OPTiMA..bit
business information technology

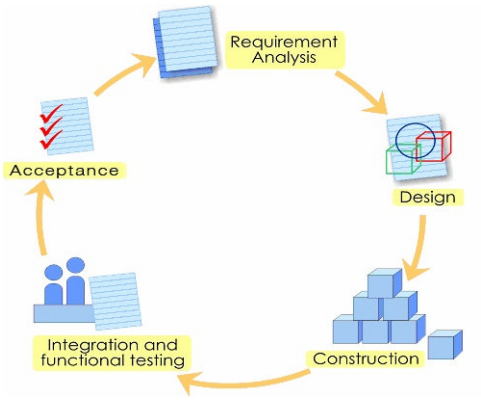
Secure Software Development

Dr. Bruce Sams **bruce.sams@optimabit.com**

Rettet die Anwendungen!

The Software Development Life Cycle

Software development takes place within a "Software Development Life Cycle" (SDLC)



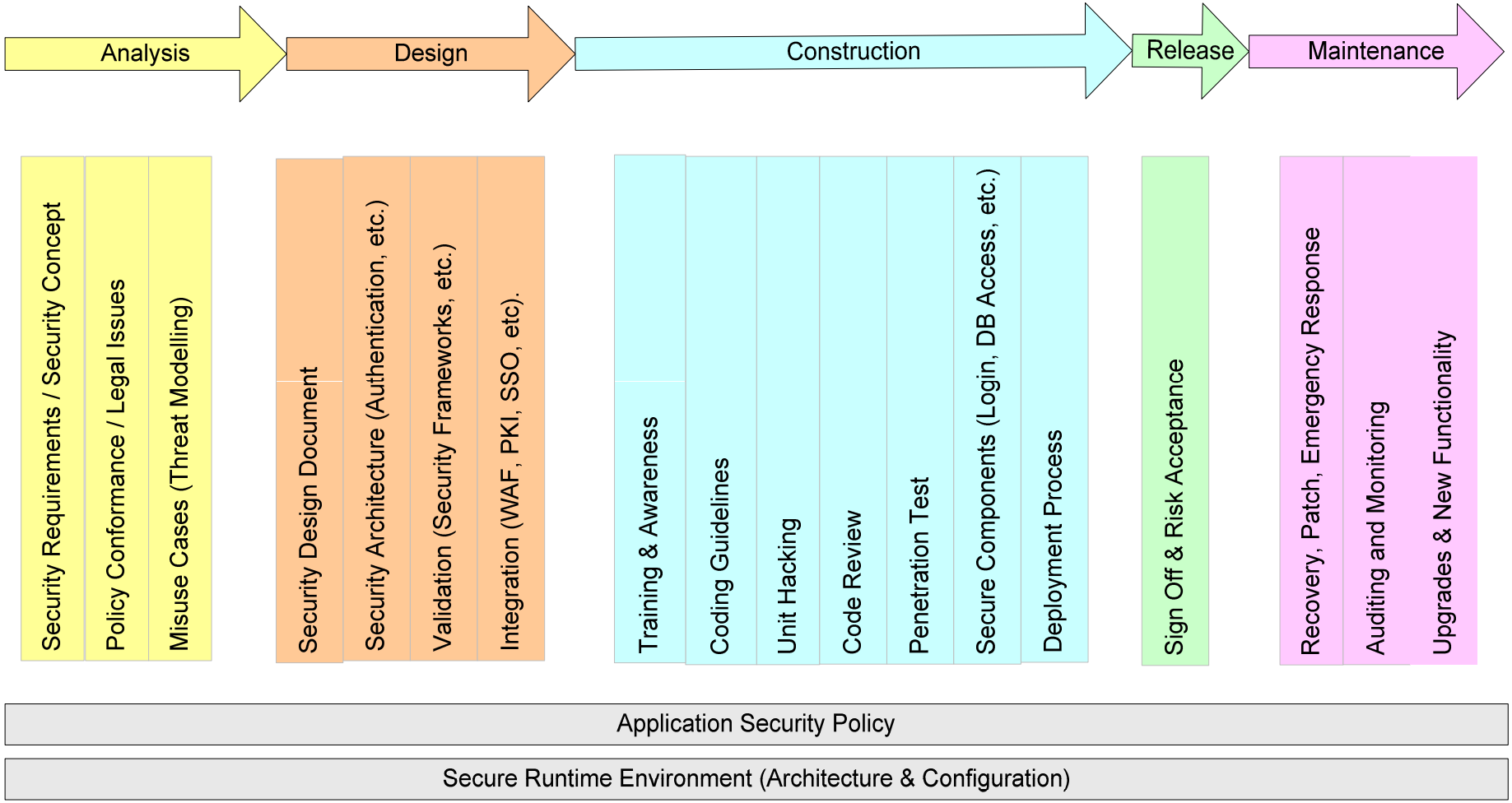
Security should be integrated into the SDLC, so that security is "built in" from the beginning and can be maintained over the lifetime of the software.

There is no "standard" for the secure SDLC.

Several attempts at a "standard" have been made, e.g. CLASP, BSI (Build Security In)

Each company must create a secure SDLC that fits into their development process (V, RUP, Agile)

Aspects of OPTIMA's secure SDLC



OPTIMA..bit
business information technology

Application Security Policy

Application Security Policy

An application security policy defines, at a high level, what requirements a secure application must fulfill.

A good policy balances abstract and concrete.

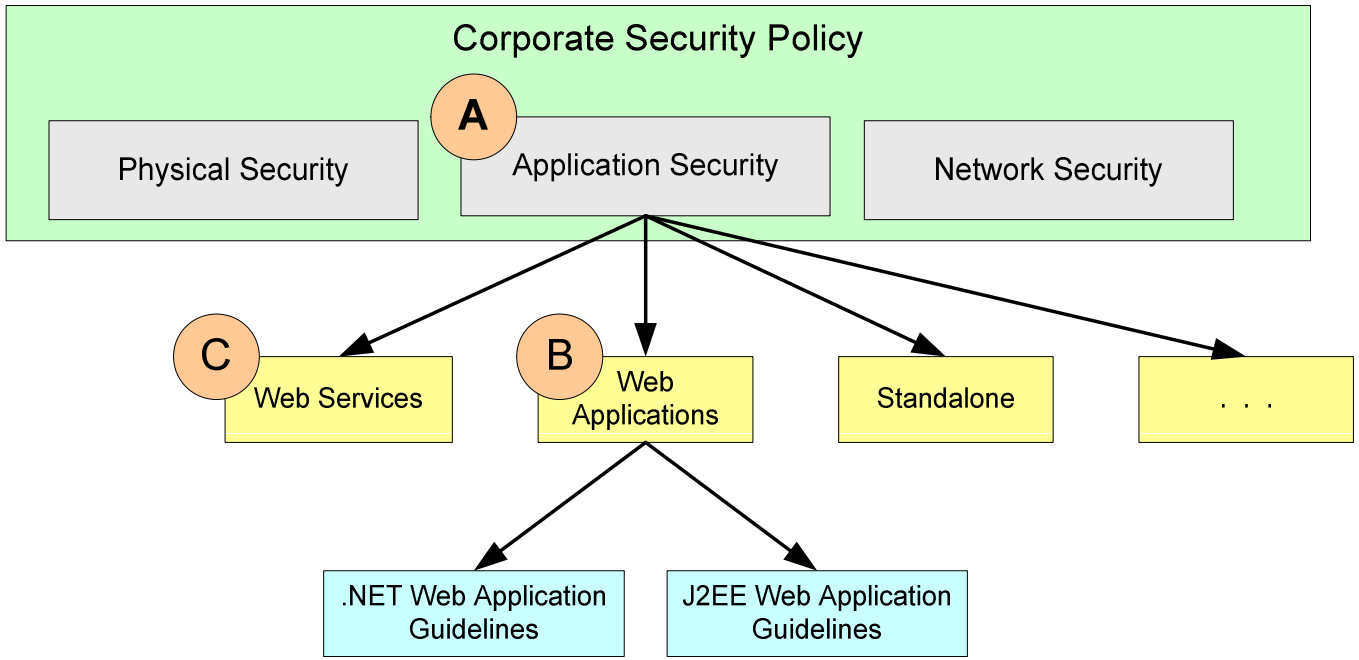
Too abstract (typical security expert document):

```
"Systems must be protected according to §2.4.3 of ISO/IEC
 27002 ..." (Role based Access Control)
```

Too concrete (typical Java developer):

```
"Do not use HttpRequest.isUserInRole("admin") in an
 Internet facing servlet under JDK 1.4"
```

OPTIMA uses a hierarchical policy structure



Primary Policies

Secondary Policies

Guidelines

Defining Security Requirements

Most projects begin with little concrete information about security requirements.

This is normal, because the focus is on business requirements.

Security Concept (compare to "Fachkonzept")

- **The project managers should be required to create the initial "security concept" document**
- **Warning: attempts to produce such a document without clear instructions will result in failure and aggravation on all sides.**
- **Therefore: every company should have a template for the security concept.**

OPTIMA..bit
business information technology

Code Review

All about code review

Fact: Every piece of software can have errors. Some errors cause security problems.

Conclusion: If you write software, you have security problems

- **Explain what techniques are used in a code review**
- **Show what kind of problems a review can identify**
- **Discuss the practical limits of code review**
- **Discuss when a review should be performed and by whom**

Security of Old vs. New Languages

▪C, C++, Perl	▪Java, C#
Manual memory management	Automatic memory management
Buffer overflows	No buffer overflows
Lots of Gotchas, e g. if(x = 1)	Fewer Gotchas, e.g. if (x == 1)
Errors/KLOC =~ 1 - 10	Error/KLOC =~ 0.1 - 1

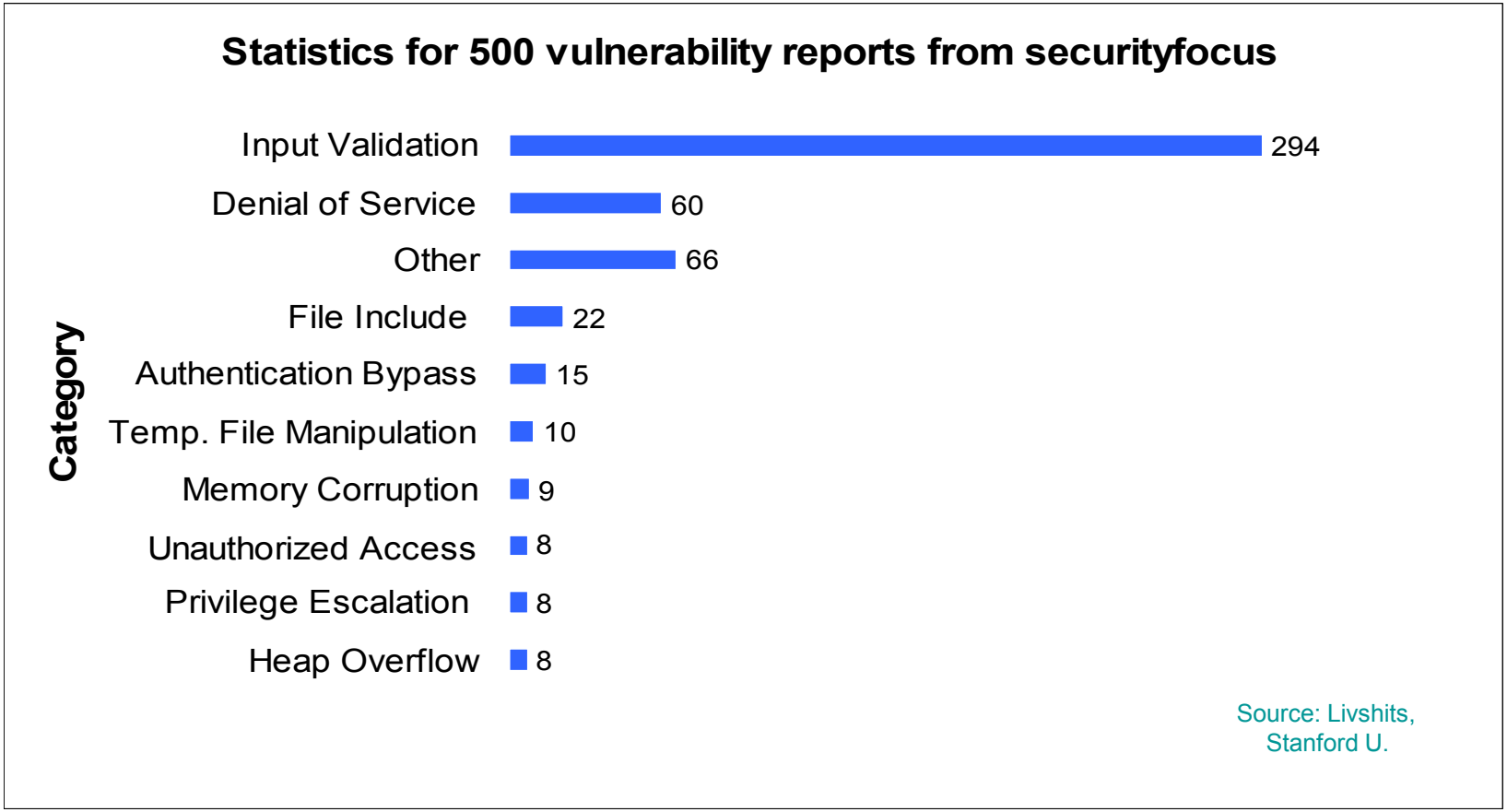
Web applications with browser front ends (JSP, ASP) are particularly difficult to secure.

Problem Categories

Category	Examples	Complexity of Identification
Conventions	naming, formatting	1
Structure	cyclomatic complexity, affine/afferent binding, package dependencies, etc.	2
Implementation	null pointer, endless loop, unreachable code, dangerous API calls	2 - 4
Frameworks	Conformity to architecture & framework requirements.	3 - 4
Security	buffer overflow, url encoding, injection, elevation of privilege, sessions	4 - 5

Real World Vulnerabilities

Real world vulnerablilies are related to information flow, not control flow.



OPTIMA..bit
business information technology

False Positives

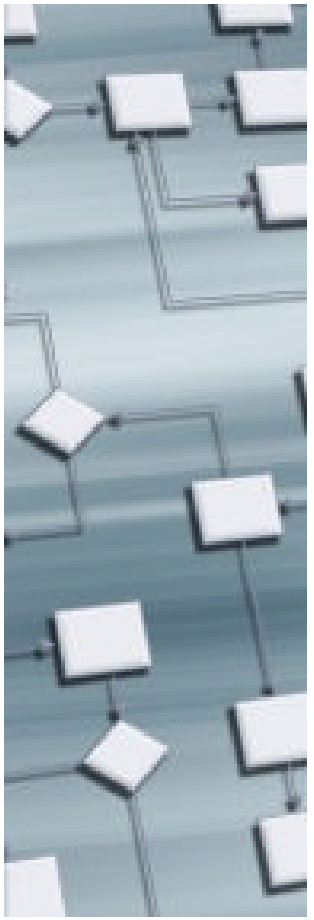
Code review usually produces a lot of false positives that need to be sorted out by inspection.

```
int getString(int i) {
   String s = null;
   switch (i) {
      case 1:
         s = "one";
         break;
      case 2:
         s = "two";
         break;
   }
   return s.length(); //NPE? Depends on allowed range of i
}
```

Data Flow Analysis

Typical problems in web applications are related to data flow.

- **Data from an external source is used without validating it first.**
- **(Mostly) simple to correct, but the places in code are sometimes hard to find!**



Source/Sink Example: SQL Injection

```
HttpServletRequest req = …;
String s = req.getParamter("name");
Connection connection = …;
String q = "'SELECT * FROM Users WHERE NAME ='" + s + "'";
Connection.executeQuery(q);
```

Source = Manipulated Information in Request

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

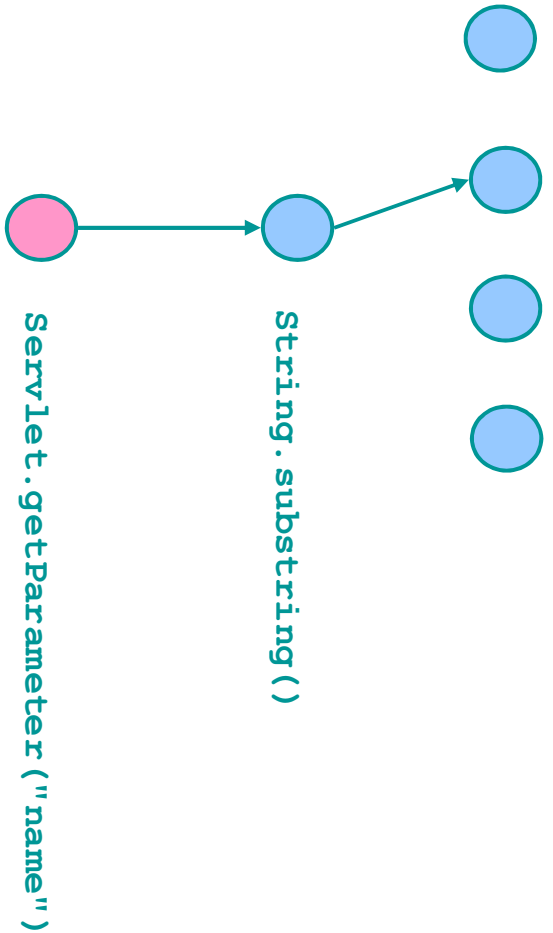
Sink = executeQuery

Tracing Tainted Data

"Tainted" data comes from sources external to the program itself and is "untrusted".

The call graph can be large:

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



OPTIMA..bit
business information technology

Model for Static Analysis in Code

"Sources" of tainted information

- **Input from a web form**
- **RSS feeds**
- **Web Services**
- **Database Information**
- **…**

"Sinks" where the information is used in a potentially "dangerous" way.

- **DB Queries**
- **File operations**
- **Directory Lookups**
- **…**

Identifying Vulnerabilities

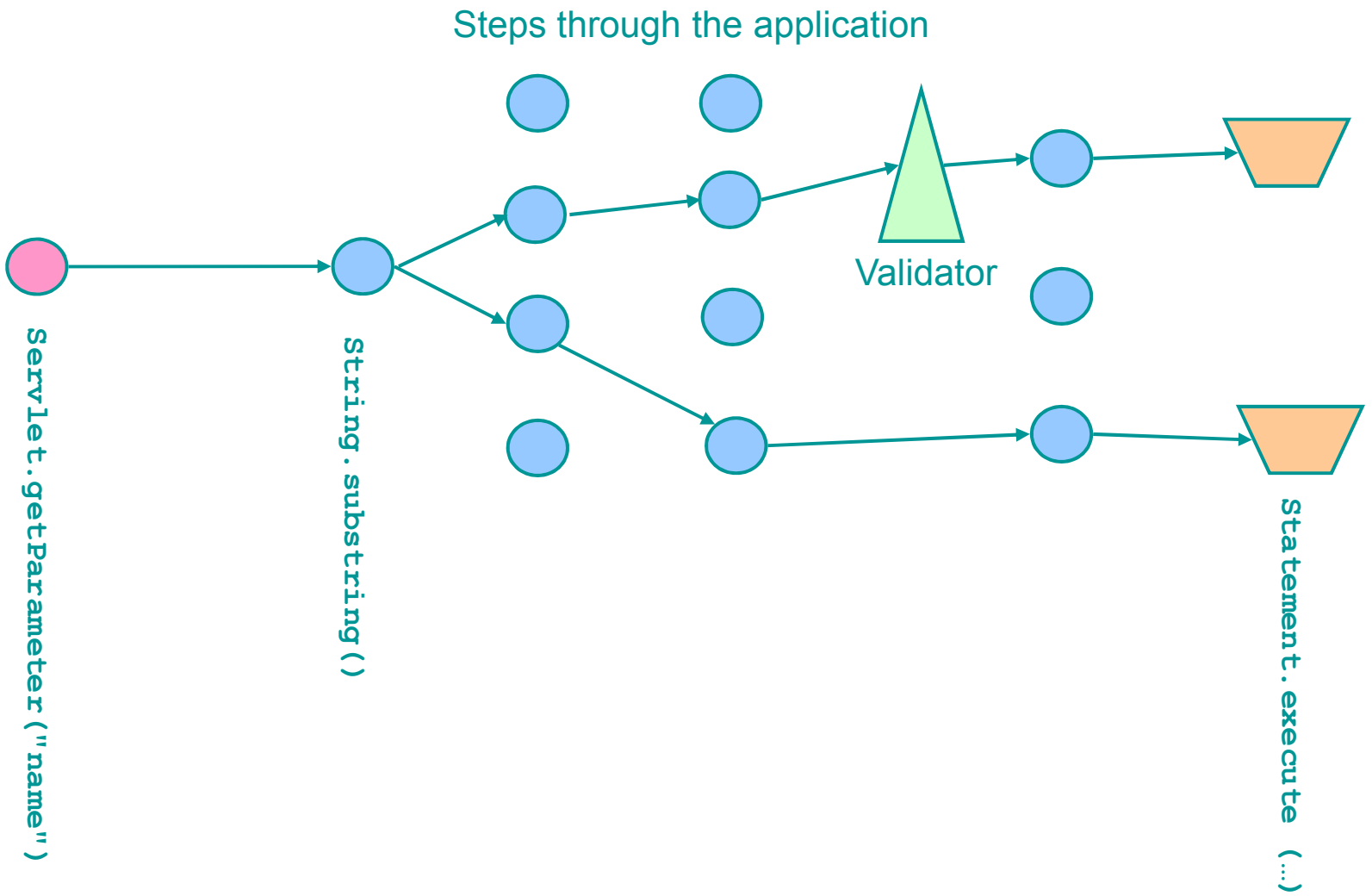
A vulnerability is identified as a place in the code where tainted information from a Source is delivered to a Sink with an exploitable weakness.

Sources
Parameter Manipulation
Hidden Field Manipulation
Cookie Poisoning
Second Order Injection

Sinks
SQL Injection
Cross Site Scripting
HTTP Request Splitting
Path Traversal
Command Injection

Example: Hidden Field Manipulation + Path Traversal = Vulnerability

Tracing Tainted Information



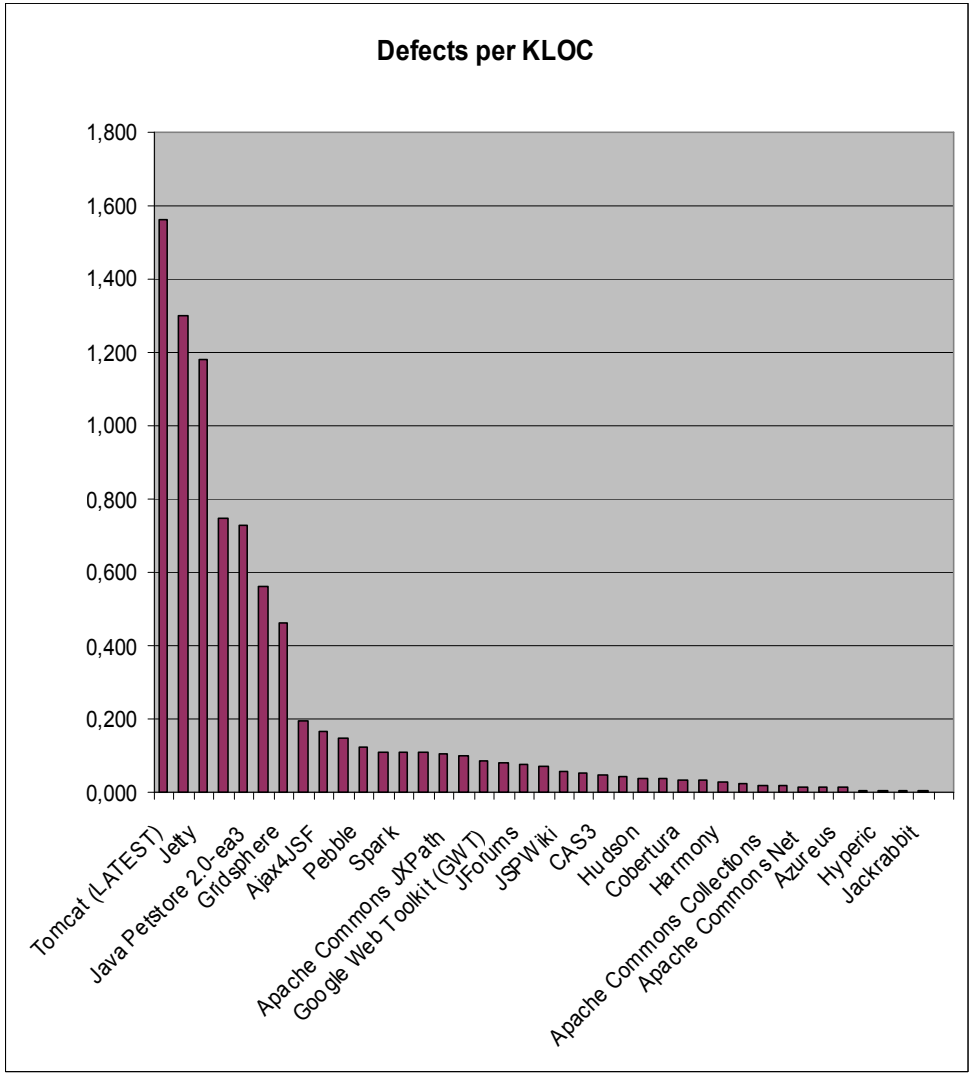
Problems to watch out for

- **False positives (detecting a error when none exists)**
- **False negatives (not detecting a error when one exists)**
- **Completeness (how much of the code was tested)**
- **Deep/Shallow paths (how many steps into the application are tested)**
- **Handling dynamic class loading and reflection**
- **Trying to solve the general problem is VERY hard. What about solving simpler, less general problems?**

How Many Defects are Detected?

A joint project between the FindBugs Group and Fortify has analyzed ca. 40 open source projects.

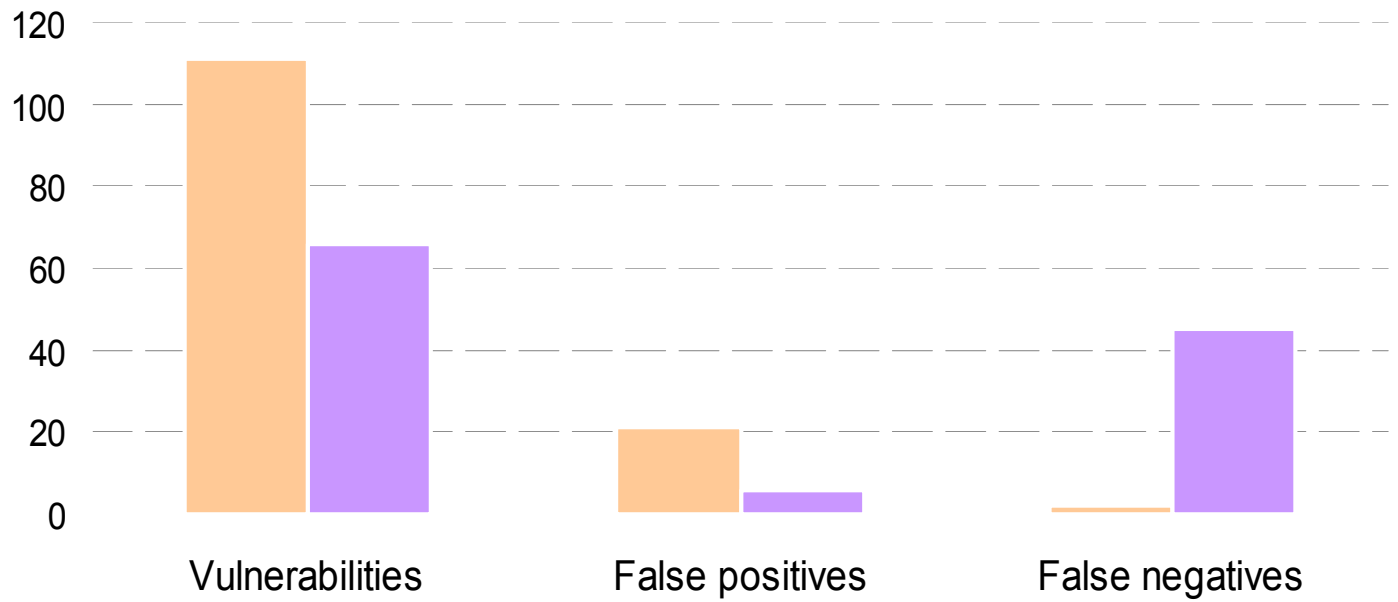
- **Typical defect rates are about 0.2 per 1000 LOC.**
- **One tested project (Net Trust, not shown on chart) was much higher (ca 10 Defects/KLOC)**



False Positives vs. False Negatives

Example: two tools (1 research, 1 commercial) analyzed the same code base (ca 300K LOC, 4M ExLOC).

The commercial tool is heavily biased toward false negatives.



OPTiMA..bit
business information technology

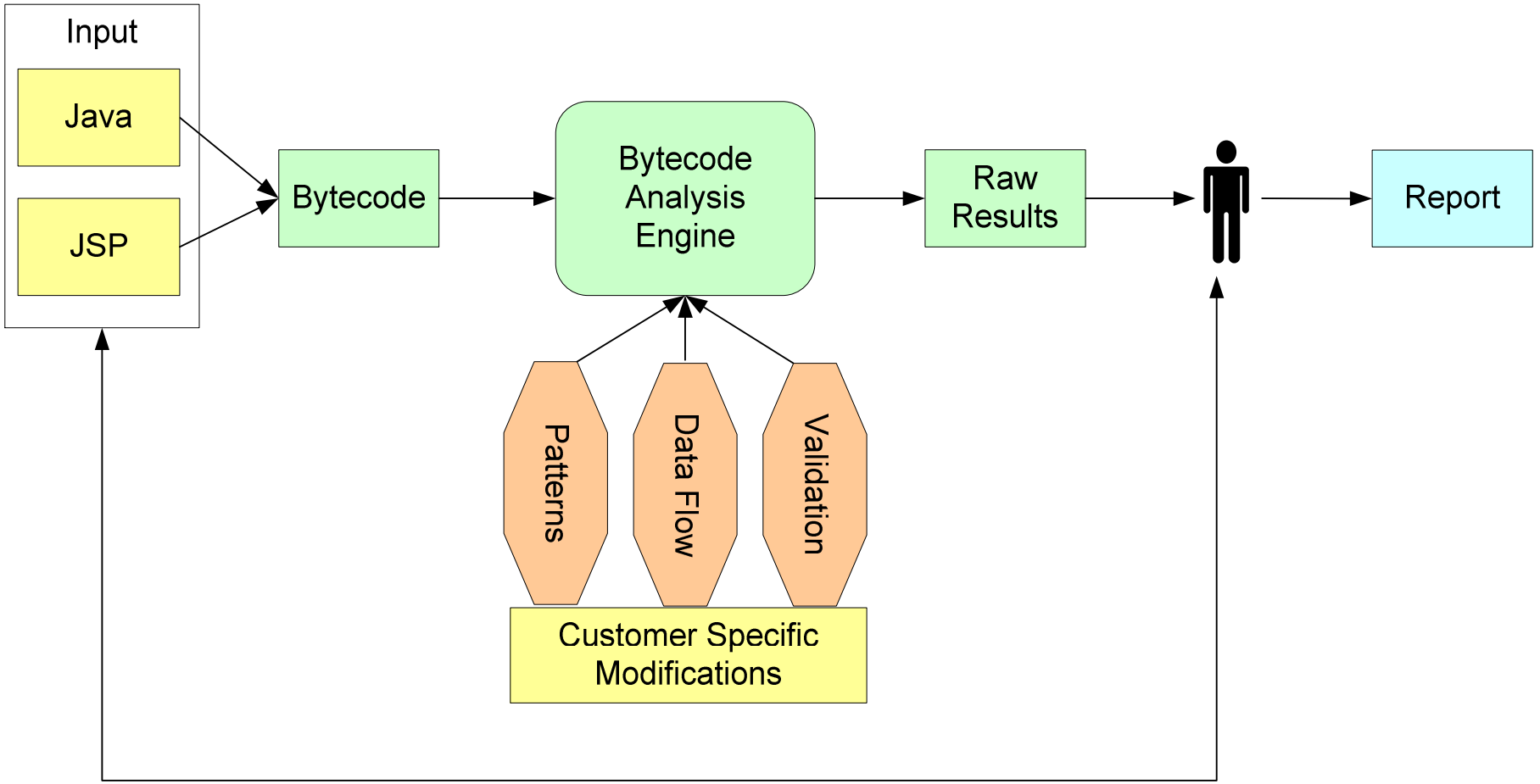
Tool Metrics

OPTIMA Bytecode Scanner

Works in bytecode, performs pattern analysis and data flow analysis.

- **Is designed to detect the maximum number of security problems, even at the expense of more false positives.**
- **Allows special modifications for, e.g. validation, that can "untaint" an object.**
- **Offers powerful analysis of cryptography and other special security topics.**
- **Proven in numerous successful code reviews for large projects**
- **Human analysis is required (!!)**

OPTIMA Bytecode Scanner



Strategies for Security Code Review

Do it yourself

- Code remains in-house
- Build up internal knowledge (requires training and updates)
- Integrate into build process
- Easy to do "quick" reviews

Outsource to specialists

- Specialists have more experience
- Specialists use multiple tools
- Cheaper ? (licenses and manpower)
- Easy to plan resources (time and manpower) for a review

OPTIMA offers code review as a service.
Fast, accurate, reliable, plannable, consistent

Conclusions

Static analysis can be complex, particularly if

- **the path depth is high and has many branches**
- **the pointer and context analysis is complex**
- **many maps are used**
- **Dynamic classes & reflection play an important role**

Conclusions

The secure SDLC is a reality, and can substantially improve the security of software development.

There is no Out Of The Box process, because the development process varies from company to company.

Customizing the process requires sensible policies and templates that are developer friendly

Code Review is a crucial aspect of the SDLC, but performing an in-depth review is hard to do.