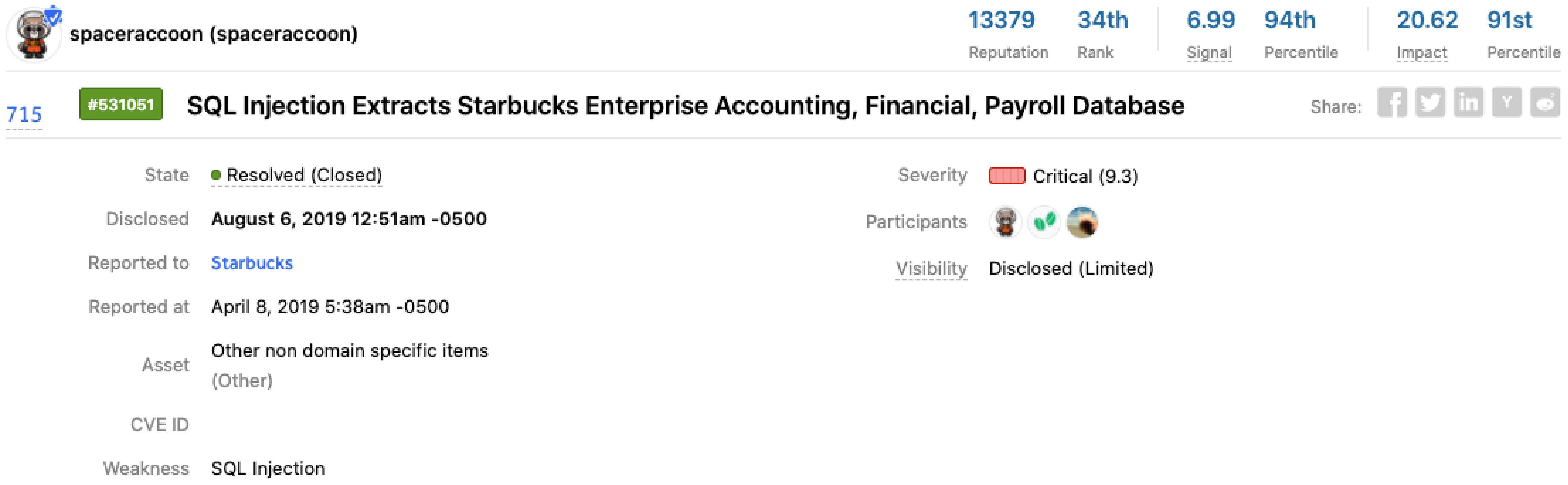
LISA 2021

WHAT YOU'LL LEARN IN THIS SESSION

OWASP TOP TEN : WHAT COULD GO WRONG?

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entities (XXE)
- Broken Access Control
- Security Misconfiguration
- Cross-Site Scripting (XSS)
- Insecure Deserialization
- Using Components with Known Vulnerabilities
- Insufficient Logging & Monitoring

SQL INJECTION



spaceraccoon (spaceraccoon)

13379	34th	6.99	94th	20.62	91st
Reputation	Rank	Signal	Percentile	Impact	Percentile

715 #531051 SQL Injection Extracts Starbucks Enterprise Accounting, Financial, Payroll Database

Share:

State	● Resolved (Closed)
Disclosed	August 6, 2019 12:51am -0500
Reported to	Starbucks
Reported at	April 8, 2019 5:38am -0500
Asset	Other non domain specific items (Other)
CVE ID	
Weakness	SQL Injection
Severity	🟥 Critical (9.3)
Participants	
Visibility	Disclosed (Limited)

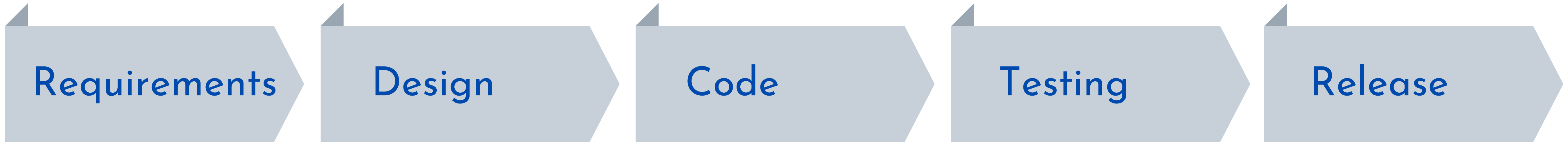
- `Starbucks SQL injection: https://hackerone.com/reports/531051`

LISA 2021

ShiftLeft

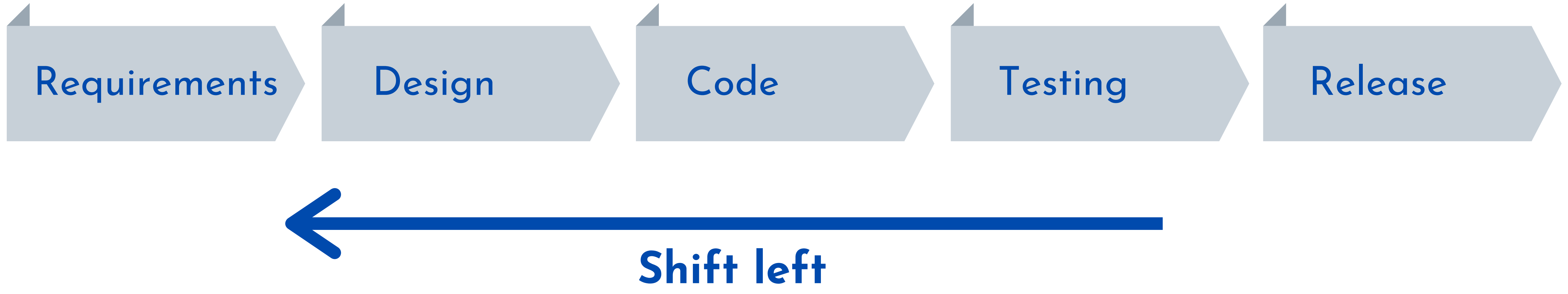
THE SDLC:
SOFTWARE DEVELOPMENT LIFE CYCLE

- Requirements
- Design
- Code
- Testing
- Release



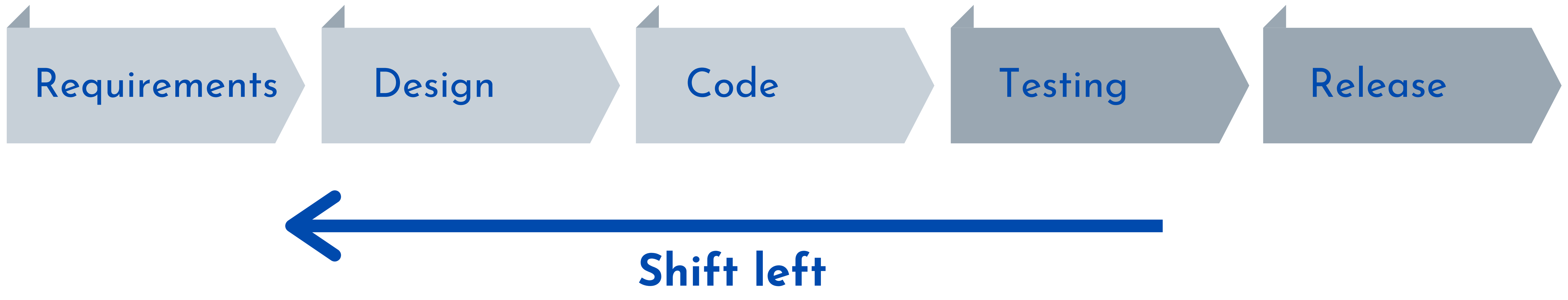
THE SDLC:
SOFTWARE DEVELOPMENT LIFE CYCLE

- Requirements
- Design
- Code
- Testing
- Release



THE SDLC:
SOFTWARE DEVELOPMENT LIFE CYCLE

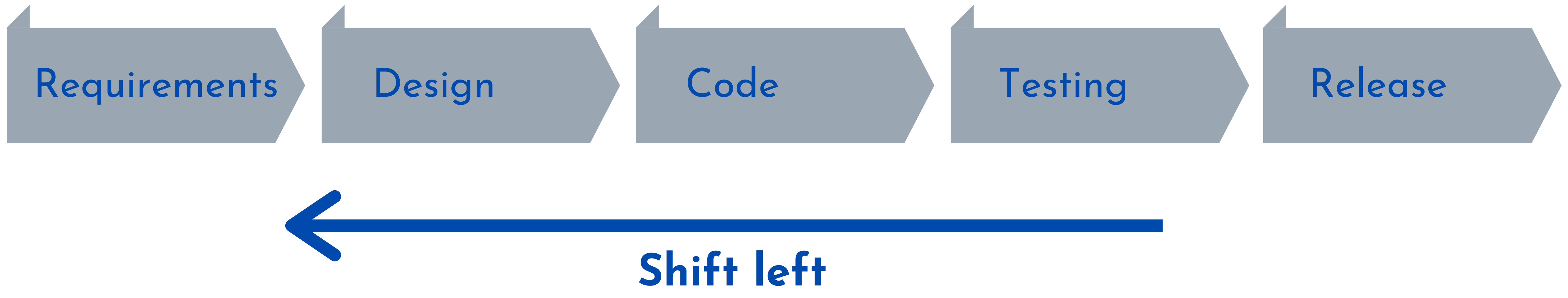
- Requirements
- Design
- Code
- Testing
- Release



ShiftLeft

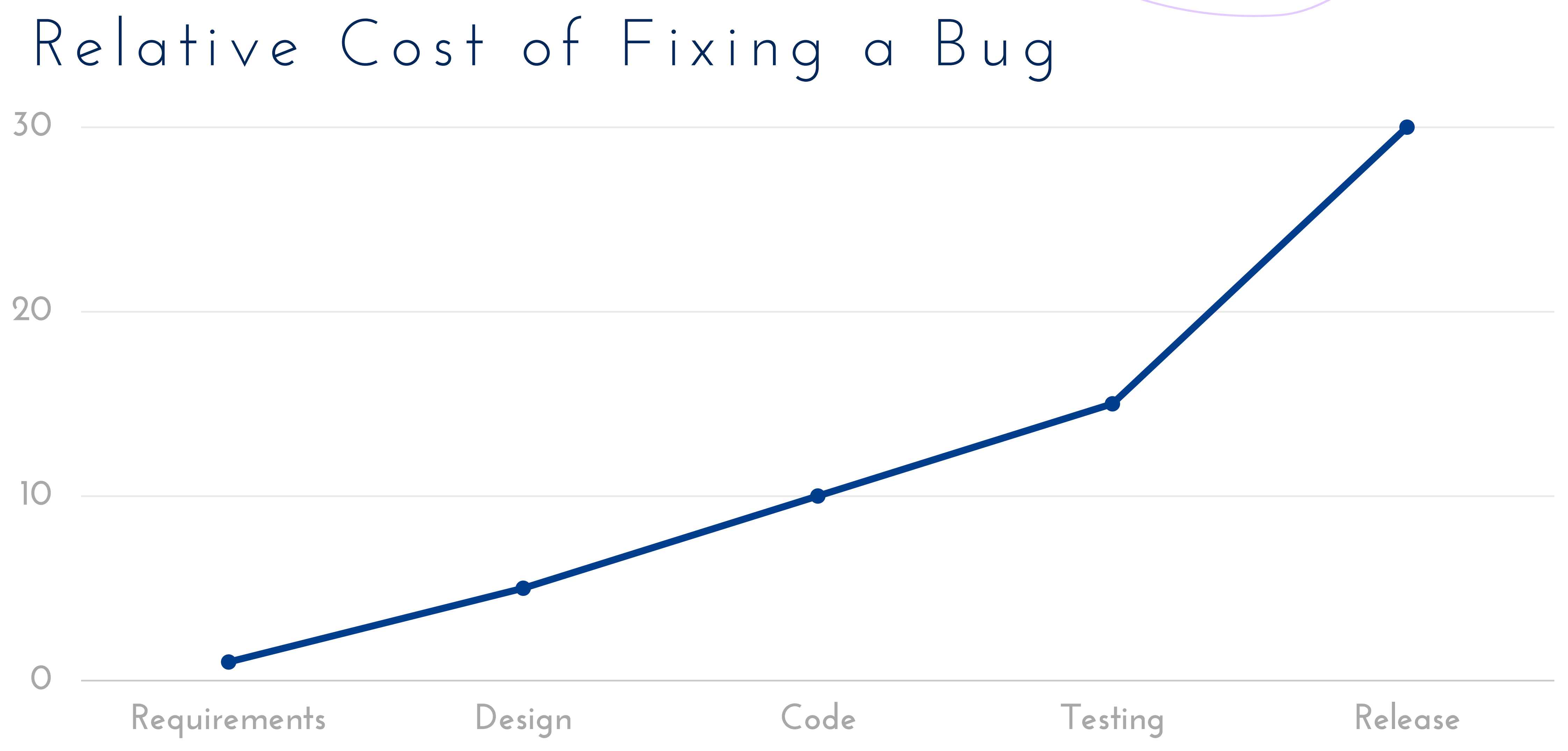
THE SDLC:
SOFTWARE DEVELOPMENT LIFE CYCLE

- Requirements
- Design
- Code
- Testing
- Release



ShiftLeft

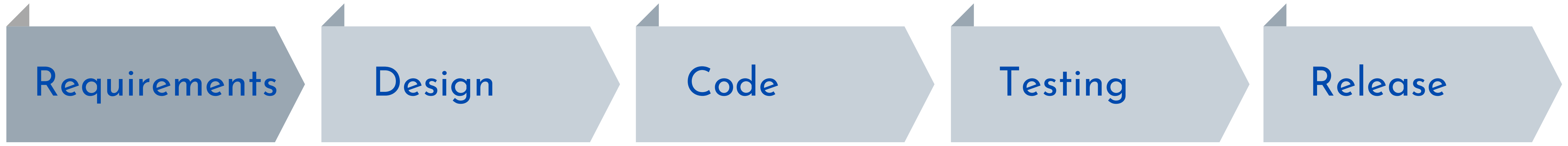
SHIFTING LEFT



The relative costs of fixing bugs in terms of person-hours. Data courtesy of NIST: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf.

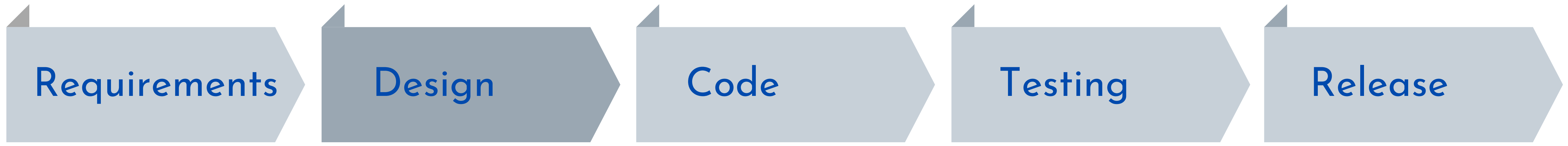
SHIFT LEFT: REQUIREMENTS

- Ask security questions from the very start.
- Include security folks in requirement planning.



SHIFT LEFT: DESIGN

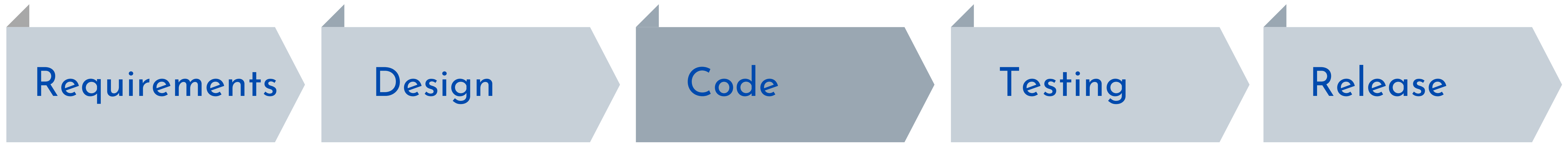
- Plan application design around security requirements.
- Consider building in security mechanisms like input validation, output encoding, and prepared statements from the start.



ShiftLeft

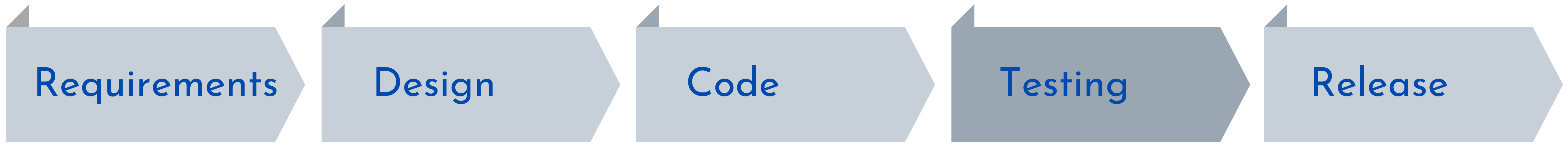
SHIFT LEFT: CODE

- Choose a secure programming language and framework.
- Handle untrusted data safely via validation, sanitization, and output encoding.
- Implementing proper error handling and logging.



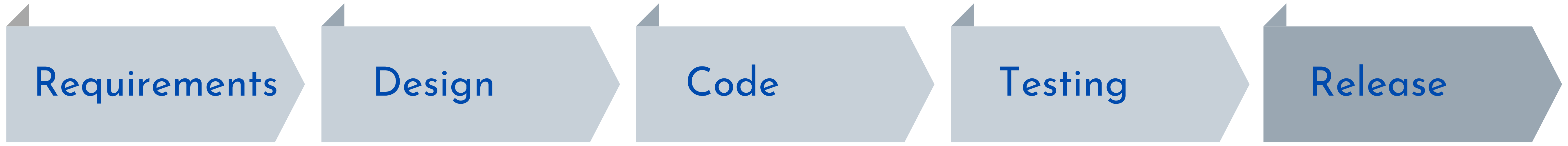
SHIFT LEFT: TESTING

- Manual code review
- SAST (Static Analysis Security Testing)
- SCA (Software Composition Analysis)
- DAST (Dynamic Analysis Security Testing)
- Pentests + bug bounty programs



SHIFT LEFT: RELEASE

- Pay attention to the security of your CICD pipeline.
- Build security tests into the pipeline, such as dependency monitoring and SAST scans.



SQL INJECTION

- A SQL Injection attack is when an attacker can inject arbitrary SQL code into SQL statements that an application uses to access its database.

ShiftLeft

SQL INJECTION

- A SQL Injection attack is when an attacker can inject arbitrary SQL code into SQL statements that an application uses to access its database.

```
String queryString =
   "SELECT * FROM USER WHERE
   USERNAME = '" + Username + "'
   AND PASSWORD = '" + Password + "'";

sql.executeQuery(QueryString)
```

ShiftLeft

SQL INJECTION

- A SQL Injection attack is when an attacker can inject arbitrary SQL code into SQL statements that an application uses to access its database.

```
String queryString =
   "SELECT * FROM USER WHERE
   USERNAME = '" + Username + "'
   AND PASSWORD = '" + Password + "'";

sql.executeQuery(QueryString)
```

HTTP request:

```
POST /login

Username=Vickie
Password=password123
```

SQL INJECTION

- A SQL Injection attack is when an attacker can inject arbitrary SQL code into SQL statements that an application uses to access its database.

```
String queryString =
   "SELECT * FROM USER WHERE
   USERNAME = '" + Username + "'
   AND PASSWORD = '" + Password + "'";

sql.executeQuery(QueryString)
```

```
HTTP request:

POST /login

Username=Vickie
Password=password123
```

ShiftLeft

SQL INJECTION

- A SQL Injection attack is when an attacker can inject arbitrary SQL code into SQL statements that an application uses to access its database.

```
SELECT Id FROM Users
WHERE Username='vickie' AND Password='password123';
```

SQL INJECTION

- A SQL Injection attack is when an attacker can inject arbitrary SQL code into SQL statements that an application uses to access its database.

```
String queryString =
   "SELECT * FROM USER WHERE
   USERNAME = '" + Username + "'
   AND PASSWORD = '" + Password + "'";

sql.executeQuery(QueryString)
```

```
HTTP request:

POST /login

Username=admin';--
Password=password123
```

SQL INJECTION

- A SQL Injection attack is when an attacker can inject arbitrary SQL code into SQL statements that an application uses to access its database.

```
SELECT Id FROM Users
   WHERE Username='admin';-- ' AND Password='password123';
```

SQL INJECTION

- A SQL Injection attack is when an attacker can inject arbitrary SQL code into SQL statements that an application uses to access its database.

```
SELECT Id FROM Users
   WHERE Username='admin';-- ' AND Password='password123';
```

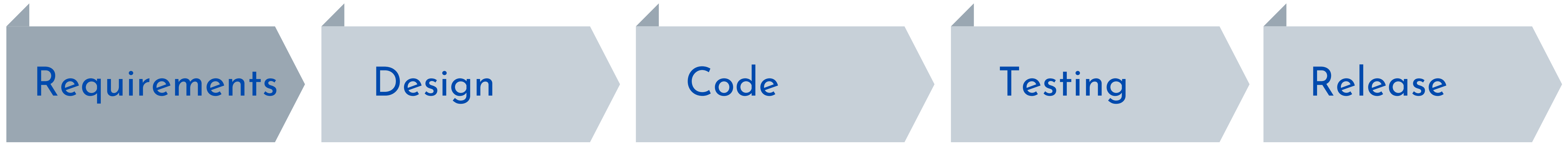
PREVENTING SQL INJECTION

- A SQL Injection attack is when an attacker can inject arbitrary SQL code into SQL statements that an application uses to access its database.

```
SELECT Id FROM Users
   WHERE Username='admin';-- ' AND Password='password123';
```

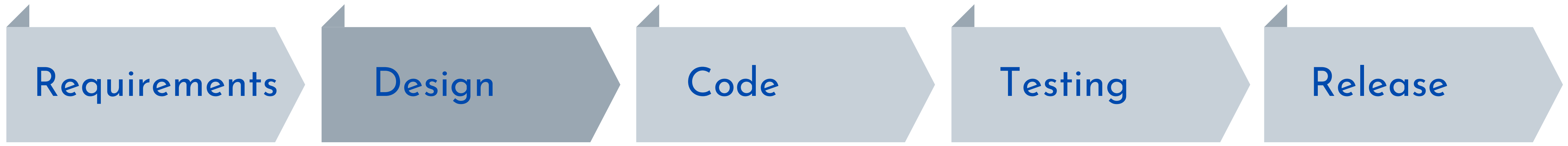
PREVENTING SQL INJECTION

- How will sensitive data be stored and transported?
- When does this app need to take in user input?
- Where does this app make database calls?
- Are user input needed in database calls?



PREVENTING SQL INJECTION

- What mechanisms should we use to handle user input safely?
- Where are input validation, sanitization, and escaping needed?
- How do we secure database calls?
- How do we store sensitive data safely to minimize damage in case of a breach?
- What is the least privilege needed for the application to run?
- How do we backup data and code?
- How should we log potential attacks and errors?



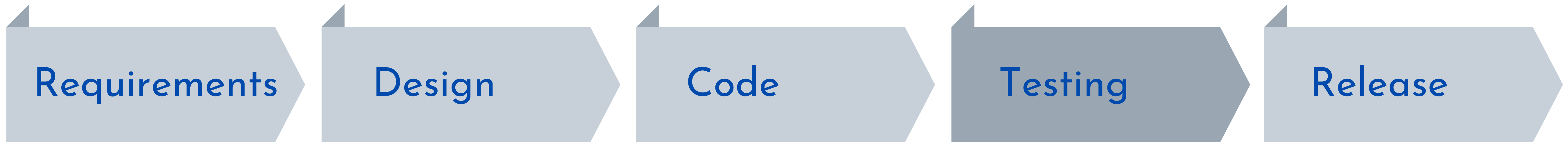
PREVENTING SQL INJECTION

- Implement input validation.
- Escape or reject dangerous characters.
- Implement prepared statements.
- Implement the principle of least privilege.
- Store data securely.

Requirements
Design
Code
Testing
Release

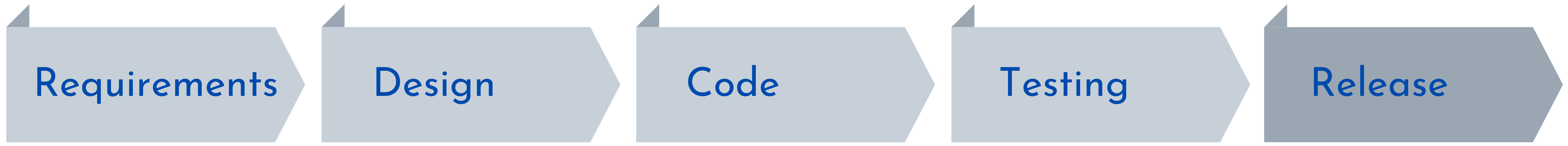
PREVENTING SQL INJECTION

- Manual code review of dangerous functions.
- SAST scanning for signatures of SQL injection.
- SCA to ensure third-party components are secure.



PREVENTING SQL INJECTION

- Build security tests into the pipeline, such as SCA and SAST scans.
- Bug bounty programs.
- Regularly back up important data and code.
- Monitor the application for potential attacks.



THANK YOU!

Feel free to connect:
@vickieli7
@tuxology

ShiftLeft