Static Code Analysis @ Swisscom

Group Security
Secure Software Development

Frank Bennewitz
Cloud Application Management

swisscom

Who am I?

- Frank Bennewitz
- Developer
- CAD, Web Applications, BPE, OAuth2
- IT Security Analyst @ Swisscom
- Automated source code scanning for security vulnerabilities

Introduction
What do we do?

- Application security key business success factor
- Professional hackers are after you (govs, criminals, terrorists, hacktivists, wargamers, ..)
- Broad arsenal of technologies
- Assumtion you might be vulnerable
- Our approach
- Detect in the earliest stage (where product gets implemented) as possible: Development
- Secure Software Development Lifecycle (SSDLC)
- This does not save the world but makes it a better place.
- So just one building block in a program

SSDLC – Problems
Governance

- Uncertainty during development.
- Responsibility at the devs.
- Results in different implementations for the same problems.

SSDLC – Problems
Security Approval

- Security approval / audit only held at a fraction of projects
- Slow due to the lack of standards and automation
- Security approvals conducted in a late project state
- Flaws from earlier stages in the process (e.g. Design) are hard to mitigate against.

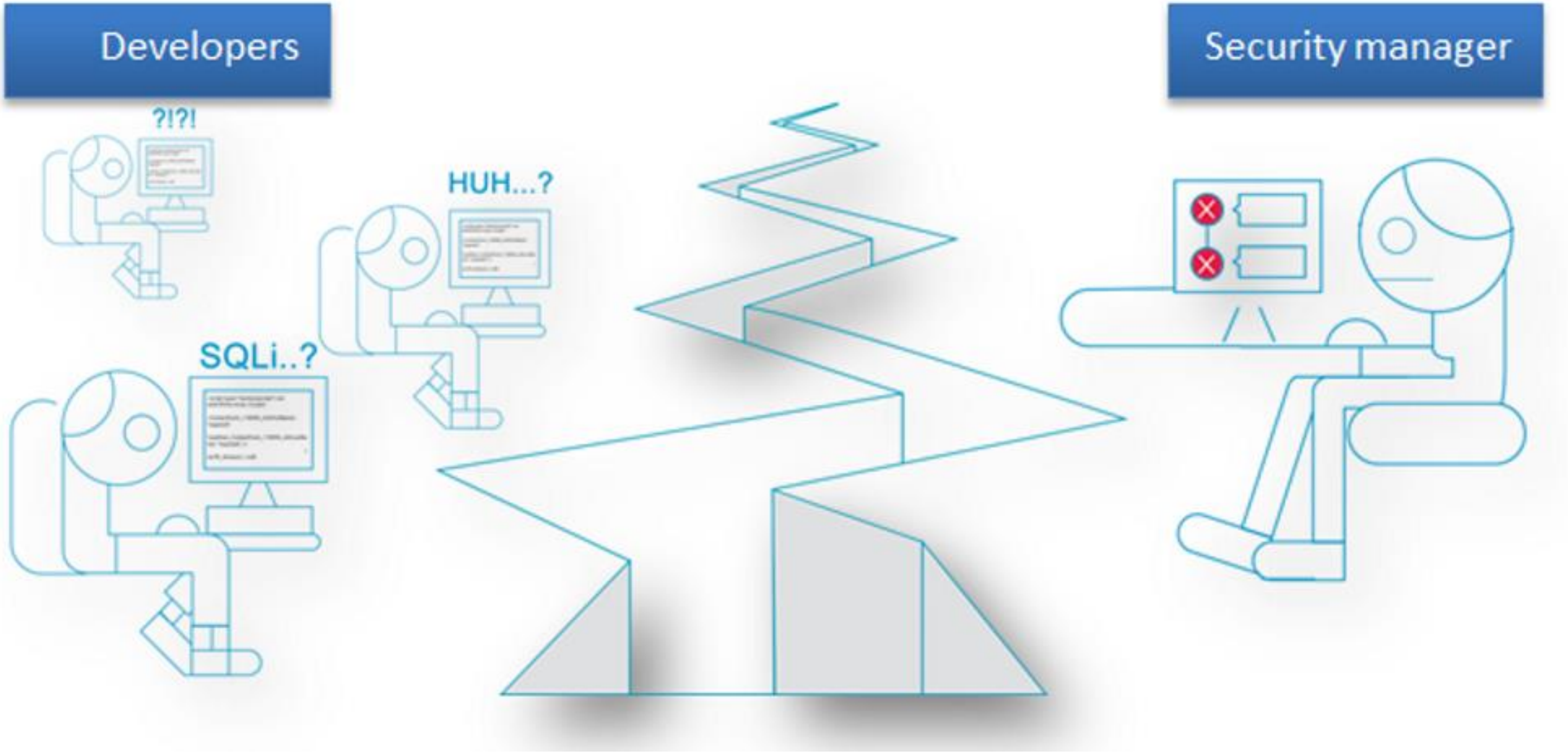
SSDLC – Problems
Dev Infrastructure

- Operating of non standardized dev infrastructure

- Working on operations equals wasted time

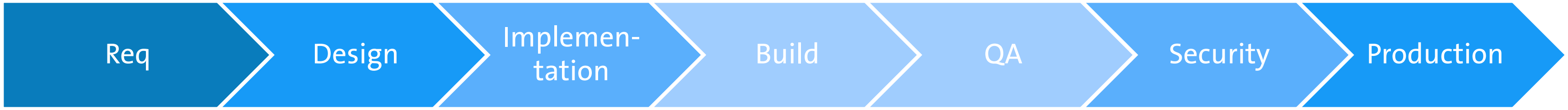
- «Do best in what you can» (dev)

SSDLC – Problems
Communication



Everybody loves Security

Development vs. Security



Development

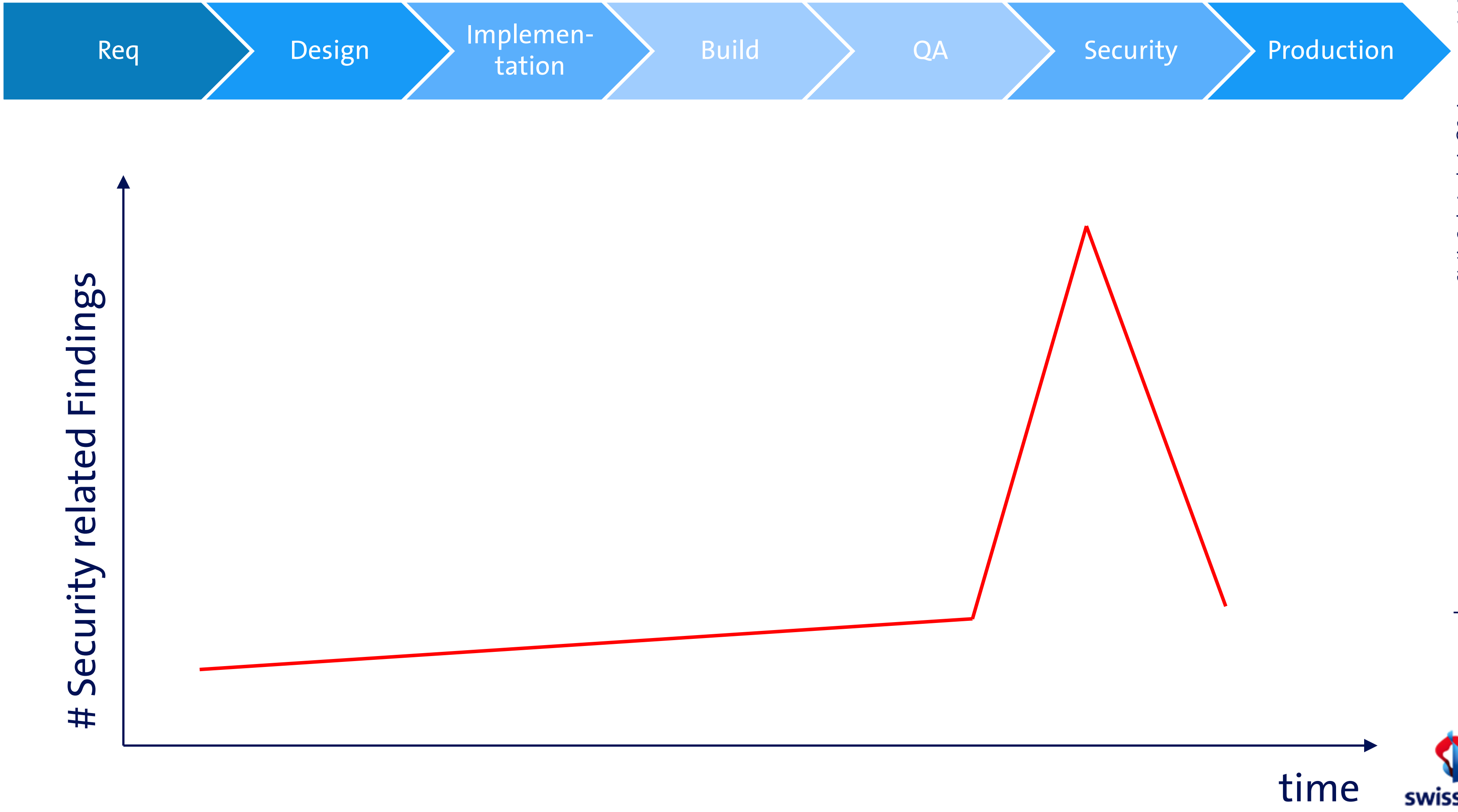
- Lack of education / standards
- Complexity
- «Not my job»

Security

- Testing just before release
- Too many applications
- Too many technologies

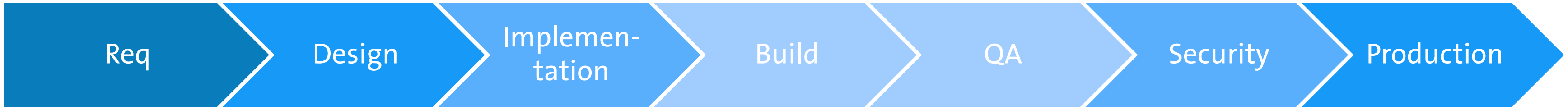
Findings per phase

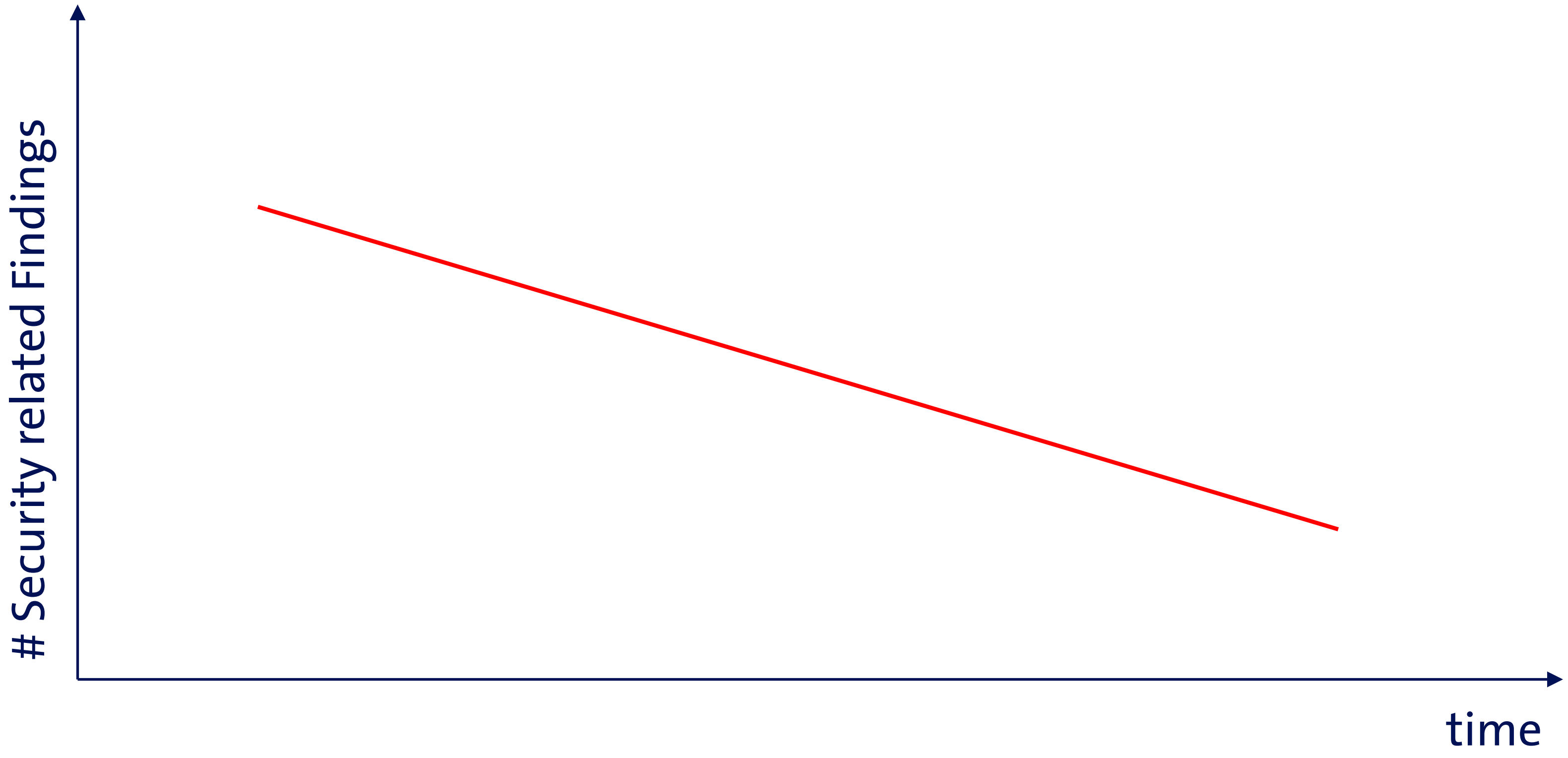
Most of the time this is what we see



Findings per phase

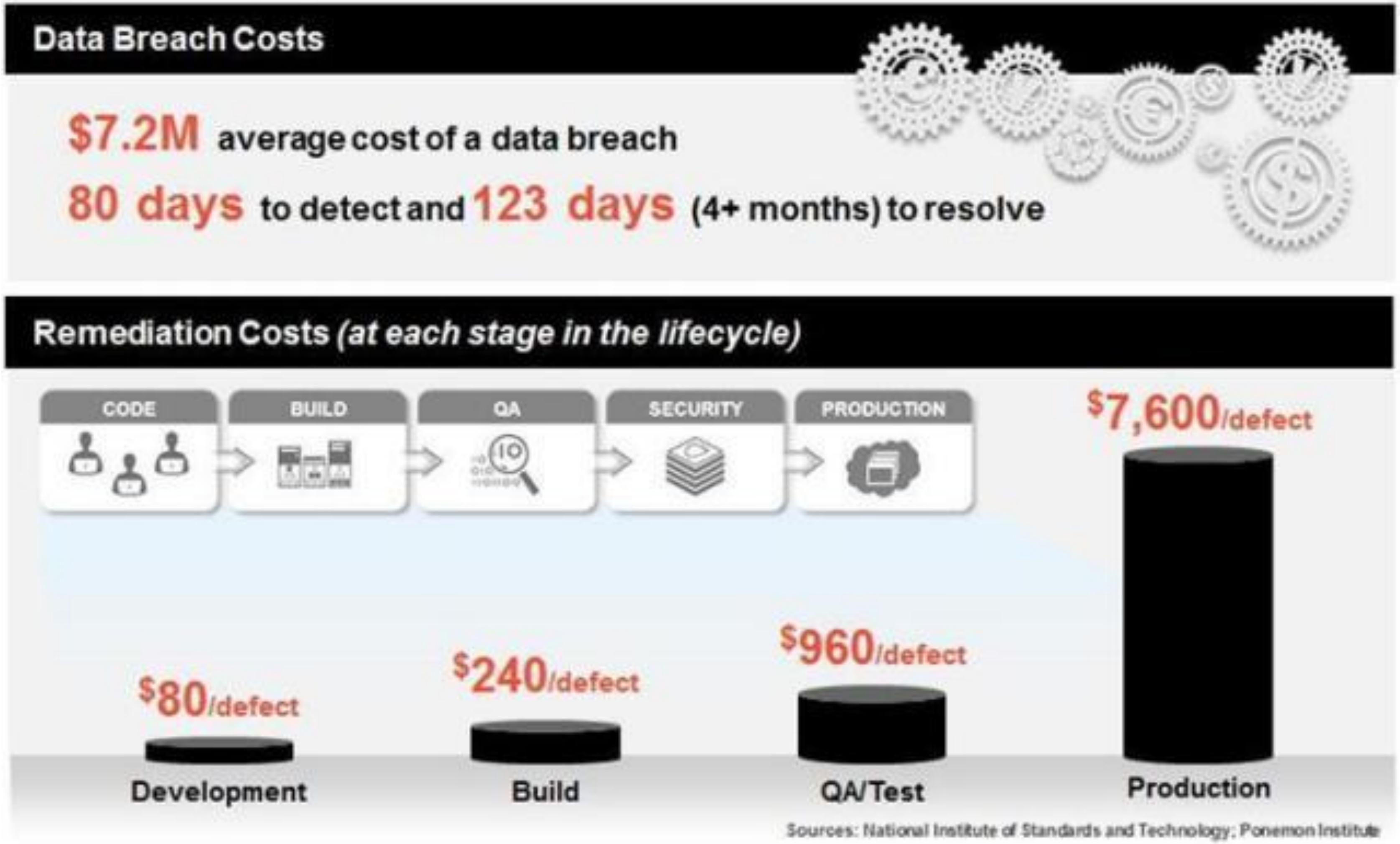
Considering mitigation costs: This is desired





SSDLC & SCA can save you $s

The earlier we find a vulnerability the cheaper its mitigation



Data Breach Costs

$7.2M average cost of a data breach

80 days to detect and 123 days (4+ months) to resolve

Remediation Costs *(at each stage in the lifecycle)*

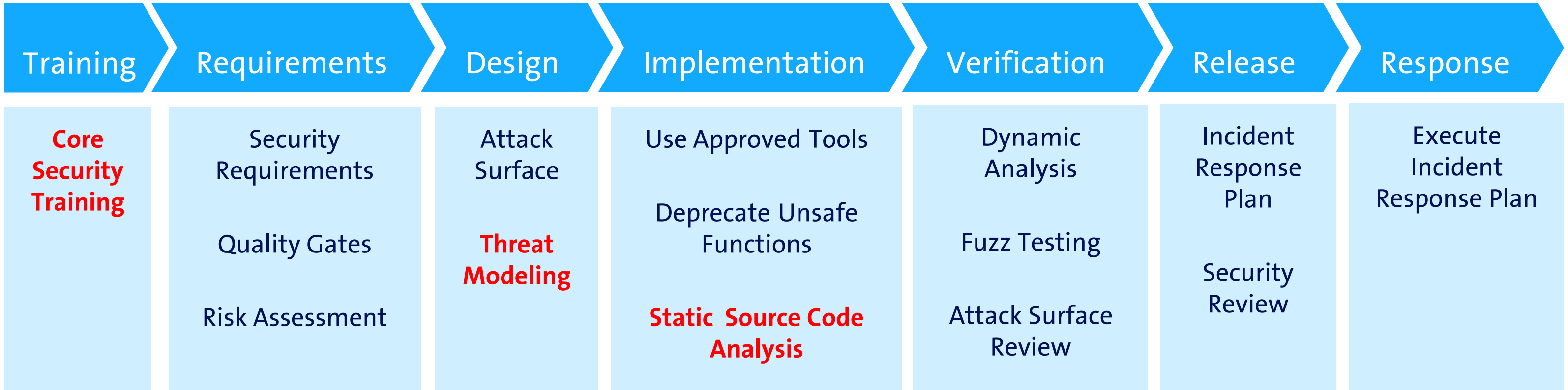
Sources: National Institute of Standards and Technology; Ponemon Institute

Fixing a flaw in production

Sometimes dangerous!

SSDLC – Framework

A formal security program



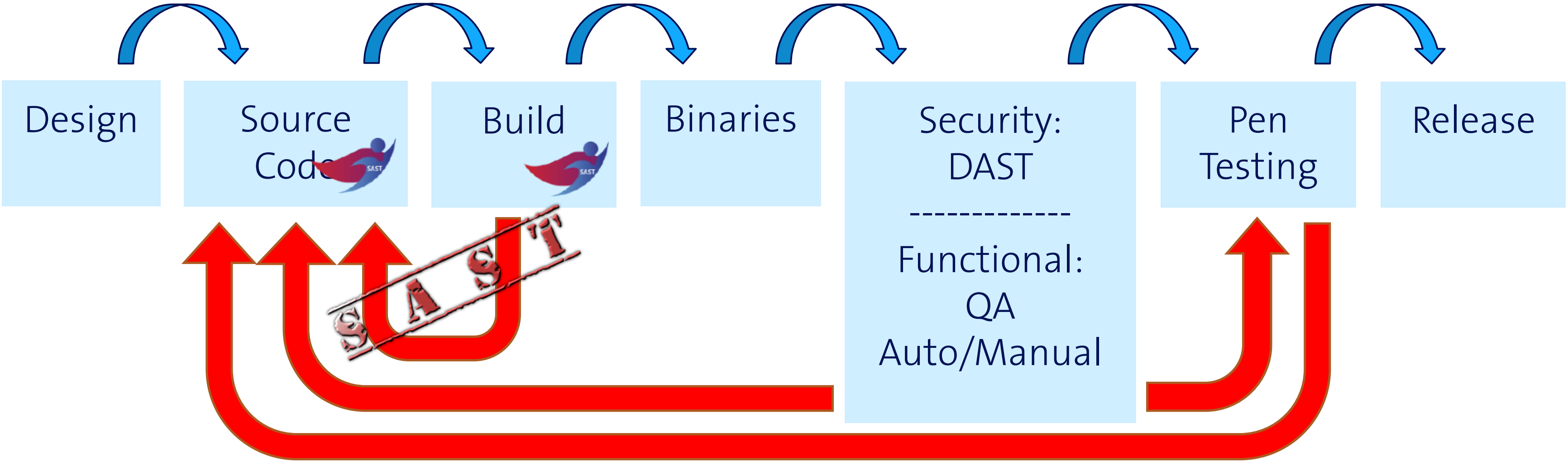
Training	Requirements	Design	Implementation	Verification	Release	Response
Core Security Training	Security Requirements Quality Gates Risk Assessment	Attack Surface **Threat Modeling**	Use Approved Tools Deprecate Unsafe Functions **Static Source Code Analysis**	Dynamic Analysis Fuzz Testing Attack Surface Review	Incident Response Plan Security Review	Execute Incident Response Plan

Microsoft SDL (https://www.microsoft.com/en-us/sdl/)

Comparison of different Frameworks:
http://www.opensecurityarchitecture.org/cms/images/OSA_images/SDLC_Comparison.pdf

SSDLC – Framework

Who feeds back into the code?

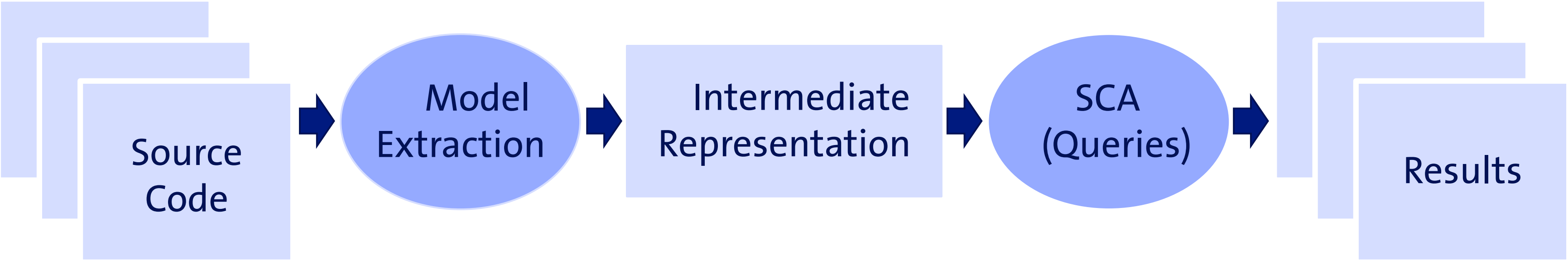


Static Code Analysis
Theories Overview

- Static vs. Dynamic Application Security Testing.
- Whitebox Approach.
- Needs the sources.
- Ranges from simple style checks , to buffer overflows, memory leaks, up to higher level security vulnerabilities.
- Similar checks might be already carried out by compiler.
- Available tools language specific or multi language.
- Open source tools
  - Lint, Checkstyle, Findbugs, PMD (Java)
  - FXCop / StyleCop (C#)
  - Cppcheck (C++)

Static Code Analysis

Process



Source Code

Model Extraction

Intermediate Representation

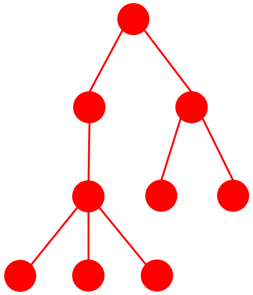
SCA (Queries)

Results

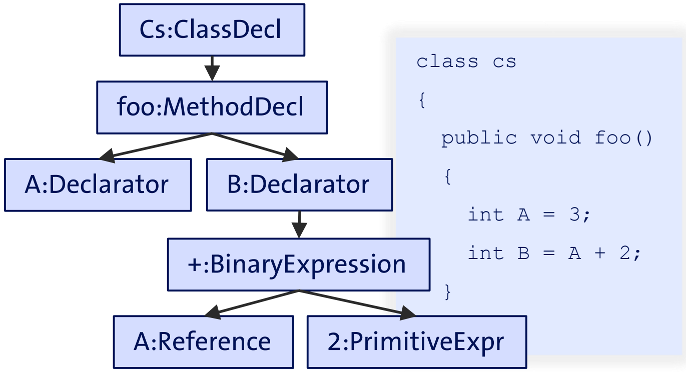
Symbol Table

Name	Kind	Location
add	Method	Helper.java
result	MemberVar	Helper.java
sum_a	Parameter	Helper.java
sum_b	Parameter	Helper.java

Abstract Syntax Tree (Tokens)



Data Flow Graph (DFG)



Cs:ClassDecl

foo:MethodDecl

A:Declarator

B:Declarator

+:BinaryExpression

A:Reference

2:PrimitiveExpr

```
class cs
{
  public void foo()
  {
    int A = 3;
    int B = A + 2;
  }
```

Open Source Tools
4 SAST

Multi Language Support

- Visual Code Grepper (C#, VB, C++, PHP, Java)
- YASCA (C/C++, Java, JavaScript), offers integration of other Tools

JAVA: OWASP LAPSE+

PHP: RIPS, DevBug

C/C++: Flawfinder, CppCheck

Ruby on Rails: Brakeman

Python: PyLynt

Vulnerabilities that we find

- OWASP Top 10, SANS 25

- E.g.
- XSS (stored, reflected, DOM based)
- Code injections
- LDAP injections
- SQL injection
- Sensitive data stored insecure (credentials in logfile)

- Vulnerabilities are ordered by severity (High, Medium, Low, Info)

Some lessons learned

We are now running the program for one year

- Developers want to deliver secure software
- There needs to be governance (sec champion, process, training)
- There needs to be automation
- Analog to other project disciplines the process has to be lived
- When projects under pressure security is left out first (NFR)

- If projects can use any technology they want it might be hard to find a scanning solution.

Q&A

- Does anyone have set up / plan to set up an SSDLC?
- Do you have experience with static analysis tools?
- Was there a tool missing?
- Any questions?
- ...

Kontakt

Frank.Bennewitz@swisscom.com

www.swisscom.com