Code Review: Prinzipien und Grenzen

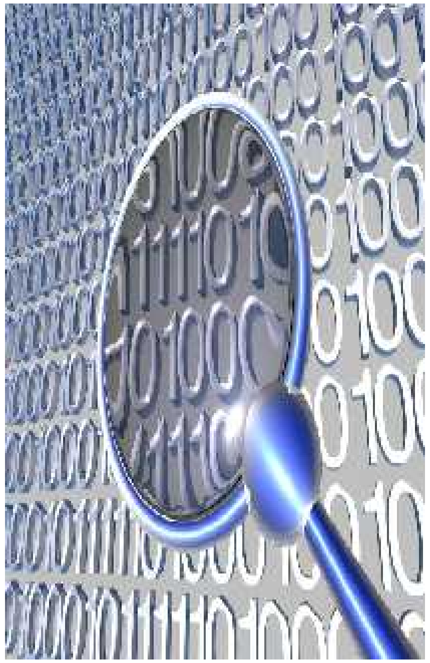
Dr. Bruce Sams
OPTIMAbit GmbH

OWASP
07.11.2012

The OWASP Foundation
http://www.owasp.org

Agenda

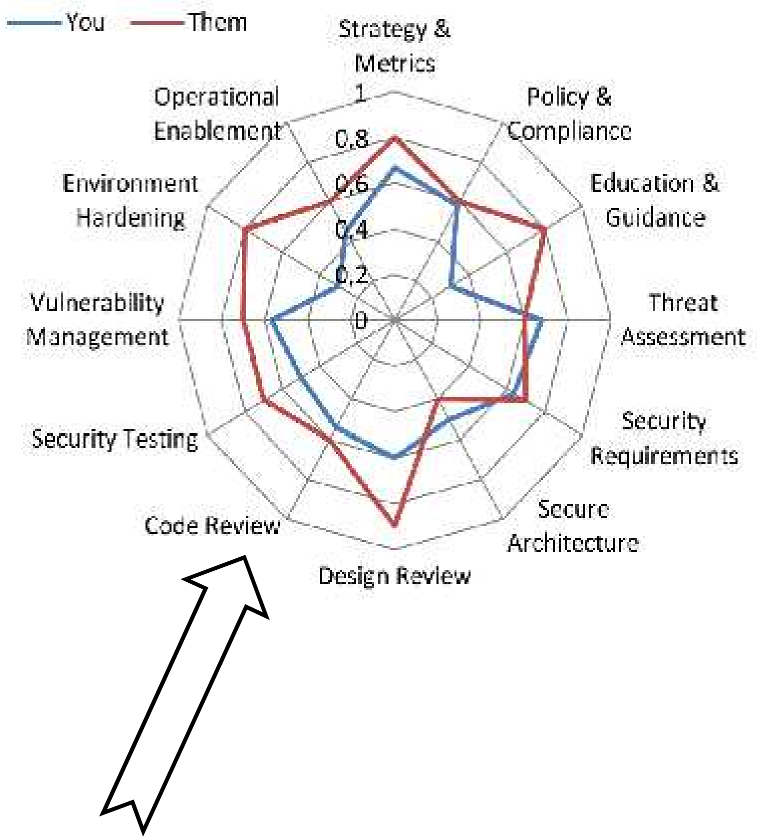
- Explain what kind of problems a review can and cannot identify
- Discuss the practical limits of code review
- Discuss when a review should be performed and by whom
- Not a product based discussion, but will use FindBugs as an example, as it is opensource and free.



Code review in context of the secure SDLC

Code review is a process

- Code review is the complete process of searching for problems in code and reporting them or integrating into the secure SDLC
- Code analysis is the detection of problems in code.
- Static code analysis is performed using tools or by humans
- Tools and humans both have limits: what are they in this context?



OWASP

Problem Categories

Category	Examples	Difficulty	Strategy
Conventions	naming, formatting	1	•Patterns
Structure	cyclomatic complexity, affine/afferent binding, package dependencies, etc.	2	•Patterns
Implemen-tation	null pointer, endless loop, unreachable code, dangerous API calls	2 - 4	•Patterns •Stack Analysis
Security	Authorization, authorization, url encoding, injection, sessions, configuration	4 - 5	•Patterns •Stack Analysis •Data Flow •Business Logic

SOME EXAMPLE CODE PROBLEMS

Some Simple Examples

Null Pointer Exception

```
String s = null;

if(s != null || s.length() > 0)  //evaluate s.length()

if(s == null | s.equals(""))     //evaluate s.equals()
```

Little Bug Patterns

```
int x = 1;

int y = x<<32; //bit shift more than 31 places
```

Some Simple Examples

Bad/Incorrect Method Invocation

```
String s = "hello  ";  //extra white space
s.trim();
someMethod(s);  //should use  s = s.trim()

s = s.trim();
```

What is reality?

- False negative (FN): Non-detection of an existing problem
- False positive (FP): Detection of a „problem“ that is not really a problem
- True positive (TP): Detection of a real problem.
- Flagged Non-Conformity (FNC): Indication that a problem might exist at a given code location.

- Most FNCs are the result of turning on too many filters in a tool (e.g. implement Serializable in Java).
- This can lead to thousands of FNCs:

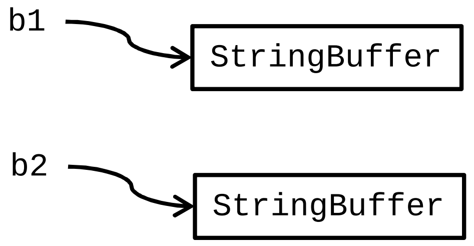
  FNC/TP >= 1000 !

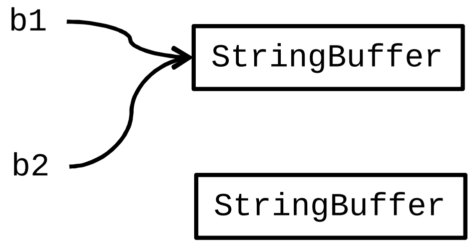
False/True Positive?: Simple Example

```
int getLength(int i) {
   String s = null;
   switch (i) {
      case 1:
         s = "hello";
         break;
      case 2:
         s = "goodbye";
         break;
   }
   return s.length();  //NullPointerException?
}
```

False Negative: Pointer Complexity

```
String s1 = request.getParameter("name");
StringBuffer b1 = new StringBuffer();
StringBuffer b2 = new StringBuffer();


//Do something here  . . .


b1.append(s1);
String s1 = b1.toString();
String s2 = b2.toString();
```





Question: is s2 safe to use?

Further Issues and Complications

Maps and containers are generally difficult to handle.

```
String name = request.getParameter("user");
map.put("USER", name);
map.put("USER", someOtherString);


map.get("USER");  // tainted? Complete Map?
```

Dynamic Class Loading

Dynamic class loading and reflection complicates knowing which classes will be instantiated at runtime.

```
String myClass = request.getParameter("MyClass");
Class class = Class.forName(myClass);
Object o = class.newInstance();
```

Pointer uncertainties add to the reflection problem (e.g., suppose the className String comes from a Map)

False Negative: Authentication Logic

```
String userType = request.getParameter("type");
if (userType.equals("NormalUser")){
   setUserPermisions("Normal_Permissions");
} else {
   setUserPermissions("Admin_Permissions");
}
```

How can a tool understand the business logic?

How can a tool interpret the Permission Strings?

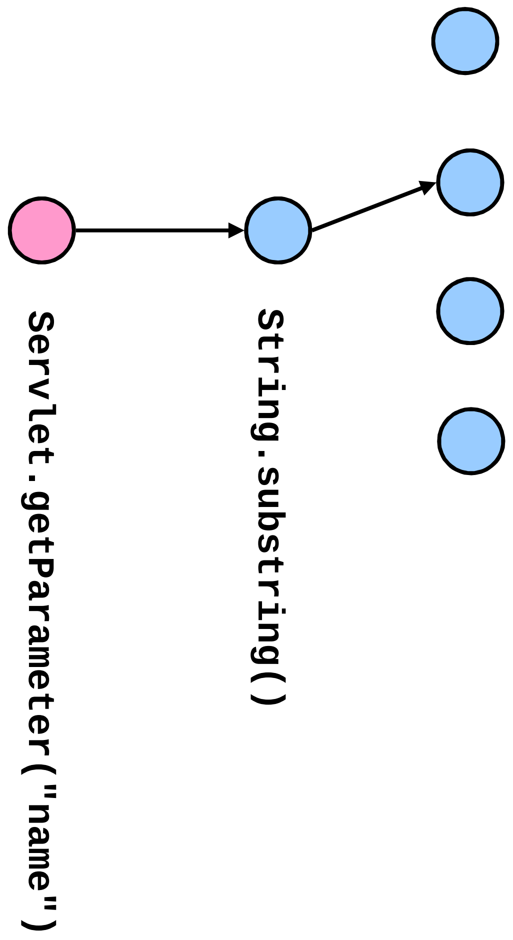
DATA FLOW ANALYSIS

Tracing Tainted Data

Untrustworthy data is „Tainted“

An application‘s call graph can be large and difficult to model:

- 10 steps with 10 branches gives ~ 10^{10} nodes.
- A 1KB memory per node, a full model requires 10^{13} Bytes = 10^4 GB RAM.
- Hard to solve the general problem completely.



Source/Sink Example: SQL Injection

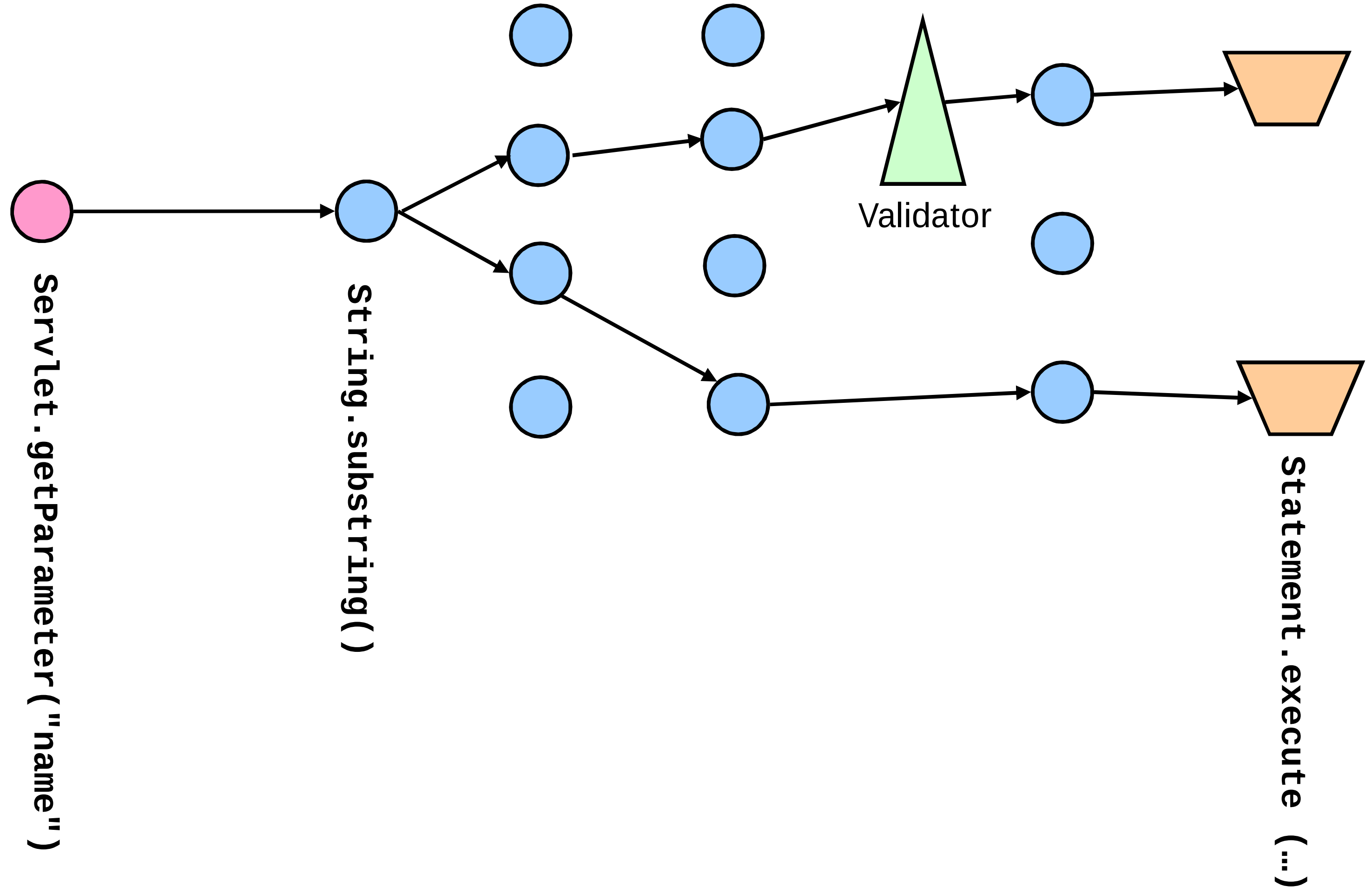
```
String s = request.getParameter("name");

Connection connection = …;

String q = "'SELECT * FROM Users WHERE NAME ='" + s +
  "'";

connection.executeQuery(q);
```

- Source = Manipulated Information in Request

```
name = xxx' OR 1 = 1;--
```

- Sink = executeQuery

Tracing Tainted Information



FINDBUGS EXTENSIONS FOR SECURITY ANALYSIS

New FindBugs detectors to improve review

Focus on security issues

- 31 bug patterns from “CERT Oracle Secure Coding Standard for Java”
  - MSC01-J. Do not use insecure or weak cryptographic algorithms
  - MET04-J. Ensure that constructors do not call overridable methods
  - SER10-J. Do not serialize direct handles to system resources
  - . . .

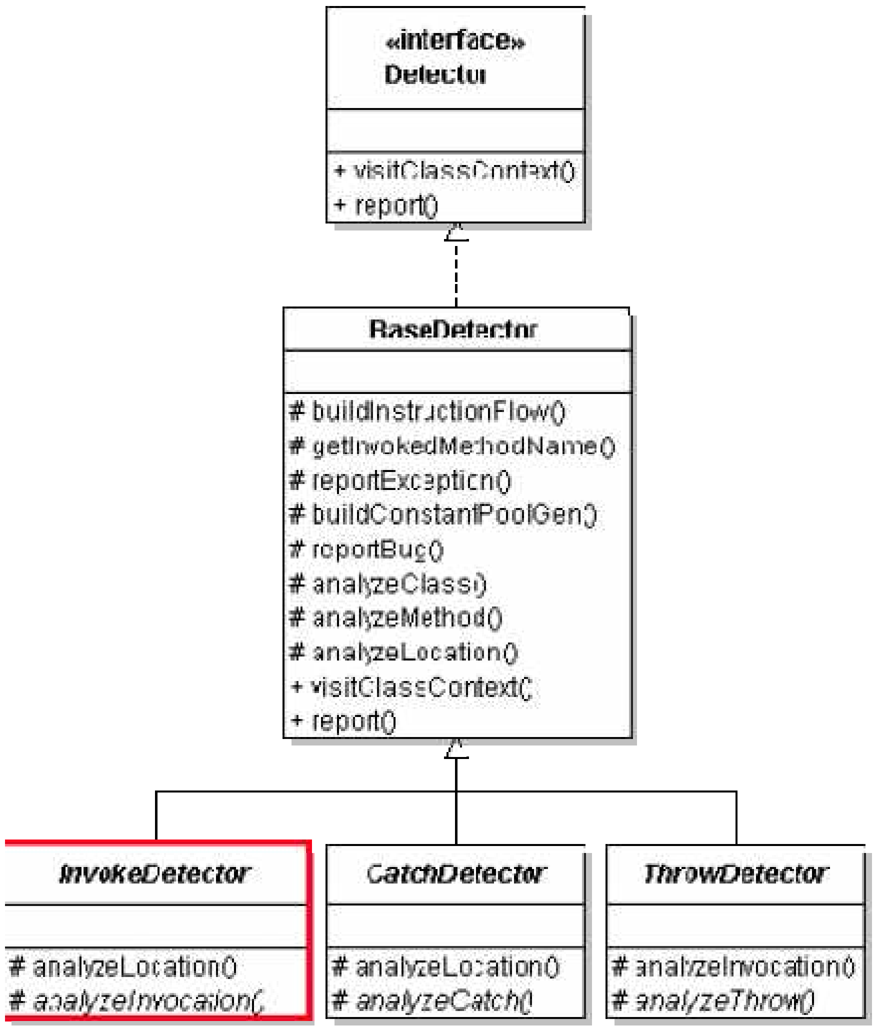
- 5 further common bug patterns
  - SQL Injection
  - HTTP Response Splitting
  - Command Injection
  - . . .

FindBugs Detector Hierachy

FindBugs

+ Easily extensible
+ OpenSource

– Only Java
– Weak Taint Tracking



Analyse WebGoat with OPTIMAbit Detectors

More bugs, but also more false positives

- 3 Installations:
  - FindBugs without Plugins
  - FindBugs + fb-contrib
  - FindBugs + OPTIMAbit detectors
- OPTIMAbit detectors:
  - XPath Injection
  - Command Injection
  - Insecure Cryptography

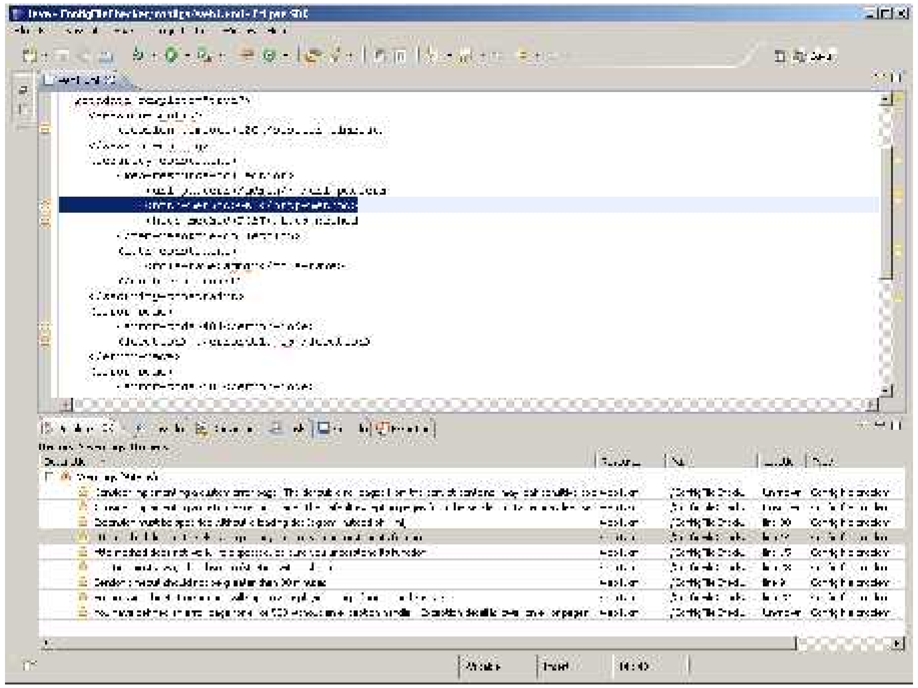
Installation	Problem Types Detected	True Positives	False Positives
No Plugins	3	15	0
fb-contrib	4	20	0
OPTIMAbit Detectors	8	117	16

SQL Injection: 14
HTTP Response Splitting: 2

Scan config files for security issues

Scanner knows about some frameworks

- 11 configuration issues in web.xml
  - Custom exception pages
  - Session timeout
  - HTTP Methods
  - . . .
- 8 configuration issues in Spring
  - Password Hashing
  - Sichere LDAP-Kommunikation
  - Method security
  - . . .



OWASP

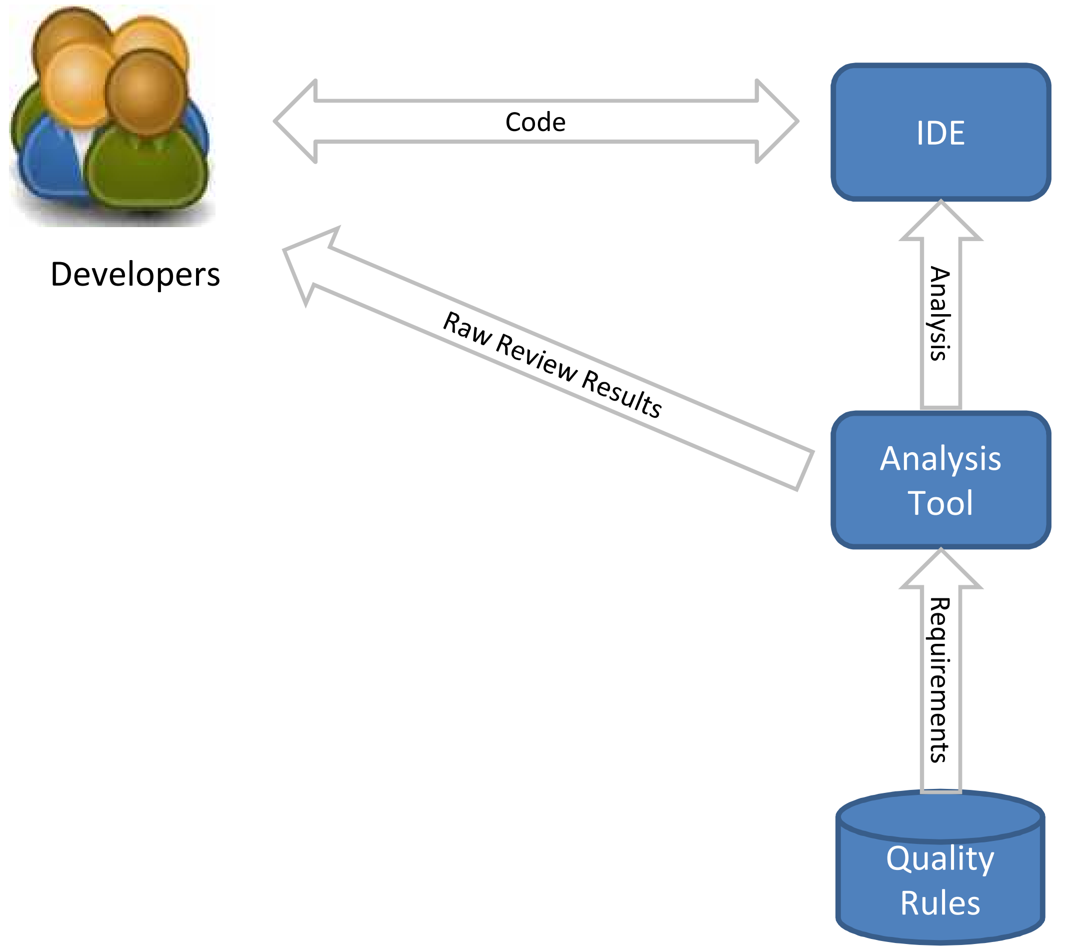
Practical issues in selecting a tool

Topic	Comment
Support for multiple Languages	•Java/C#: very good •Objective C, JavaScript, PHP: weak •COBOL, PL1, XSLT: effectively nonexistent
Analyze source and/or binaries	•Binaries need compileable project •Compiled active pages can be linked to code.
IDE Integration Build Management	•Nightly build •Ticketing •Code markers
Framework Support	•Data flow analysis •Input methods, validators for frameworks. •Dependency injection

CODE REVIEW AS A PROCESS

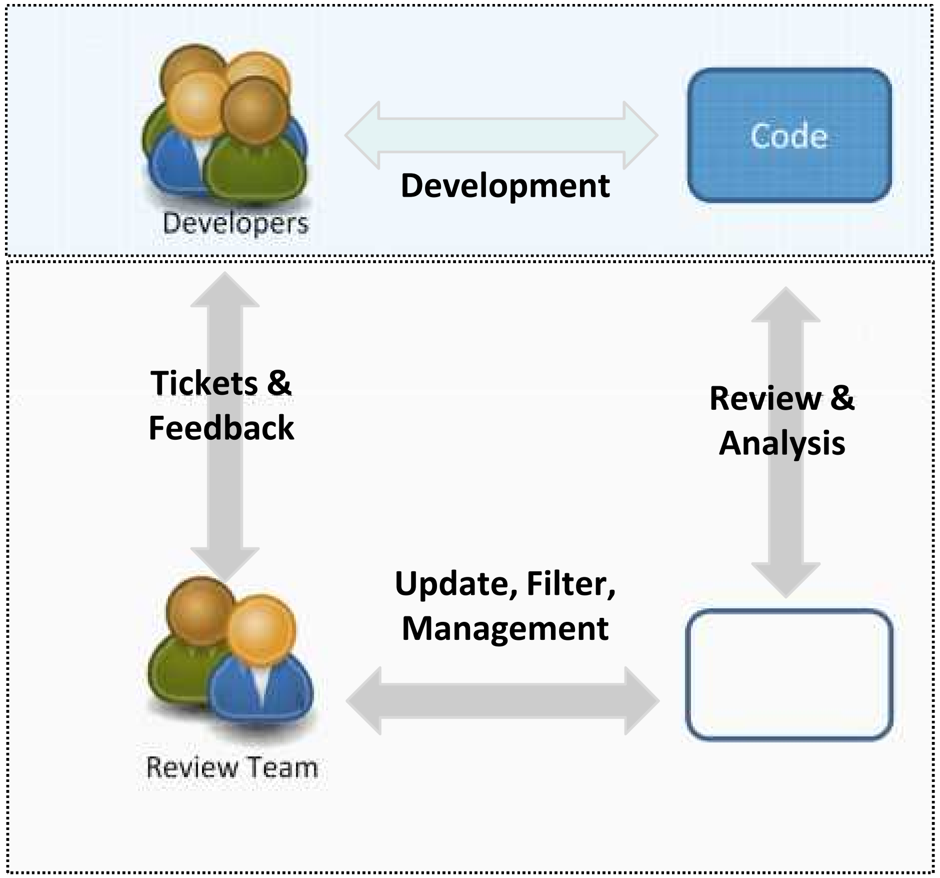
Scenario: Local Review (Worst Case)

- Developers are overloaded with tool output
- No central learning or feedback
- No independent review
- Unstructured



Scenario: Expert Review Team

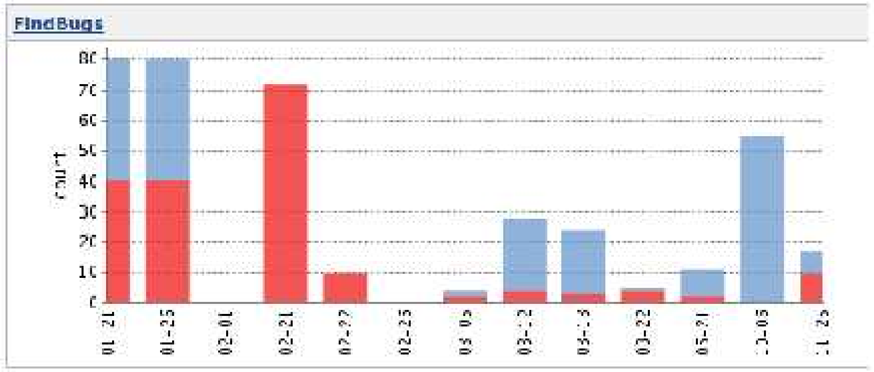
+ Developers see no false positives
+ Central learning & feedback
+ Independent review
+ Structured



Metrics for code review

Metrics are useful if correctly chosen

- Select useful metrics: Errors/KLOC, Total Errors, Security Errors.
- Metrics are good for tracking individual projects
- Metrics are hard to compare across frameworks, languages, analysis tools and application types.



OWASP

Conclusions

Static analysis is a powerful method for examining code that can detect many problems that human reviewers would not find.

The quality of the results depends strongly on the tools, code type and reviewer.

Setting up an effective code review process at the enterprise level requires experience and time.

Vielen Dank

für Ihre Aufmerksamkeit