Some Things I Have Learned About Static Analysis and Static Analysis Tools, Including Ideas on Their Role in Software Development

Paul E. Black

paul.black@nist.gov

U.S. DEPARTMENT OF HOMELAND SECURITY

SAMATE

http://samate.nist.gov/

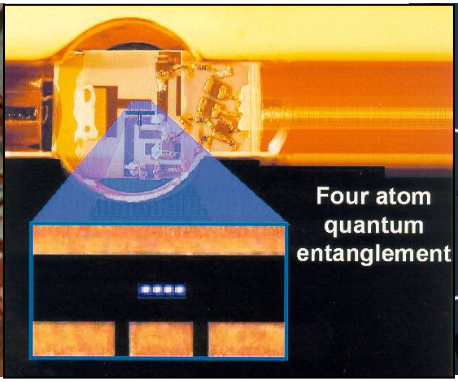
NIST National Institute of Standards and Technology • U.S. Department of Commerce

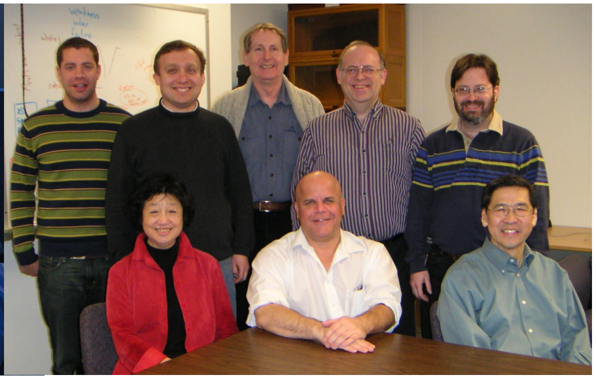
Outline

- The Software Assurance Metrics And Tool Evaluation (SAMATE) project
- What is static analysis?
- Limits of automatic tools
- State of the art in static analysis tools
- Static analyzers in the software development life cycle

What is NIST?

- **U.S. National Institute of Standards and Technology**
- **A non-regulatory agency in Dept. of Commerce**
- **3,000 employees + adjuncts**
- **Gaithersburg, Maryland and Boulder, Colorado**
- **Primarily research, not funding**
- **Over 100 years in standards and measurements: from dental ceramics to microspheres, from quantum computers to fire codes, from body armor to DNA forensics, from biometrics to text retrieval.**

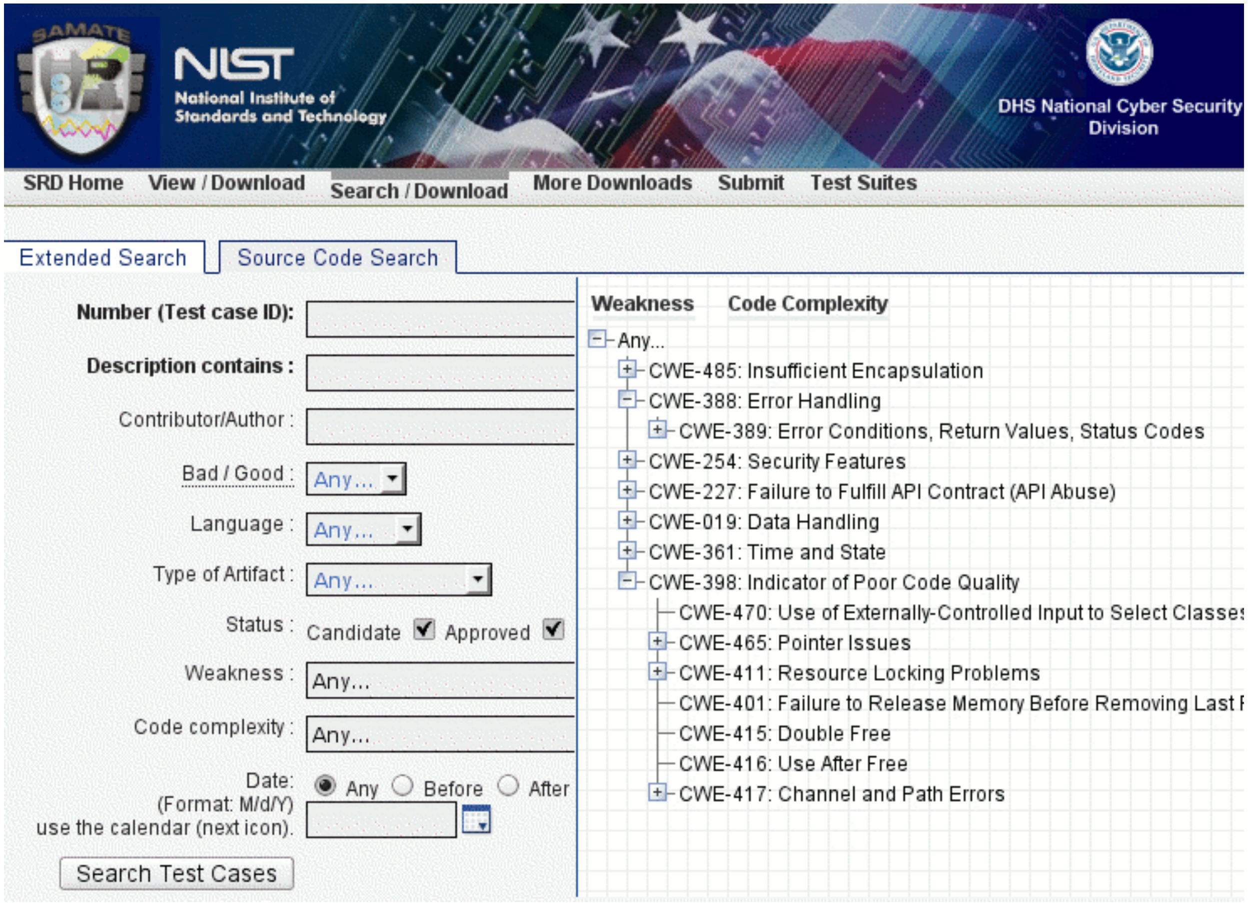



The NIST SAMATE Project

- Software Assurance Metrics And Tool Evaluation (SAMATE) project is sponsored in part by DHS
- Current areas of concentration
  - Web application scanners
  - Source code security analyzers
  - Static Analyzer Tool Exposition (SATE)
  - Software Reference Dataset
  - *Software labels*
  - *Malware research protocols*

- Web site http://samate.nist.gov/

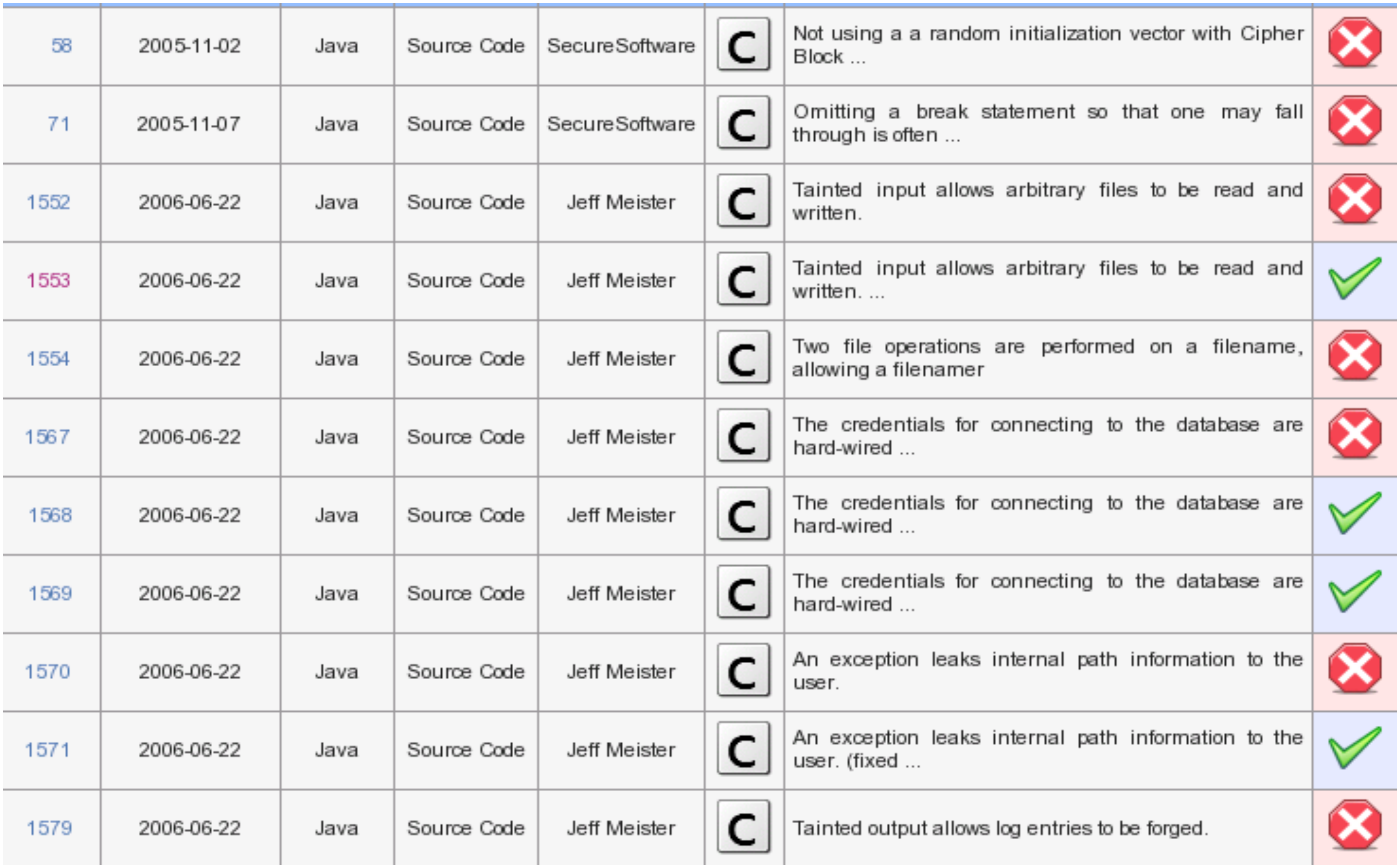
Software Reference Dataset



Number (Test case ID):
Description contains :
Contributor/Author :
Bad / Good : Any...
Language : Any...
Type of Artifact : Any...
Status : Candidate ☑ Approved ☑
Weakness : Any...
Code complexity : Any...
Date: ● Any ○ Before ○ After
(Format: M/d/Y)
use the calendar (next icon).

- Any...
  - CWE-485: Insufficient Encapsulation
  - CWE-388: Error Handling
    - CWE-389: Error Conditions, Return Values, Status Codes
  - CWE-254: Security Features
  - CWE-227: Failure to Fulfill API Contract (API Abuse)
  - CWE-019: Data Handling
  - CWE-361: Time and State
  - CWE-398: Indicator of Poor Code Quality
    - CWE-470: Use of Externally-Controlled Input to Select Classes
    - CWE-465: Pointer Issues
    - CWE-411: Resource Locking Problems
    - CWE-401: Failure to Release Memory Before Removing Last R
    - CWE-415: Double Free
    - CWE-416: Use After Free
    - CWE-417: Channel and Path Errors

- **Public repository for software test cases**
- **Almost 1800 cases in C, C++, Java, and Python**
- **Search and compose custom Test Suites**
- **Contributions from Fortify, Defence R&D Canada, Klocwork, MIT Lincoln Laboratory, Praxis, Secure Software, etc.**

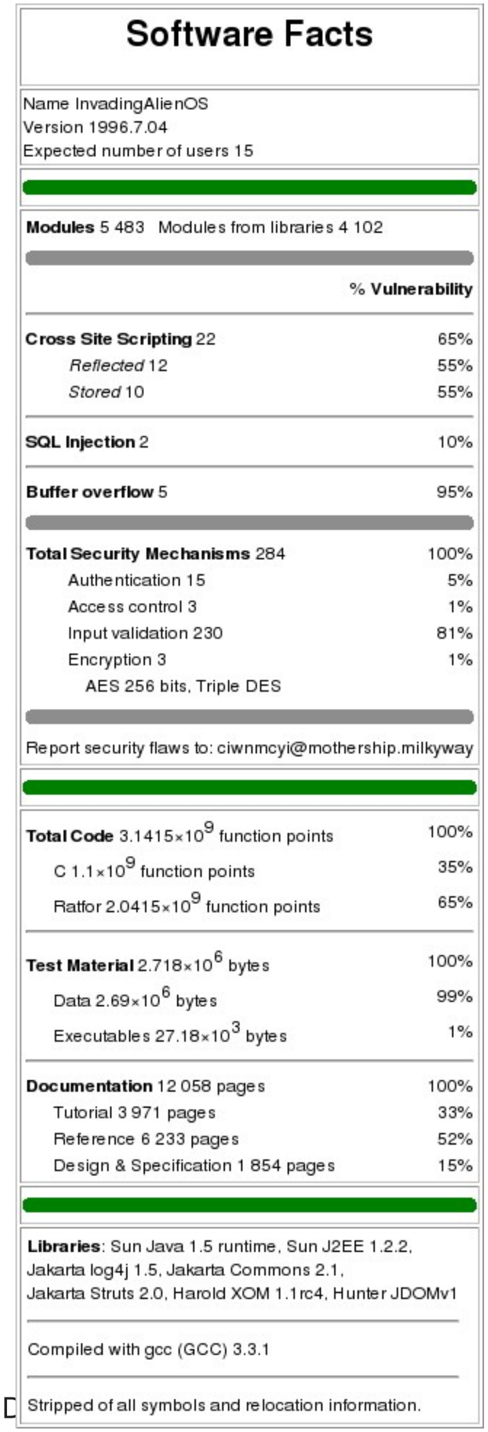


58	2005-11-02	Java	Source Code	SecureSoftware	C	Not using a a random initialization vector with Cipher Block ...	✗
71	2005-11-07	Java	Source Code	SecureSoftware	C	Omitting a break statement so that one may fall through is often ...	✗
1552	2006-06-22	Java	Source Code	Jeff Meister	C	Tainted input allows arbitrary files to be read and written.	✗
1553	2006-06-22	Java	Source Code	Jeff Meister	C	Tainted input allows arbitrary files to be read and written. ...	✓
1554	2006-06-22	Java	Source Code	Jeff Meister	C	Two file operations are performed on a filename, allowing a filenamer	✗
1567	2006-06-22	Java	Source Code	Jeff Meister	C	The credentials for connecting to the database are hard-wired ...	✗
1568	2006-06-22	Java	Source Code	Jeff Meister	C	The credentials for connecting to the database are hard-wired ...	✓
1569	2006-06-22	Java	Source Code	Jeff Meister	C	The credentials for connecting to the database are hard-wired ...	✓
1570	2006-06-22	Java	Source Code	Jeff Meister	C	An exception leaks internal path information to the user.	✗
1571	2006-06-22	Java	Source Code	Jeff Meister	C	An exception leaks internal path information to the user. (fixed ...	✓
1579	2006-06-22	Java	Source Code	Jeff Meister	C	Tainted output allows log entries to be forged.	✗

```
public class File1_bad extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType("text/html");
        ServletOutputStream out = res.getOutputStream();
        out.println("<HTML><HEAD><TITLE>Test</TITLE></HEAD><BODY><blockquote><pre>");

            String name = req.getParameter("name");
            String msg = req.getParameter("msg");
            if(name != null) {
                try {
                    File f = new File("/tmp", name);                    /* BAD */
                    if(msg != null) {
                        FileWriter fw = new FileWriter(f);          /* BAD */
                        fw.write(msg, 0, msg.length());
                        fw.close();
                        out.println("message stored");
                    } else {
                        String line;
                        BufferedReader fr = new BufferedReader(new FileReader(f));
                        while((line = fr.readLine()) != null)
                            out.println(line);
                    }
                } catch(Exception e) {
                    throw new ServletException(e);
                }
            } else {
```

Software Label

- **Software Facts should be:**
  - **Voluntary**
  - **Absolutely simple to produce**
  - **In a standard format for other claims**
- **What could be easily supplied?**
  - **Source available? Yes/No/Escrowed**
  - **Default installation is secure?**
  - **Accessed: network, disk, ...**
  - **What configuration files? (registry, ...)**
  - **Certificates (e.g., "No Severe weaknesses found by CodeChecker ver. 3.2")**
- **Cautions**
  - **A label can give false confidence.**
  - **A label shut out better software.**
  - **Labeling diverts effort from real improvements.**



Software Facts

Name InvadingAlienOS
Version 1996.7.04
Expected number of users 15

Modules 5 483   Modules from libraries 4 102

	% **Vulnerability**
Cross Site Scripting 22	65%
Reflected 12	55%
Stored 10	55%
SQL Injection 2	10%
Buffer overflow 5	95%

Total Security Mechanisms 284	100%
Authentication 15	5%
Access control 3	1%
Input validation 230	81%
Encryption 3 AES 256 bits, Triple DES	1%

Report security flaws to: ciwnmcyi@mothership.milkyway

Total Code 3.1415×10^9 function points	100%
C 1.1×10^9 function points	35%
Ratfor 2.0415×10^9 function points	65%
Test Material 2.718×10^6 bytes	100%
Data 2.69×10^6 bytes	99%
Executables 27.18×10^3 bytes	1%
Documentation 12 058 pages	100%
Tutorial 3 971 pages	33%
Reference 6 233 pages	52%
Design & Specification 1 854 pages	15%

Libraries: Sun Java 1.5 runtime, Sun J2EE 1.2.2, Jakarta log4j 1.5, Jakarta Commons 2.1, Jakarta Struts 2.0, Harold XOM 1.1rc4, Hunter JDOMv1

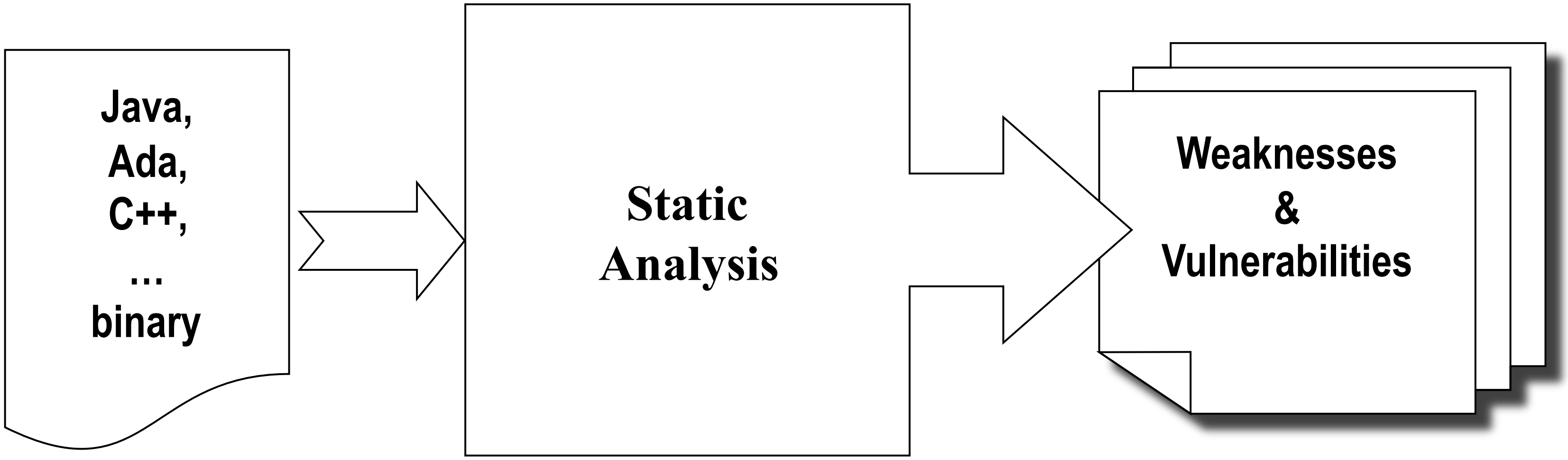
Compiled with gcc (GCC) 3.3.1

Stripped of all symbols and relocation information.

Outline

- The Software Assurance Metrics And Tool Evaluation (SAMATE) project
- **What is static analysis?**
- Limits of automatic tools
- State of the art in static analysis tools
- Static analyzers in the software development life cycle

Static Analysis



Java,
Ada,
C++,
…
binary

Static Analysis

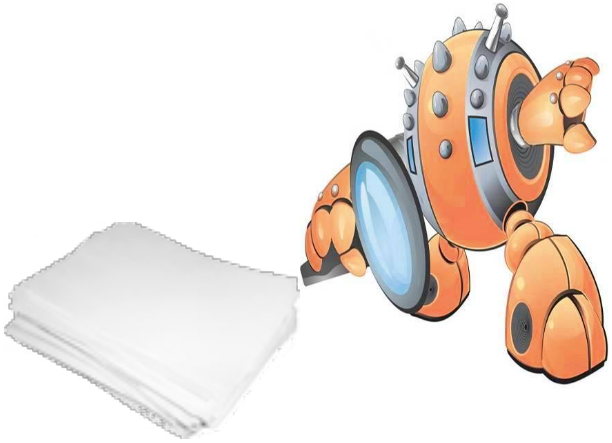
Weaknesses & Vulnerabilities

- Examine design, source code, or binary for weaknesses, adherence to guidelines, etc.

Comparing Static Analysis with Dynamic Analysis

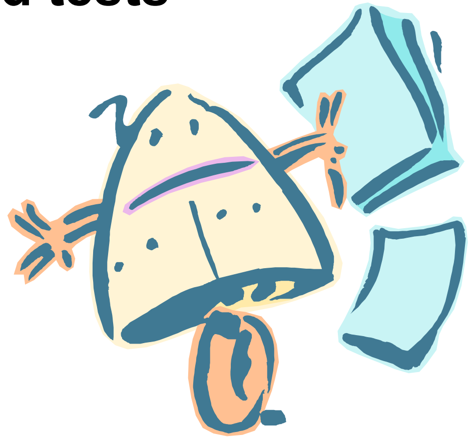
Static Analysis

- Code review
- Binary, byte, or source code scanners
- Model checkers & property proofs
- Assurance case



Dynamic Analysis

- Execute code
- Simulate design
- Fuzzing, coverage, MC/DC, use cases
- Penetration testing
- Field tests



Strengths of Static Analysis

- **Applies to many artifacts, not just code**
- **Independent of platform**
- **In theory, examines *all possible* executions, paths, states, etc.**
- **Can focus on a single specific property**

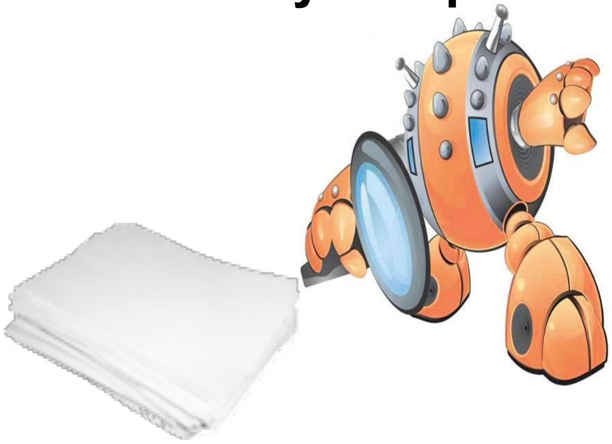
Strengths of Dynamic Analysis

- **No need for code**
- **Conceptually easier - “if you can run the system, you can run the test”.**
- **No (or less) need to build or validate models or make assumptions.**
- **Checks installation and operation, along with end-to-end or whole-system.**

Static and Dynamic Analysis Complement Each Other

Static Analysis

- Handles unfinished code
- Higher level artifacts
- Can find backdoors, e.g., full access for user name “JoshuaCaleb”
- Potentially complete



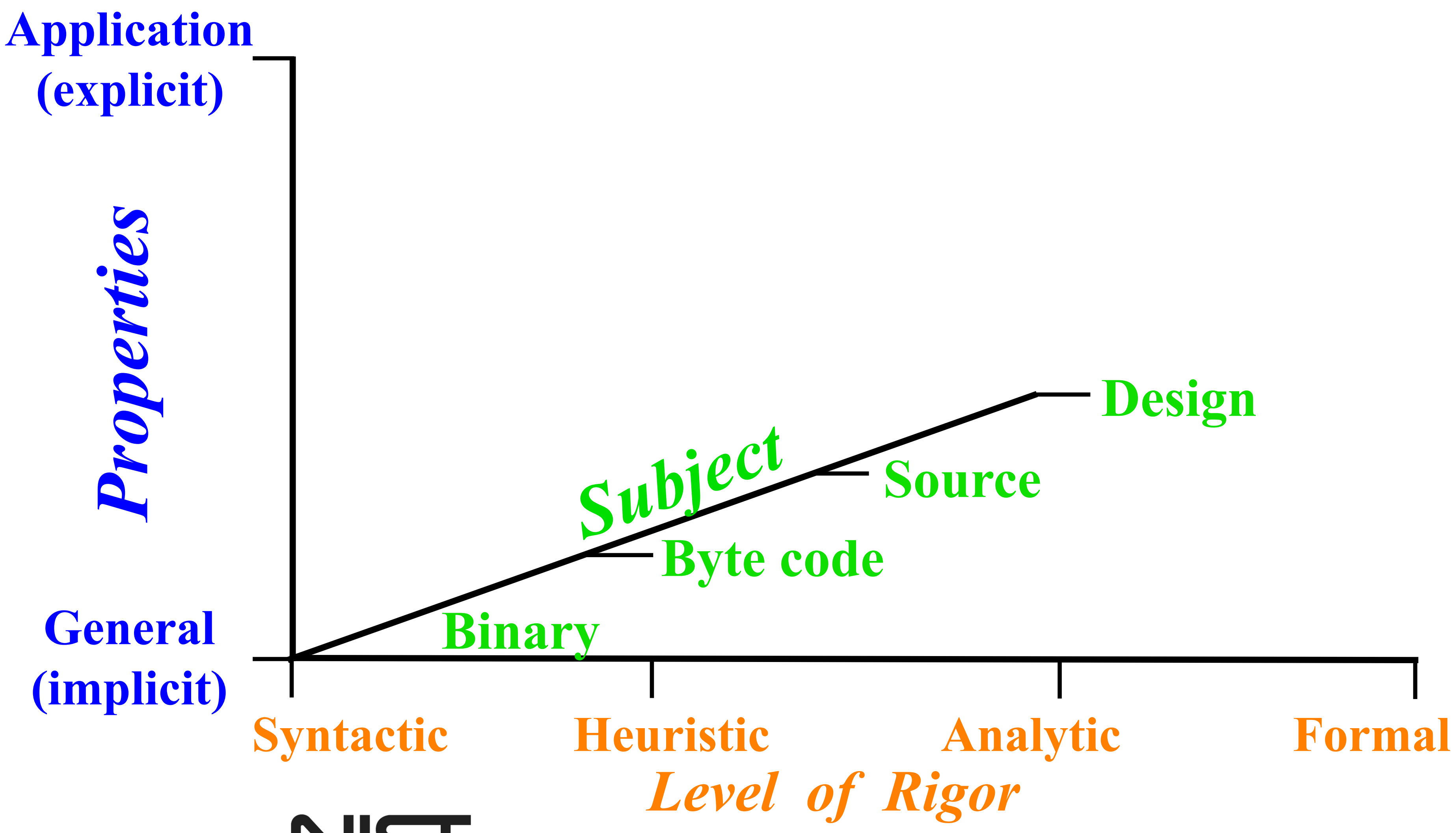
Dynamic Analysis

- Code not needed, e.g., embedded systems
- Has few(er) assumptions
- Covers end-to-end or system tests
- Assess as-installed

Different Static Analyzers Exist For Different Purposes

- **To check intellectual property violation**
- **By developers to decide what needs to be fixed (and learn better practices)**
- **By auditors or reviewer to decide if it is good enough for use**

Dimensions of Static Analysis



Dimension: Human Involvement

- **Range from completely manual**
  - **code reviews**
- **analyst aides and tools**
  - **call graphs**
  - **property prover**
- **human-aided analysis**
  - **annotations**
- **to completely automatic**
  - **scanners**

Dimension: Properties

- **Analysis can look for anything from general or universal properties:**
  - **don’t crash**
  - **don’t overflow buffers**
  - **filter inputs against a “white list”**
- **to application-specific properties:**
  - **log the date and source of every message**
  - **cleartext transmission**
  - **user cannot execute administrator functions**

Dimension: Subject

- **Design,**
- **Architecture,**
- **Requirements,**
- **Source code,**
- **Byte code, or**
- **Binary**

Dimension: Level of Rigor

- **Syntactic**
  - **flag every use of strcpy()**
- **Heuristic**
  - **every open() has a close(), every lock() has an unlock()**
- **Analytic**
  - **data flow, control flow, constraint propagation**
- **Fully formal**
  - **theorem proving**

Some Steps in Using a Tool

- **License per machine or once per site or pay per LoC**
- **Direct tool to code**
  - **List of files, “make” file, project, directory, etc.**
- **Compile**
- **Scan**
- **Analyze and review reports**

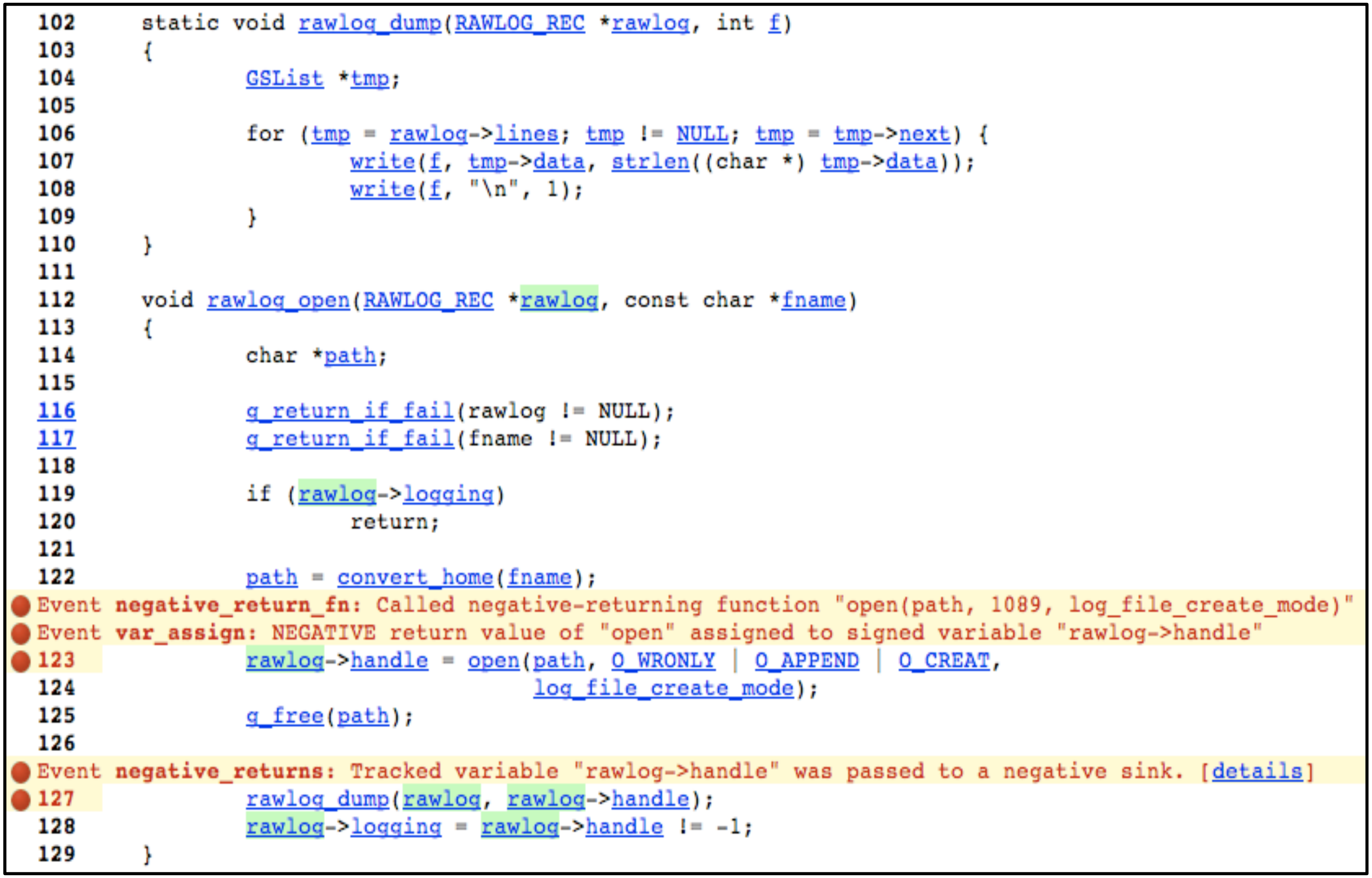
- ***May be simple:*** `flawfinder *.c`

Example tool output (1)

```
        char sys[512] = "/usr/bin/cat ";
25      gets(buff);
        strcat(sys, buff);
30      system(sys);
```

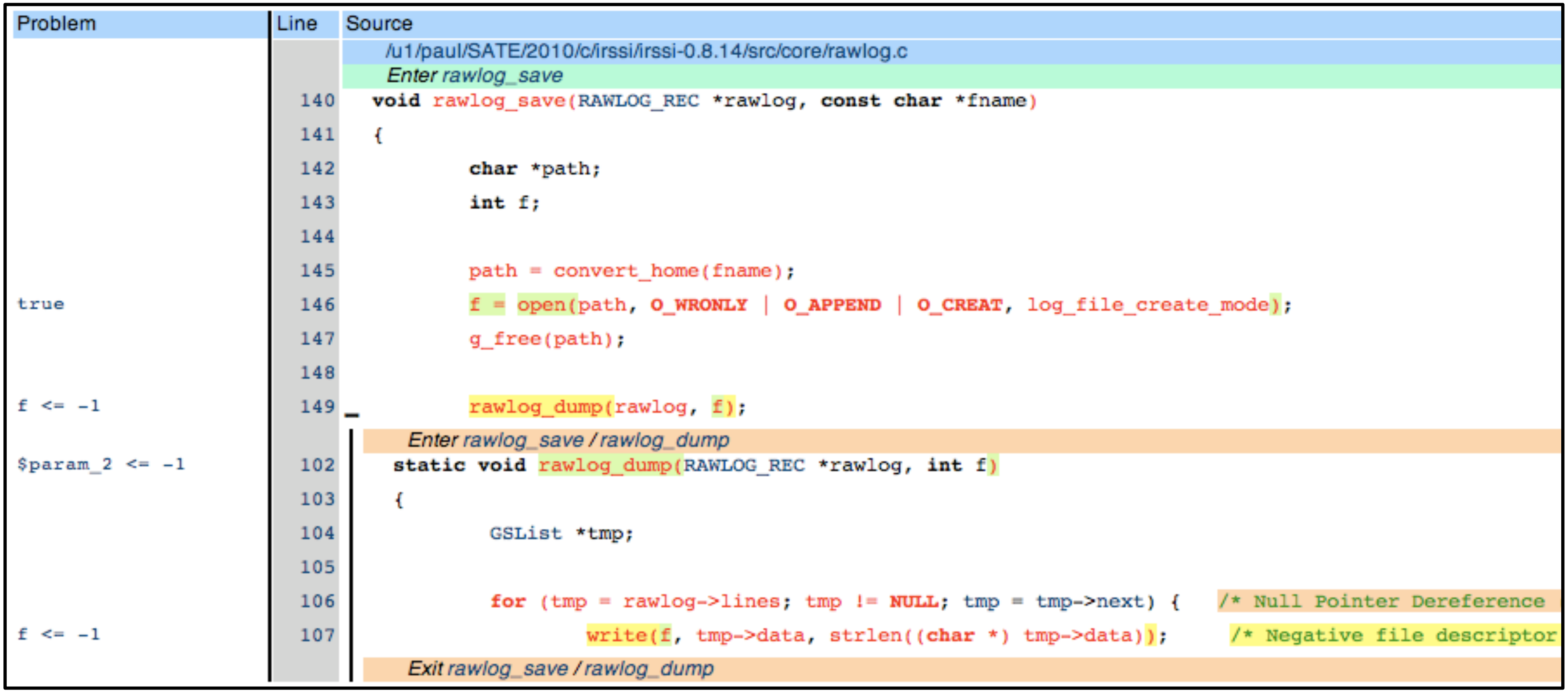
```
foo.c:30:Critical:Unvalidated string 'sys' is received from an
external function through a call to 'gets' at line 25. This can
be run as command line through call to 'system' at line 30. User
input can be used to cause arbitrary command execution on the
host system. Check strings for length and content when used for
command execution.
```

Example tool output (2)



```
static void rawlog_dump(RAWLOG_REC *rawlog, int f)
{
        GSList *tmp;

        for (tmp = rawlog->lines; tmp != NULL; tmp = tmp->next) {
                write(f, tmp->data, strlen((char *) tmp->data));
                write(f, "\n", 1);
        }
}

void rawlog_open(RAWLOG_REC *rawlog, const char *fname)
{
        char *path;

        g_return_if_fail(rawlog != NULL);
        g_return_if_fail(fname != NULL);

        if (rawlog->logging)
                return;

        path = convert_home(fname);
Event negative_return_fn: Called negative-returning function "open(path, 1089, log_file_create_mode)"
Event var_assign: NEGATIVE return value of "open" assigned to signed variable "rawlog->handle"
        rawlog->handle = open(path, O_WRONLY | O_APPEND | O_CREAT,
                              log_file_create_mode);
        g_free(path);

Event negative_returns: Tracked variable "rawlog->handle" was passed to a negative sink. [details]
        rawlog_dump(rawlog, rawlog->handle);
        rawlog->logging = rawlog->handle != -1;
}
```

Example tool output (3)



```
Problem             Line  Source
                          /u1/paul/SATE/2010/c/irssi/irssi-0.8.14/src/core/rawlog.c
                          Enter rawlog_save
                    140   void rawlog_save(RAWLOG_REC *rawlog, const char *fname)
                    141   {
                    142           char *path;
                    143           int f;
                    144
                    145           path = convert_home(fname);
true                146           f = open(path, O_WRONLY | O_APPEND | O_CREAT, log_file_create_mode);
                    147           g_free(path);
                    148
f <= -1             149           rawlog_dump(rawlog, f);
                            Enter rawlog_save / rawlog_dump
$param_2 <= -1      102     static void rawlog_dump(RAWLOG_REC *rawlog, int f)
                    103     {
                    104             GSList *tmp;
                    105
                    106             for (tmp = rawlog->lines; tmp != NULL; tmp = tmp->next) {   /* Null Pointer Dereference
f <= -1             107                     write(f, tmp->data, strlen((char *) tmp->data));     /* Negative file descriptor
                            Exit rawlog_save / rawlog_dump
```

Possible Data About Issues

- **Name, description, examples, remedies**
- **Severity, confidence, priority**
- **Source, sink, control flow, conditions**

Tools Help User Manage Issues

- View issues by
  - Category
  - File
  - Package
  - Source or sink
  - New since last scan
  - Priority
- User may write custom rules

May Integrate With Other Tools

- **Eclipse, Visual Studio, etc.**
- **Penetration testing**
- **Execution monitoring**
- **Bug tracking**

Outline

- The Software Assurance Metrics And Tool Evaluation (SAMATE) project
- What is static analysis?
- **Limits of automatic tools**
- State of the art in static analysis tools
- Static analyzers in the software development life cycle

Overview of Static Analysis Tool Exposition (SATE)

- **Goals:**
  - **Enable empirical research based on large test sets**
  - **Encourage improvement of tools**
  - **Speed adoption of tools by objectively demonstrating their use on real software**
- ***NOT* to choose the "best" tool**
- **Events**
  - **We chose C & Java programs with security implications**
  - **Participants ran tools and returned reports**
  - **We analyzed reports**
  - **Everyone shared observations at a workshop**
  - **Released final report and all data later**
- **http://samate.nist.gov/SATE.html**
- **Co-funded by NIST and DHS, Nat'l Cyber Security Division**

SATE Participants

- **2008:**
  - **Aspect Security ASC**
  - **Checkmarx CxSuite**
  - **Flawfinder**
  - **Fortify SCA**
  - **Grammatech CodeSonar**
  - **HP DevInspect**
  - **SofCheck Inspector for Java**
  - **UMD FindBugs**
  - **Veracode SecurityReview**
- **2009:**
  - **Armorize CodeSecure**
  - **Checkmarx CxSuite**
  - **Coverity Prevent**
  - **Grammatech CodeSonar**
  - **Klocwork Insight**
  - **LDRA Testbed**
  - **SofCheck Inspector for Java**
  - **Veracode SecurityReview**

SATE 2010 tentative timeline

- ✓ **Hold organizing workshop (12 Mar 2010)**
- ✓ **Recruit planning committee.**
- ● **Revise protocol.**
- ● **Choose test sets. Provide them to participants (17 May)**
- ● **Participants run their tools. Return reports (25 June)**
- ● **Analyze tool reports (27 Aug)**
- ● **Share results at workshop (October)**
- ● **Publish data (after Jan 2011)**

Do We Catch All Weaknesses?

- **To answer, we must list "all weaknesses."**
- **Common Weakness Enumeration (CWE) is an effort to list and organize them.**
- **Lists almost 700 CWEs**

http://cwe.mitre.org/

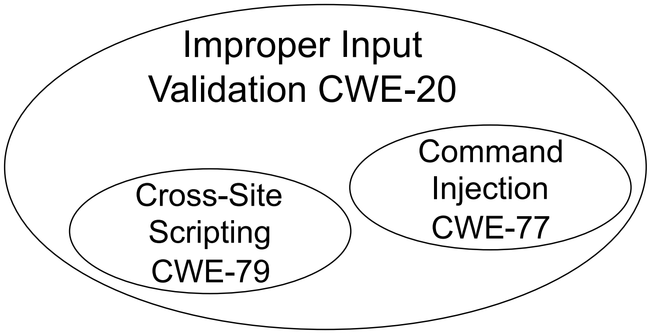
“One Weakness” is an illusion

- **Only 1/8 to 1/3 of weaknesses are simple.**
- **The notion breaks down when**
  - **weakness classes are related and**
  - **data or control flows are intermingled.**

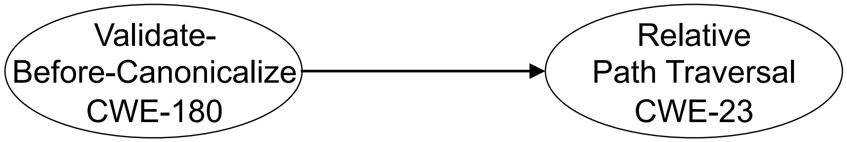
- **Even “location” is nebulous.**

How Weakness Classes Relate

- **Hierarchy**

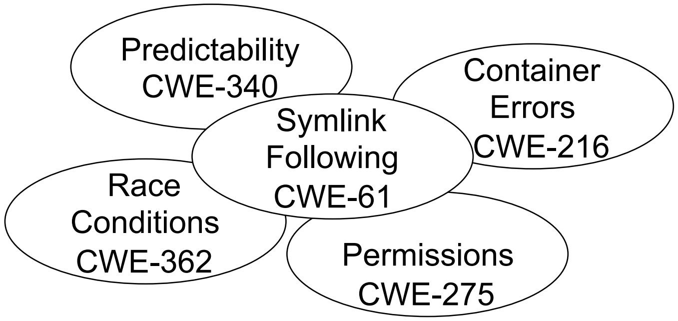


- **Chains**



```
lang = %2e./%2e./%2e./etc/passwd%00
```

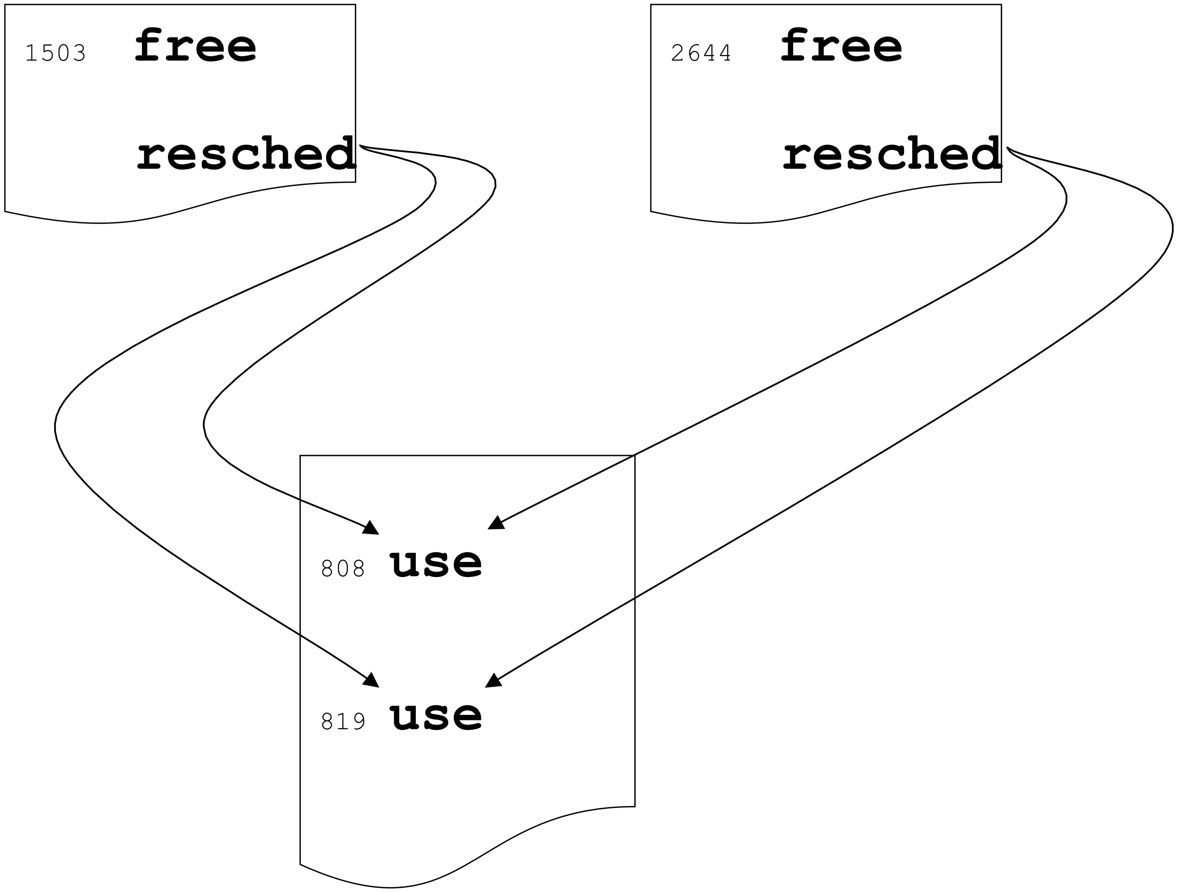
- **Composites**



- *from "Chains and Composites", Steve Christey, MITRE*
  http://cwe.mitre.org/data/reports/chains_and_composites.html

“Number of bugs” is ill-defined

Tangled Flow: 2 sources, 2 sinks, 4 paths



Many weaknesses are ill-defined

- **CWE-121 Stack-based Buffer Overflow**

  Description Summary:

  - **A stack-based buffer overflow condition is a condition where the buffer being overwritten is allocated on the stack (i.e., is a local variable or, rarely, a parameter to a function).**

  White Box Definition:

  - **A buffer overflow where the buffer from the Buffer Write Operation is statically allocated.**

From CWE version 1.3

Is this an instance of CWE-121?

```
char *buf;
int main(int argc, char **argv) {
    buf = (char *)alloca(256);
    strcpy(buf, argv[1]);
}
```

  - "… the buffer being overwritten is allocated on the stack (i.e., is a local variable or, rarely, a parameter to a function)."

- Strictly, no, because buf is a global variable.

Is this an instance of CWE-121?

```
char *buf;
int main(int argc, char **argv) {
    buf = (char *)alloca(256);
    strcpy(buf, argv[1]);
}
```

  - "… the buffer from the Buffer Write Operation is statically allocated"
- Again, strictly, no: buf dynamically allocated

We need more precise, accurate definitions of weaknesses.

- **One definition won’t satisfy all needs.**
- **“Precise” suggests formal.**
- **“Accurate” suggests (most) people agree.**
- **Probably not worthwhile for all 700 CWEs.**

Example: theoretical integer overflow, from SRD case 2083

```
int main(int argc, char **argv) {
  char buf[MAXSIZE];

  . . . put a string in buf

  if (strlen(buf) + strlen(argv[2]) < MAXSIZE) {
    strcat(buf, argv[2]);
  }

    . . . do something with buf
}
```

Example: language standard vs. convention, from SRD case 201

```
typedef struct {
  int int_field;
  char buf[10];
} my_struct;

int main(int argc, char **argv){
  my_struct s;

  s.buf[10] = 'A';

  return 0;
}
```

Outline

- The Software Assurance Metrics And Tool Evaluation (SAMATE) project
- What is static analysis?
- Limits of automatic tools
- **State of the art in static analysis tools**
- Static analyzers in the software development life cycle

General Observations

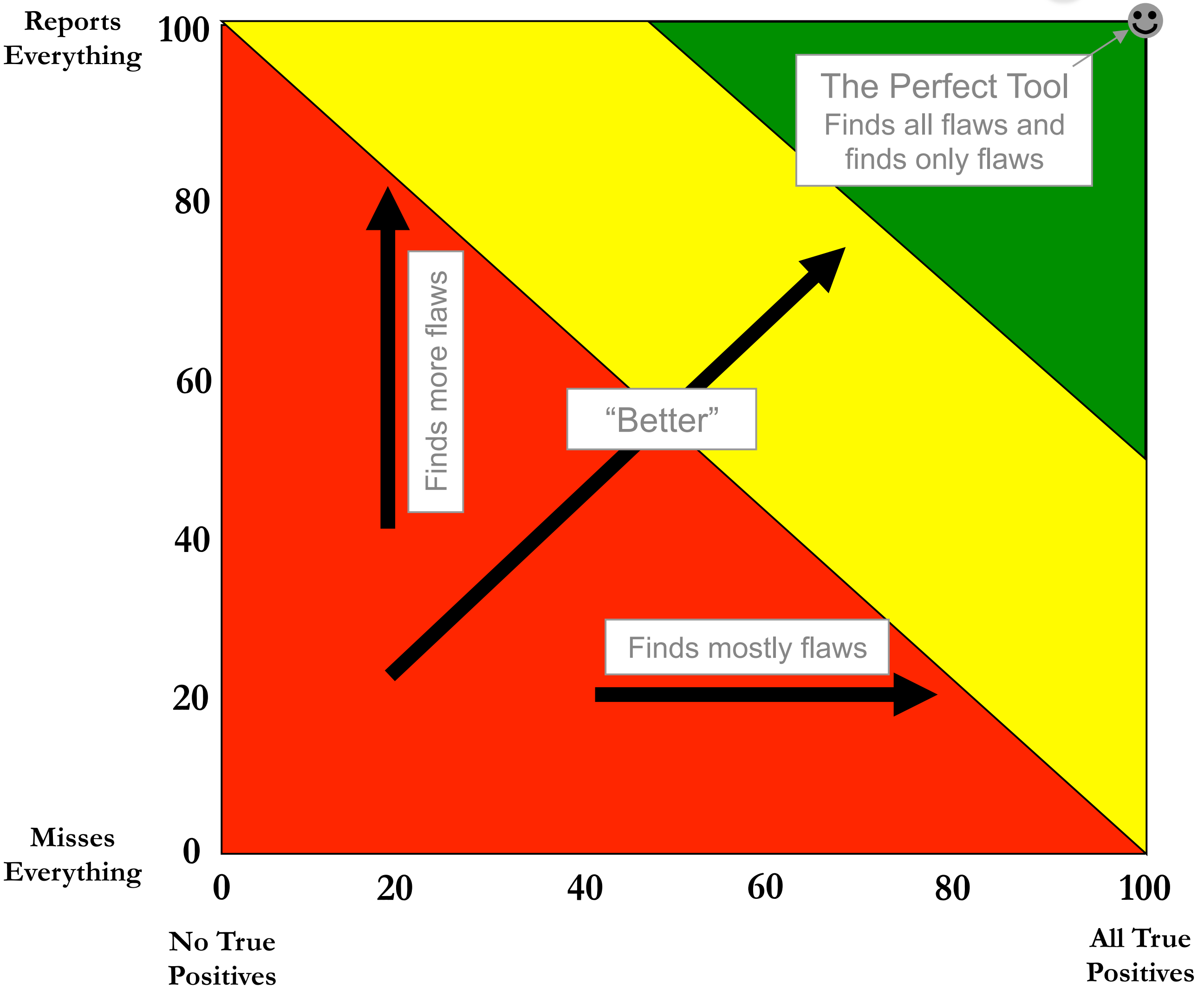
- **Tools can't catch everything: unimplemented features, design flaws, improper access control, …**
- **Tools catch real problems: XSS, buffer overflow, cross-site request forgery**
  - **13 of SANS Top 25 (21 counting related CWEs)**
- **Tools are even more helpful when tuned**

Tools Useful in Quality "Plains"

Tararua mountains and the Horowhenua region, New Zealand
Swazi Apparel Limited www.swazi.co.nz used with permission

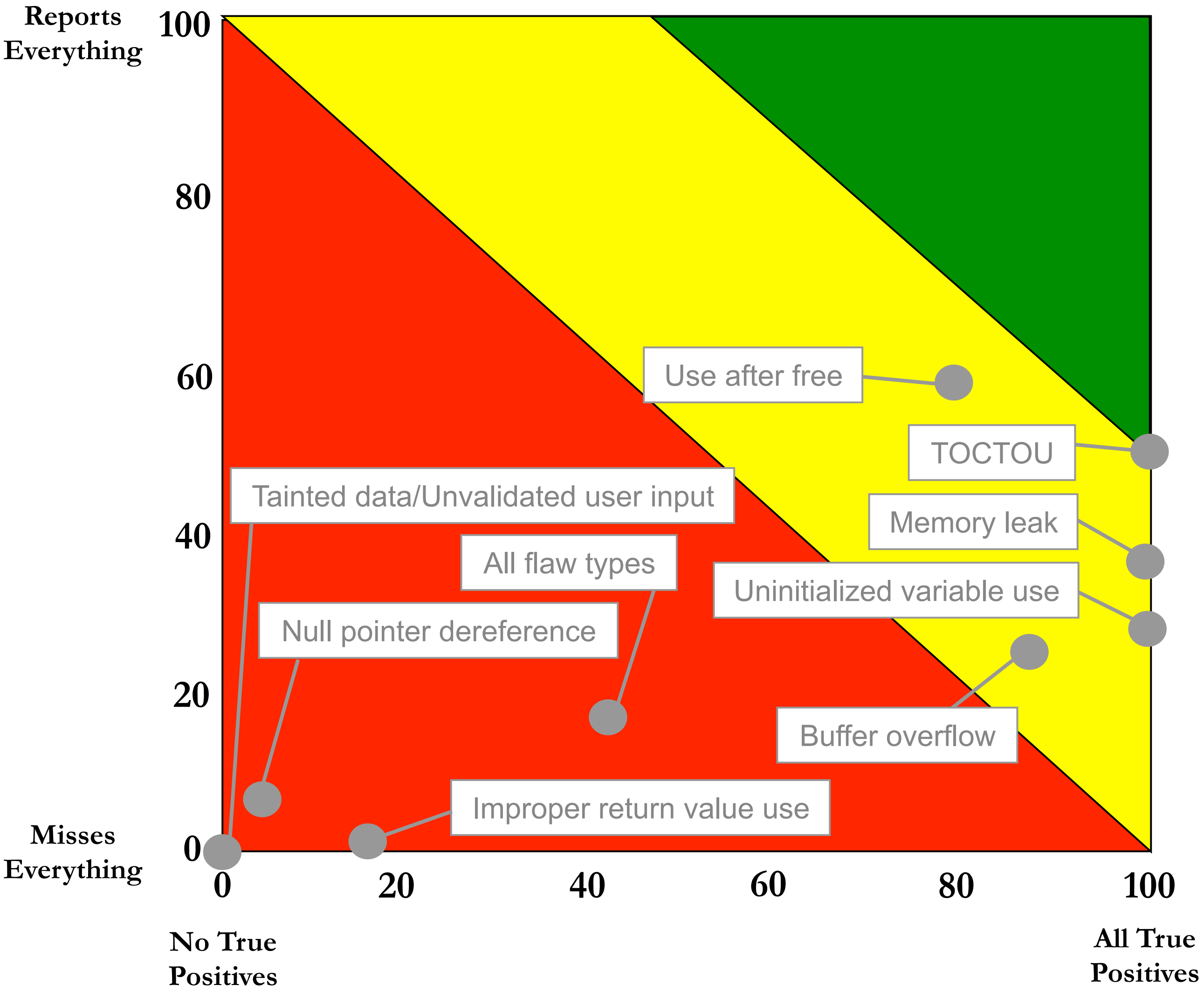
- **Tools alone are not enough to achieve the highest "peaks" of quality.**
- **In the "plains" of typical quality, tools can help.**
- **If code is adrift in a "sea" of chaos, train developers.**

Precision & Recall Scoring



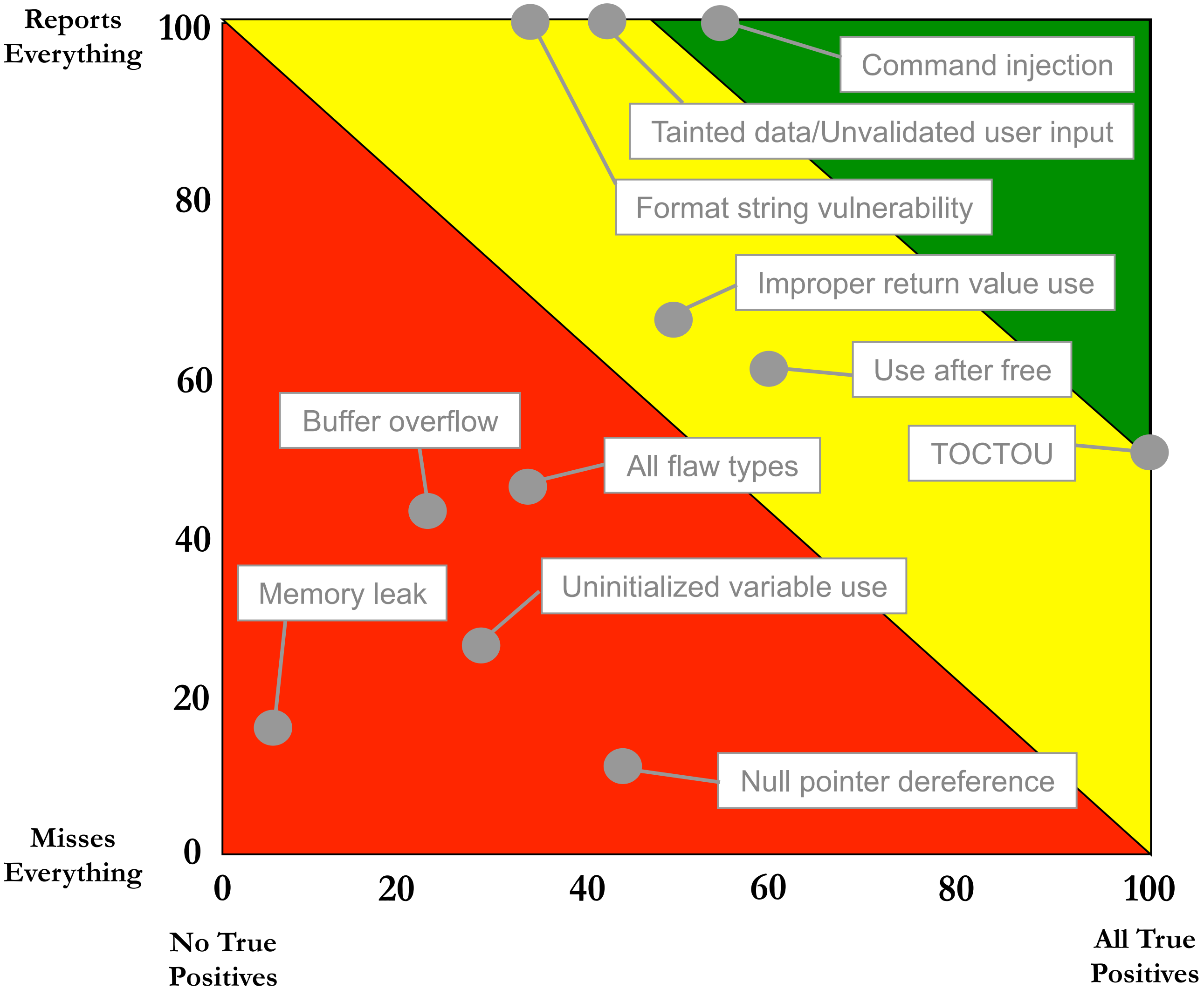
from DoD 2004

Tool A



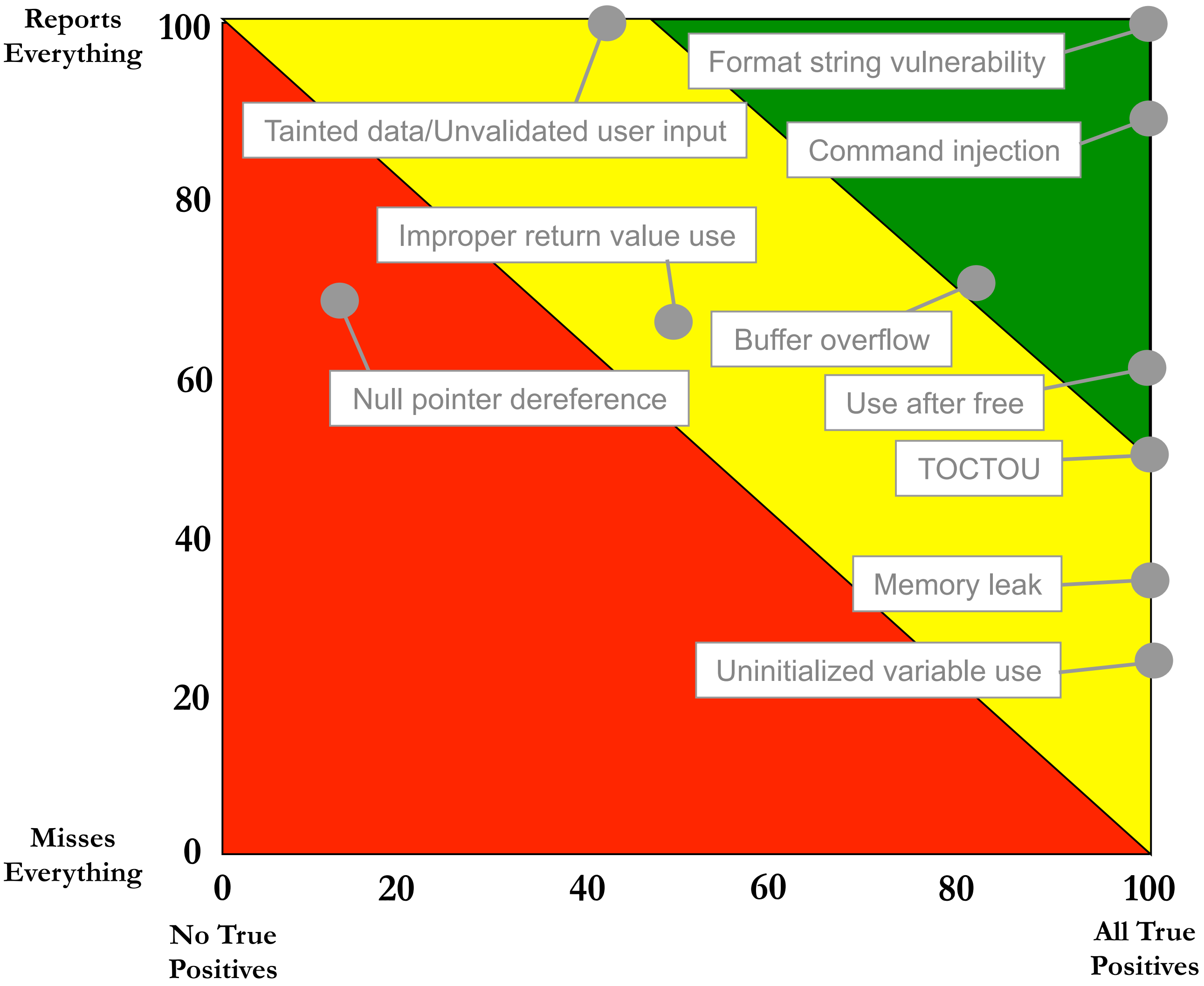
from DoD 2004

Tool B



from DoD 2004

Best of each Tool



from DoD 2004

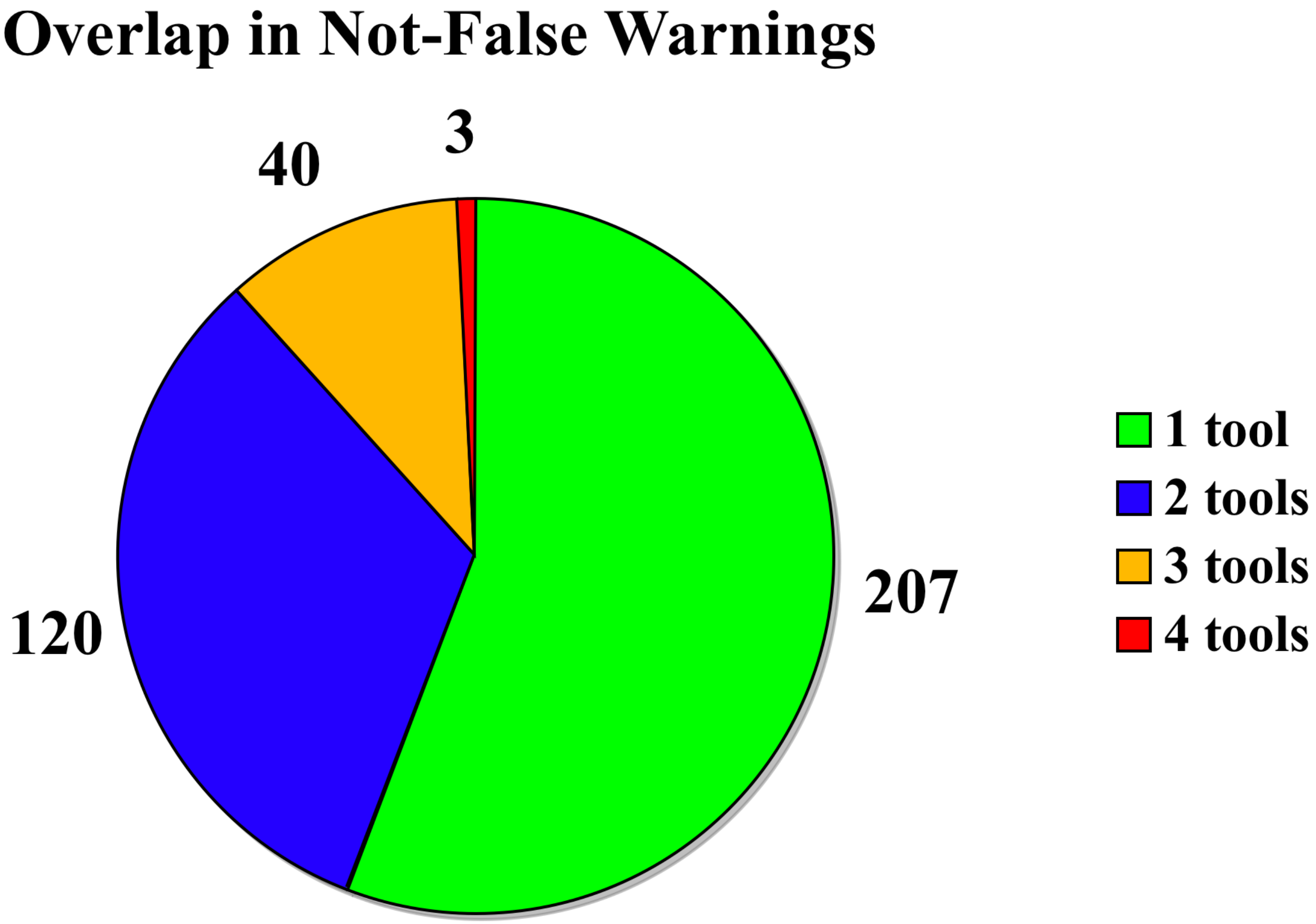
Use a Better Language

Summary of SATE 2009 reports

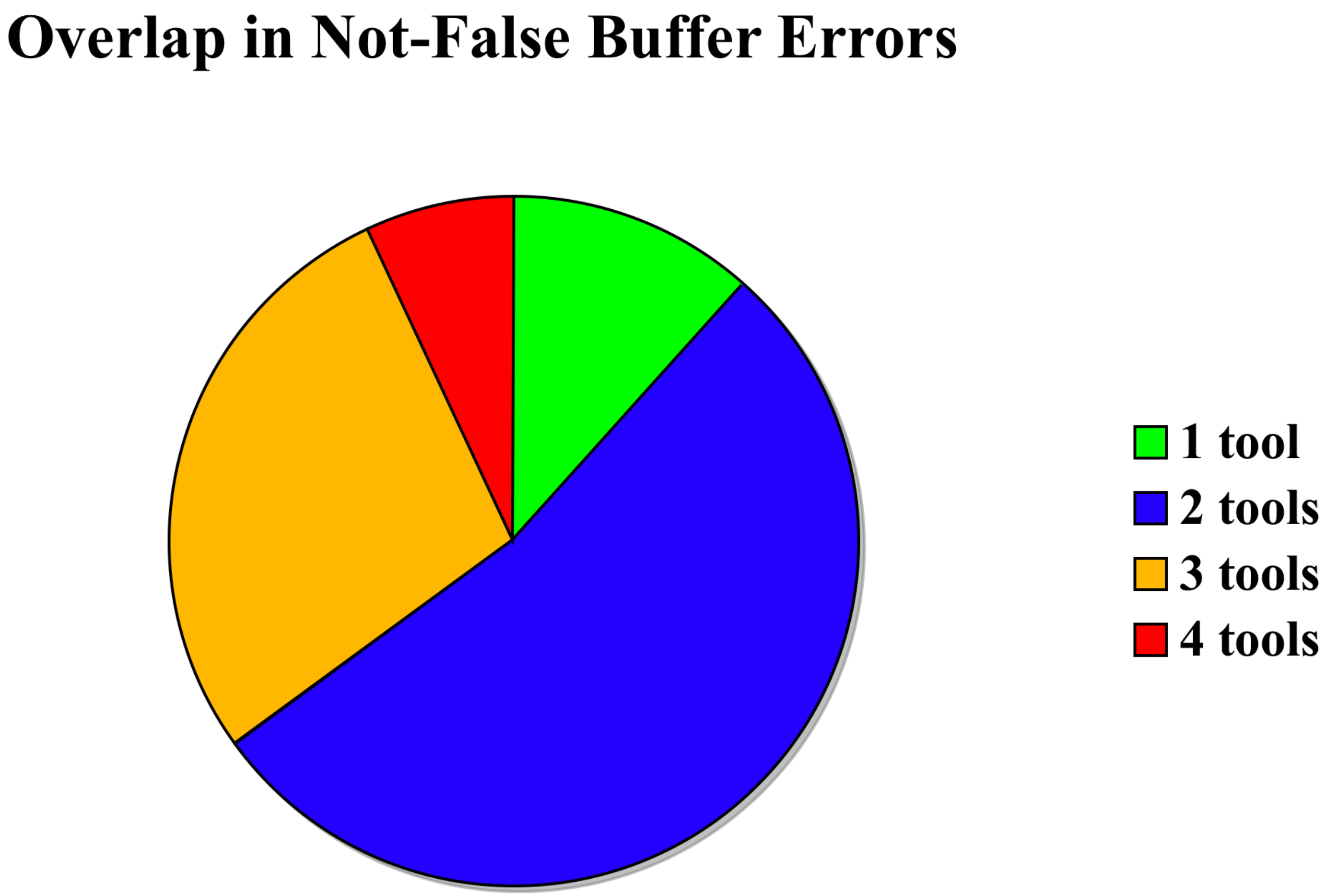
- **Reports from 18 tool runs**
  - **4 or 5 tools on each program**
- **About 20,000 total warnings**
  - **but tools prioritize by severity, likelihood**
- **Reviewed 521 warnings - 370 were not false**

- **Number of warnings varies a lot by tool and case**
- **83 CWE ids/221 weakness names**

Tools don't report same warnings

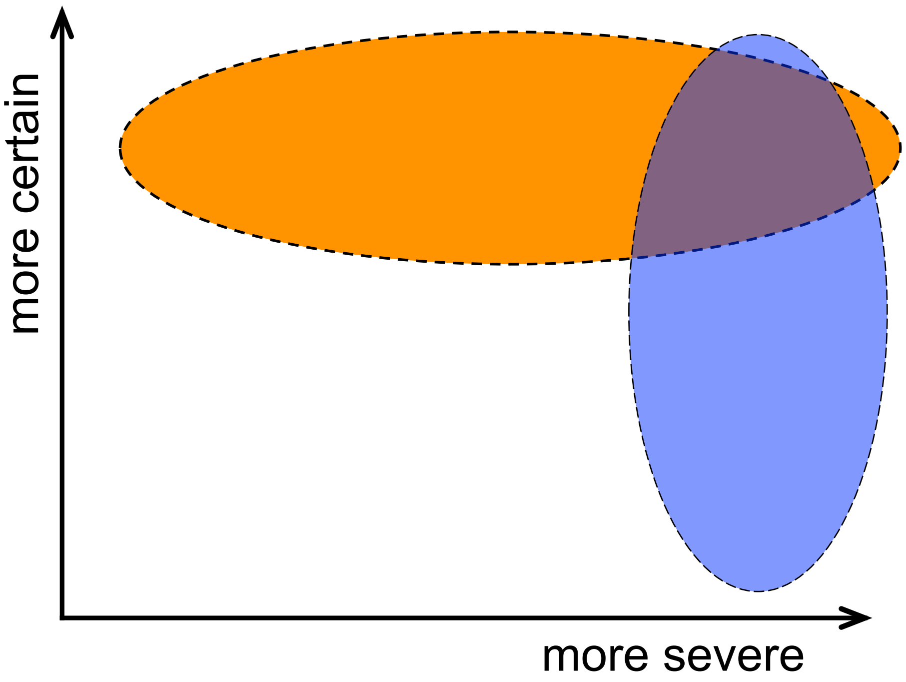


Some types have more overlap

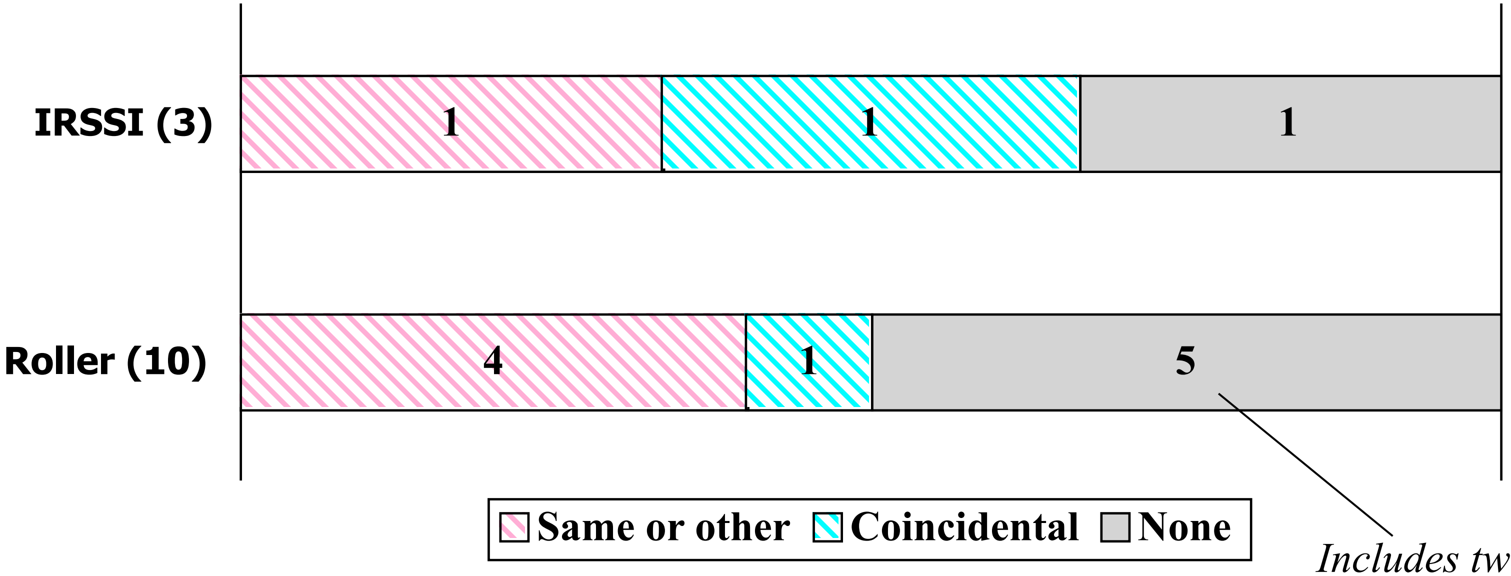


Why don’t tools find same things?

- **Tools look for different weakness classes**
- **Tools are optimized differently**



Tools find some things people find



Includes two access control issues – very hard for tools

Tools Complement Humans

- **Example from DCC Chat**

```
/* generate a random id */
p_id = rand() % 64;
dcc->pasv_id = p_id;
              .
              .
              .
    if (dcc->pasv_id != atoi(params[3]))
        /* IDs don't match! */
        dcc_destroy(DCC(dcc));
```

Humans Complement Tools

- **Example from Network**

```
    /* if there are multiple addresses, return random one */
    use_v4 = count_v4 <= 1 ? 0 : rand() % count_v4;
    use_v6 = count_v6 <= 1 ? 0 : rand() % count_v6;
```

Outline

- The Software Assurance Metrics And Tool Evaluation (SAMATE) project
- What is static analysis?
- Limits of automatic tools
- State of the art in static analysis tools
- **Static analyzers in the software development life cycle**

Assurance from three sources

$$A = f(p, s, e)$$

where *A* is functional assurance, *p* is process quality, *s* is assessed quality of software, and *e* is execution resilience.

p is process quality

$$A = f(p, s, e)$$

- **High assurance software must be developed with care, for instance:**
  - **Validated requirements**
  - **Good system architecture**
  - **Security designed- and built in**
  - **Trained programmers**
  - **Helpful programming language**

s is assessed quality of software

$$A = f(p, s, e)$$

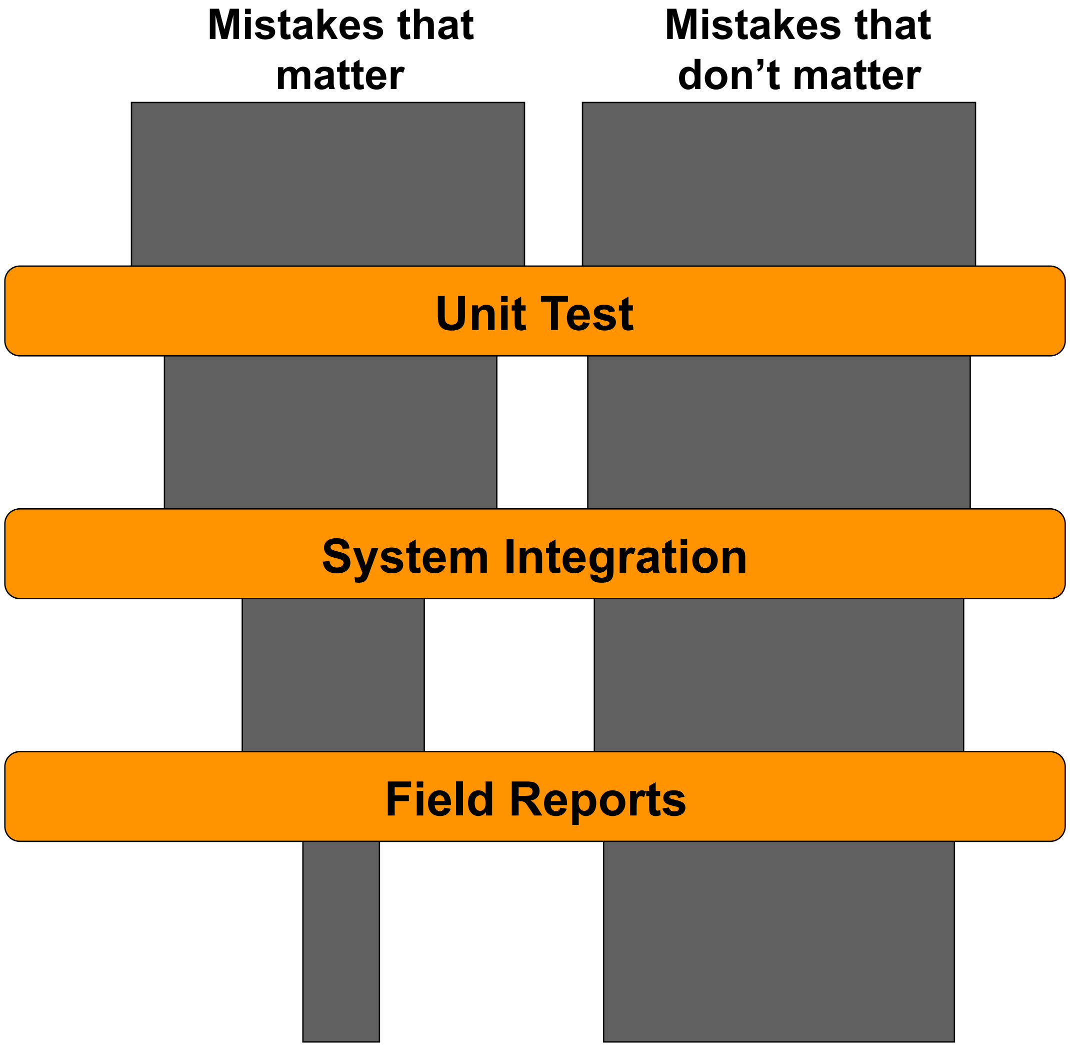
- **Two general kinds of software assessment:**
  - **Static analysis**
    - **e.g. code reviews and scanner tools**
    - **examines code**
  - **Testing (dynamic analysis)**
    - **e.g. penetration testing, fuzzing, and red teams**
    - **runs code**

e is execution resilience

$$A = f(p, s, e)$$

- **The execution platform can add assurance that the system will function as intended.**
- **Some techniques are:**
  - **Randomize memory allocation**
  - **Execute in a “sandbox” or virtual machine**
  - **Monitor execution and react to intrusions**
  - **Replicate processes and vote on output**

Survivor effect in software



Mistakes that matter

Mistakes that don’t matter

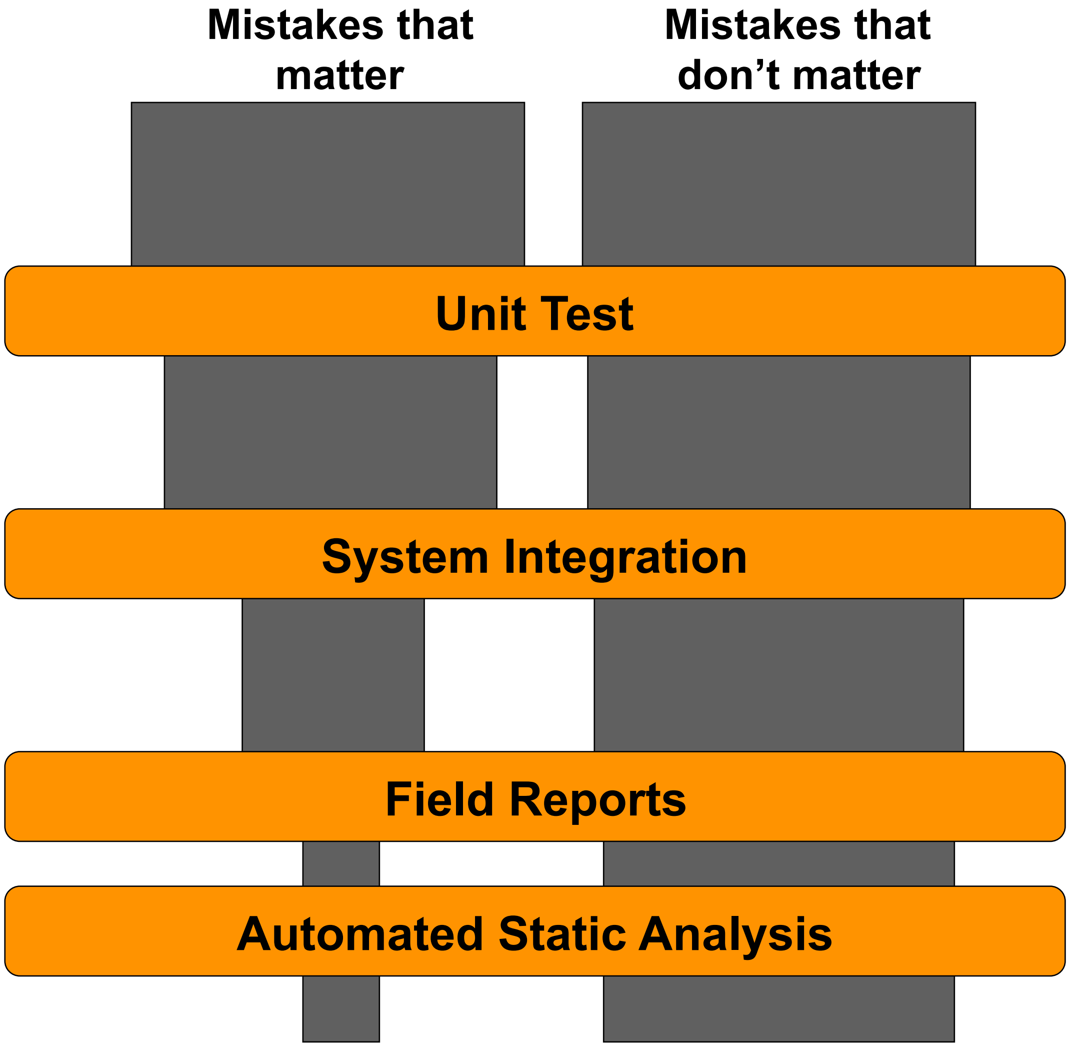
Unit Test

System Integration

Field Reports

after Bill Pugh
SATE workshop
Nov 2009

Late automated analysis is hard



Mistakes that matter

Mistakes that don’t matter

Unit Test

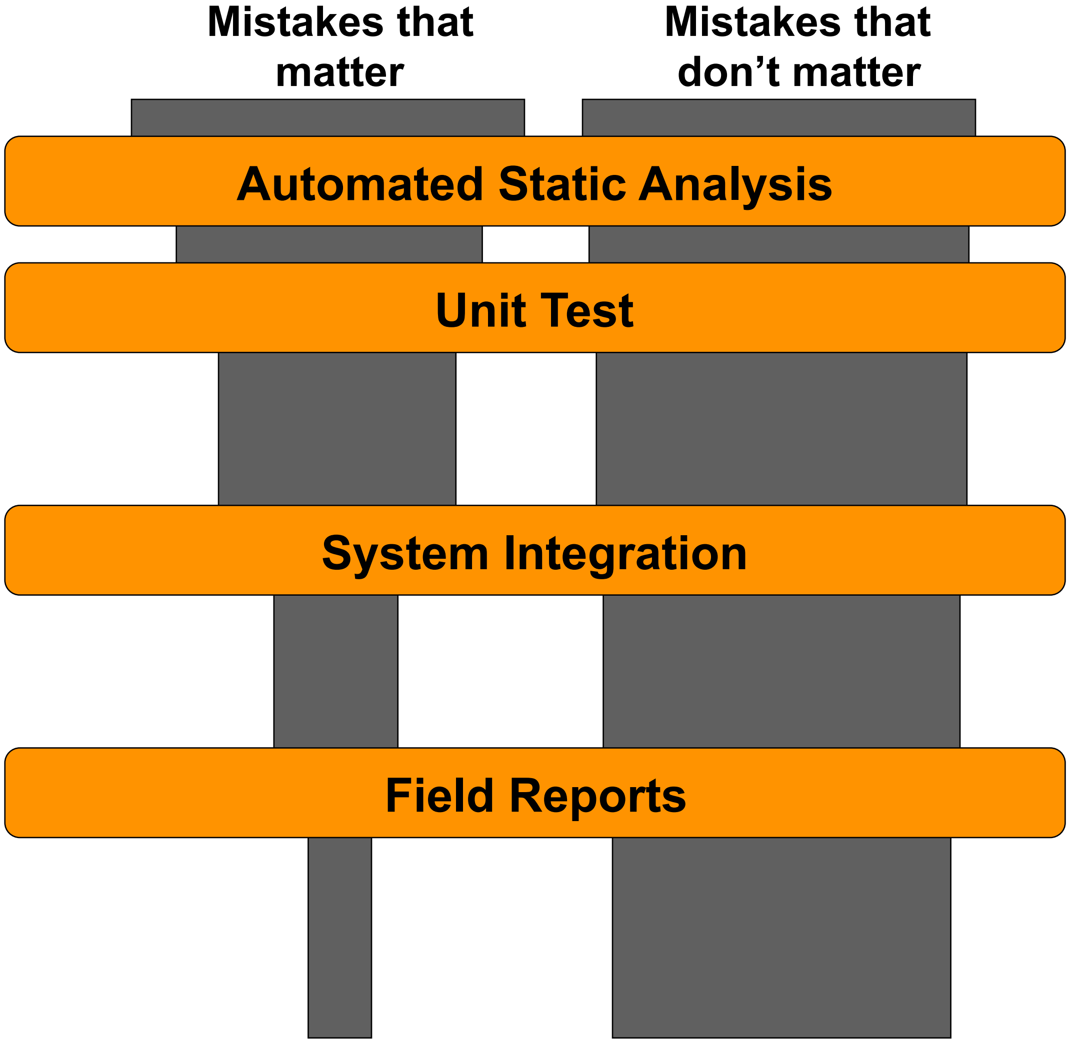
System Integration

Field Reports

Automated Static Analysis

after Bill Pugh
SATE workshop
Nov 2009

Automated analysis best at start



Mistakes that matter

Mistakes that don't matter

Automated Static Analysis

Unit Test

System Integration

Field Reports

after Bill Pugh
SATE workshop
Nov 2009

When is survivor effect weak?

- **If testing or deployment isn’t good at detecting problems**
  - **True for many security and concurrency problems**
- **If faults don’t generate clear failures**
  - **Also true for many security problems**

after Bill Pugh
SATE workshop
Nov 2009

Analysis is like a seatbelt …