

Evaluating Bug Finders

Test and Measurement of Static Code Analyzers

Aurelien DELAITRE
Bertrand STIVALET



<http://samate.nist.gov>

Authors

Aurelien DELAITRE

West Virginia University

aure@nist.gov



Bertrand STIVALET

National Institute of Standards
and Technology

stivalet@nist.gov

The logo of the National Institute of Standards and Technology (NIST), featuring the word 'NIST' in a bold, black, sans-serif font, enclosed within a dashed circular border. The logo is positioned to the left of Bertrand Stivalet's name and affiliation.

NIST

Authors

Elizabeth FONG

NIST

efong@nist.gov

NIST

NIST

Vadim OKUN

NIST

vadim.okun@nist.gov



“

*"If **debugging** is the process of removing software bugs, then **programming** must be the process of putting them in"*

E. Dijkstra

1.

SAMATE Project

Software Assurance Metrics And Tool
Evaluation



Software Assurance Reference Dataset (SARD)

◎ SARD contains

- Small test cases w/ specific vulnerabilities
- Large test suites
- Software w/ CVEs

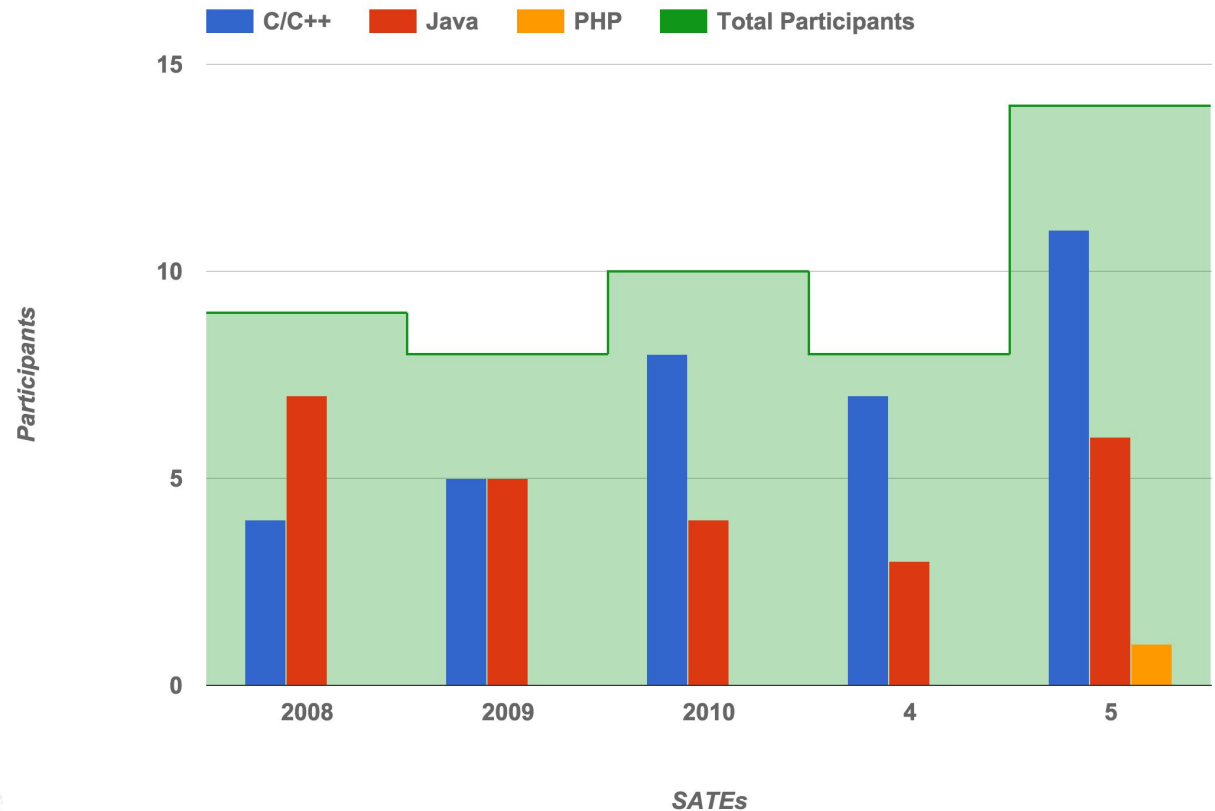
◎ SARD in numbers

- **34** Test suites
- **243** CWEs
- **148,903** Test cases
- **665,481** Files

<http://samate.nist.gov/SARD>

Static Analysis Tool Expositions (SATE)

- ◎ **5** editions of SATE
- ◎ **3** programming languages
- ◎ **5M+** lines of code for SATE V

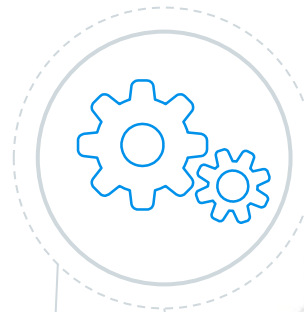


A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines, with some nodes highlighted in blue.

2.

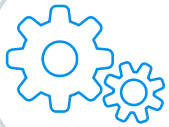
Software as Big Data

Introduction to Static Analysis



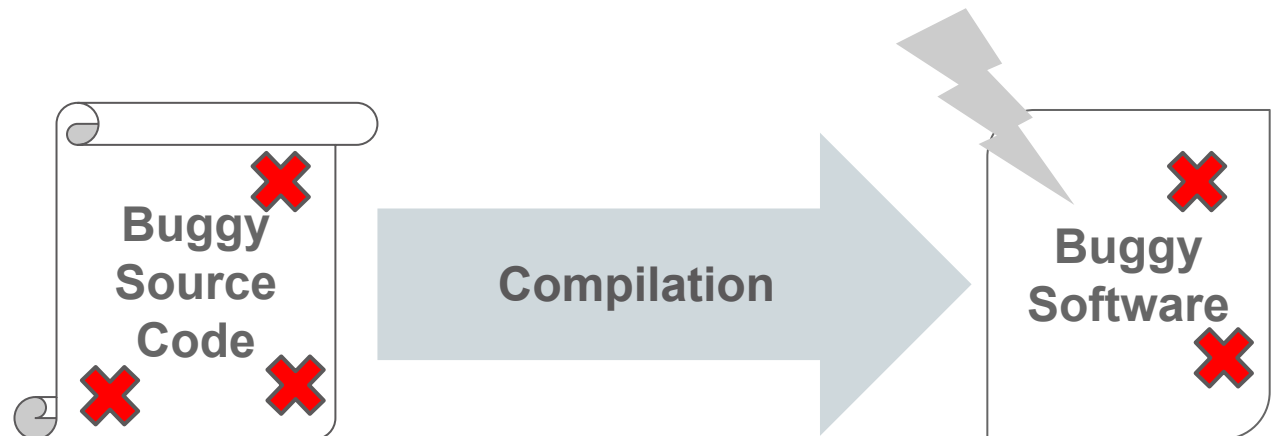
Static Analysis

- ◎ Automated analysis of large software
- ◎ Defect detection and remediation
- ◎ Use different approaches:
 - Syntax checking
 - Heuristics
 - Formal methods



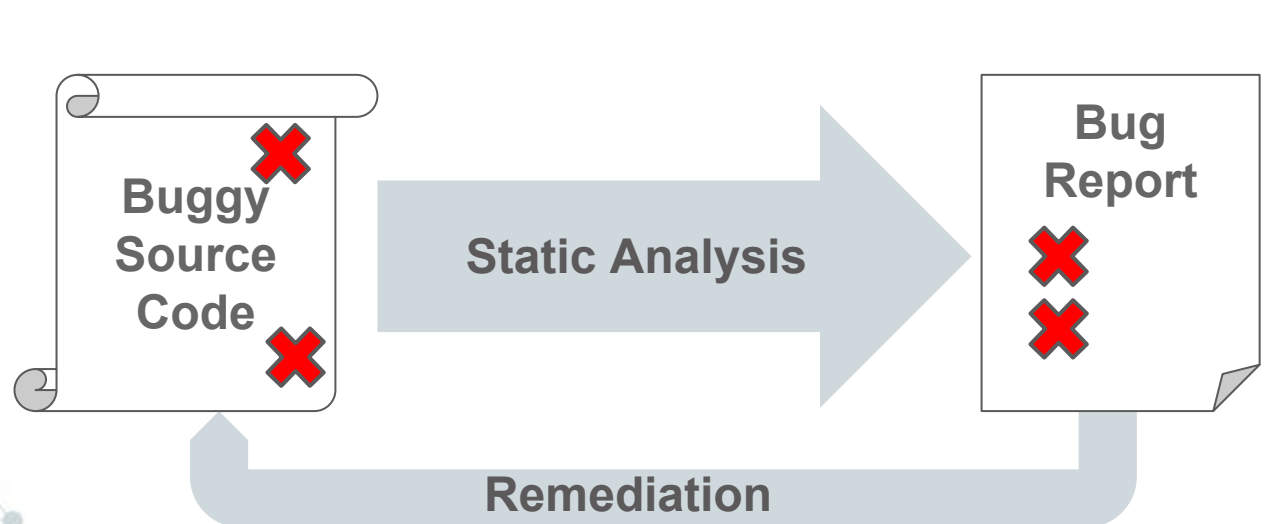
Static Analysis

- ⦿ Automated analysis of large software
- ⦿ Defect detection and remediation
- ⦿ Use different approaches



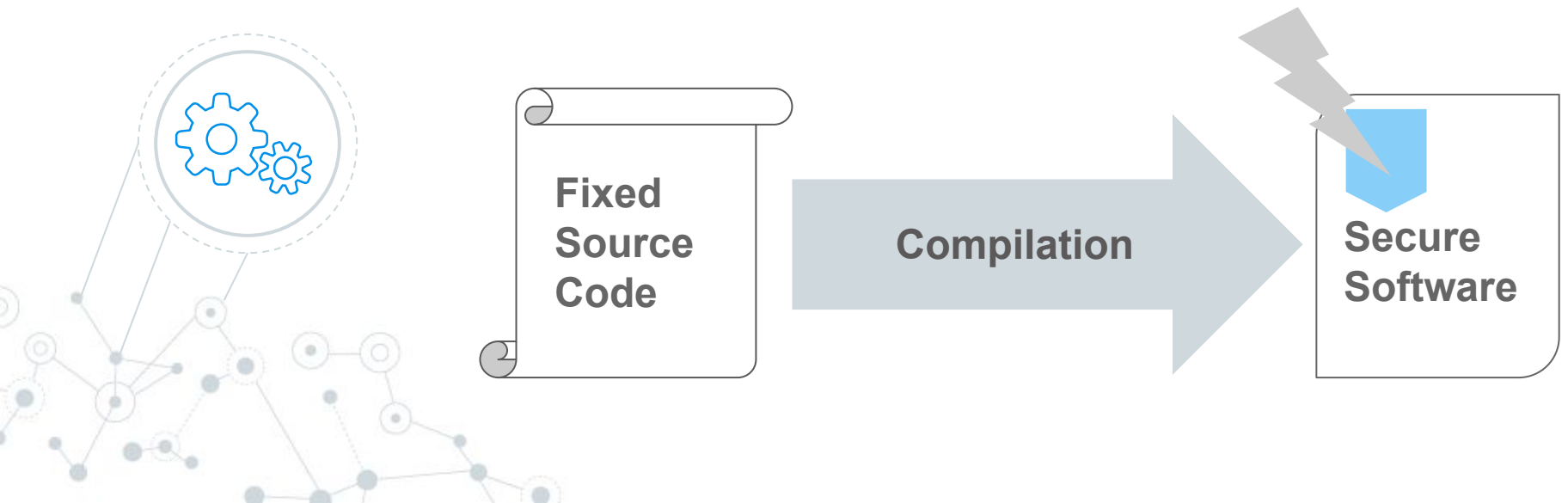
Static Analysis

- ⦿ Automated analysis of large software
- ⦿ Defect detection and remediation
- ⦿ Use different approaches



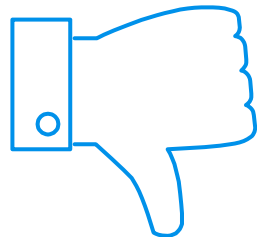
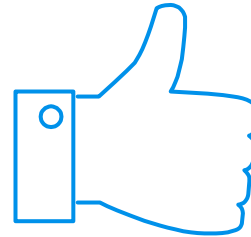
Static Analysis

- ⦿ Automated analysis of large software
- ⦿ Defect detection and remediation
- ⦿ Use different approaches



Pros and Cons

- ⦿ Improves software assurance
- ⦿ Saves time and money
- ⦿ Takes customized rule sets



- ⦿ False positive (noise)
- ⦿ False negative (missed defects)
- ⦿ Limited scope

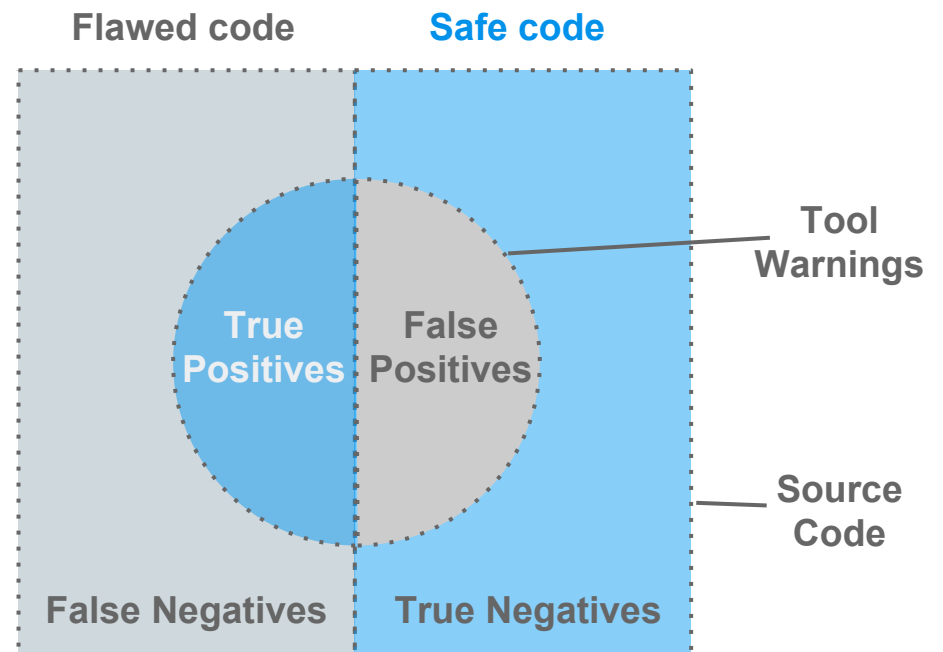
A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines, with some nodes highlighted in blue and others in grey.

3. **Metrics**

Measuring the Effectiveness of Tools

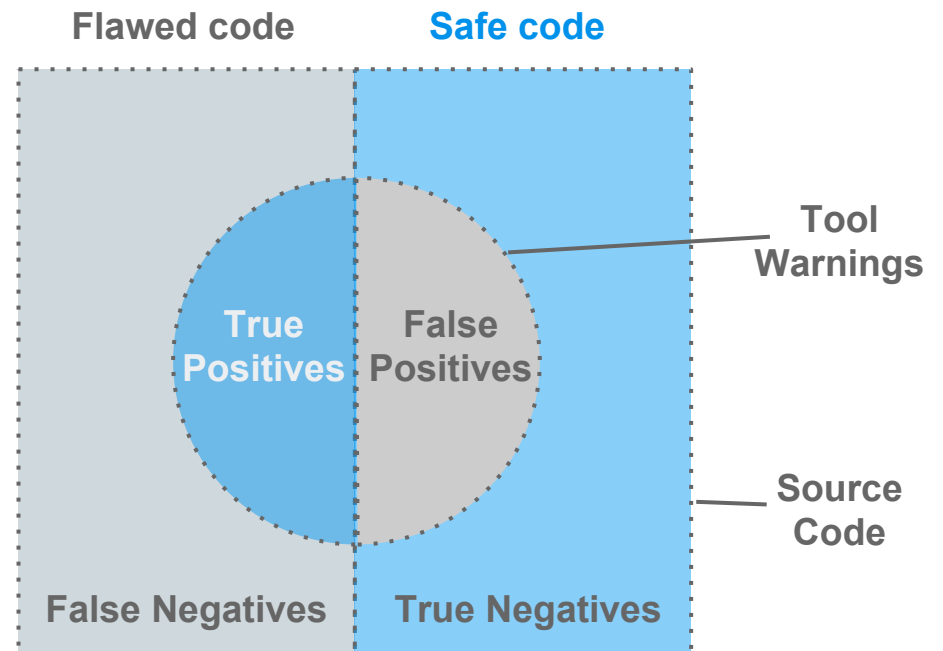


Evaluation Metrics



Evaluation Metrics

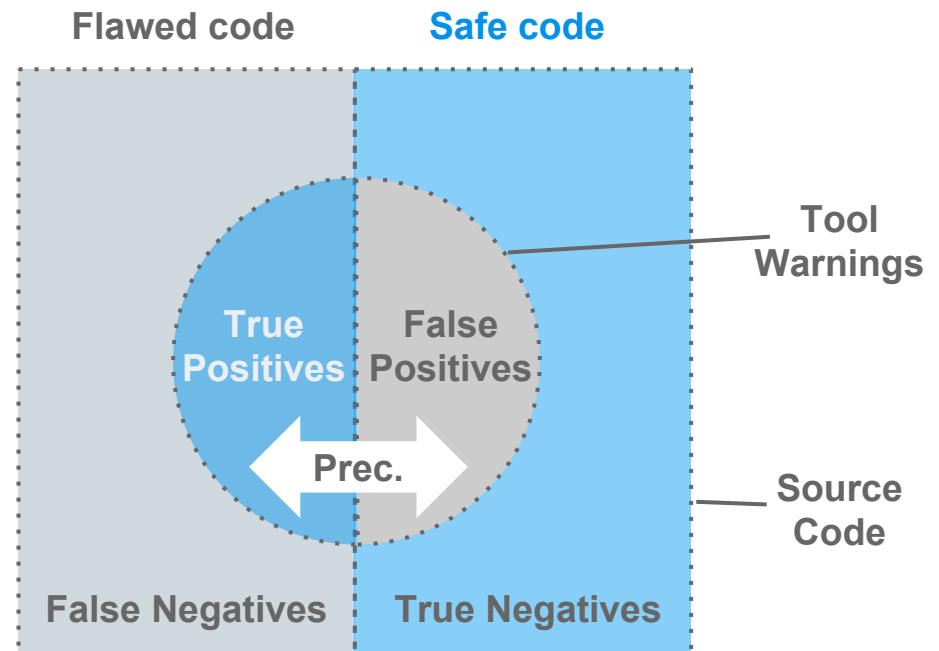
How much can I trust a tool ?



Evaluation Metrics

Precision

How much can I trust a tool ?

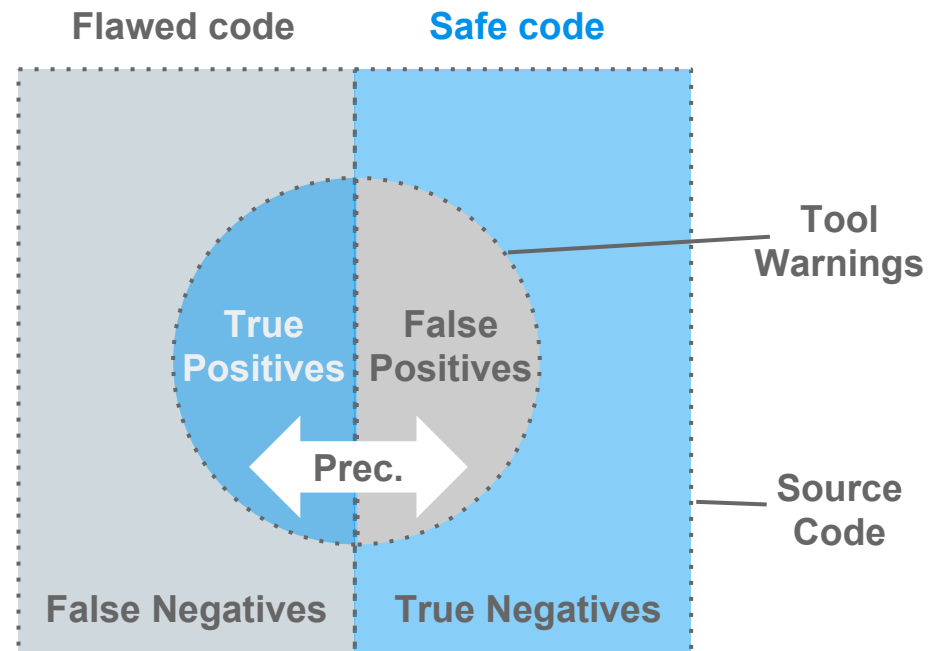


Evaluation Metrics

Precision

How much can I trust a tool ?

What proportion of flaws can a tool find ?



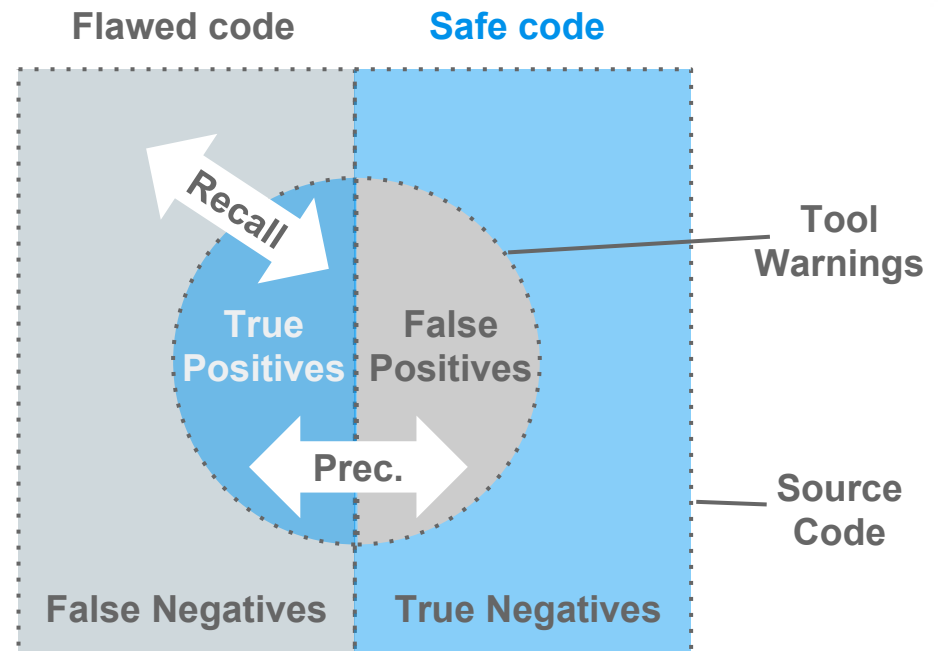
Evaluation Metrics

Precision

How much can I trust a tool ?

Recall

What proportion of flaws can a tool find ?



Evaluation Metrics

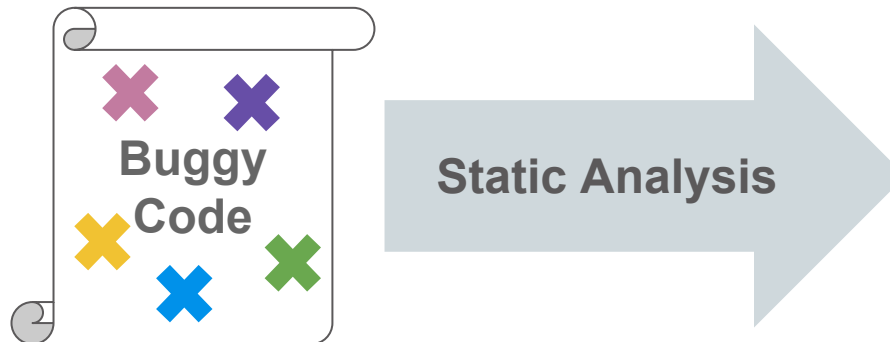
Precision

How much can I trust a tool ?

What kind of flaws can a tool find ?

Recall

What proportion of flaws can a tool find ?



Evaluation Metrics

Precision

How much can I trust a tool ?

Coverage

What kind of flaws can a tool find ?

Recall

What proportion of flaws can a tool find ?



Evaluation Metrics

Precision

How much can I trust a tool ?

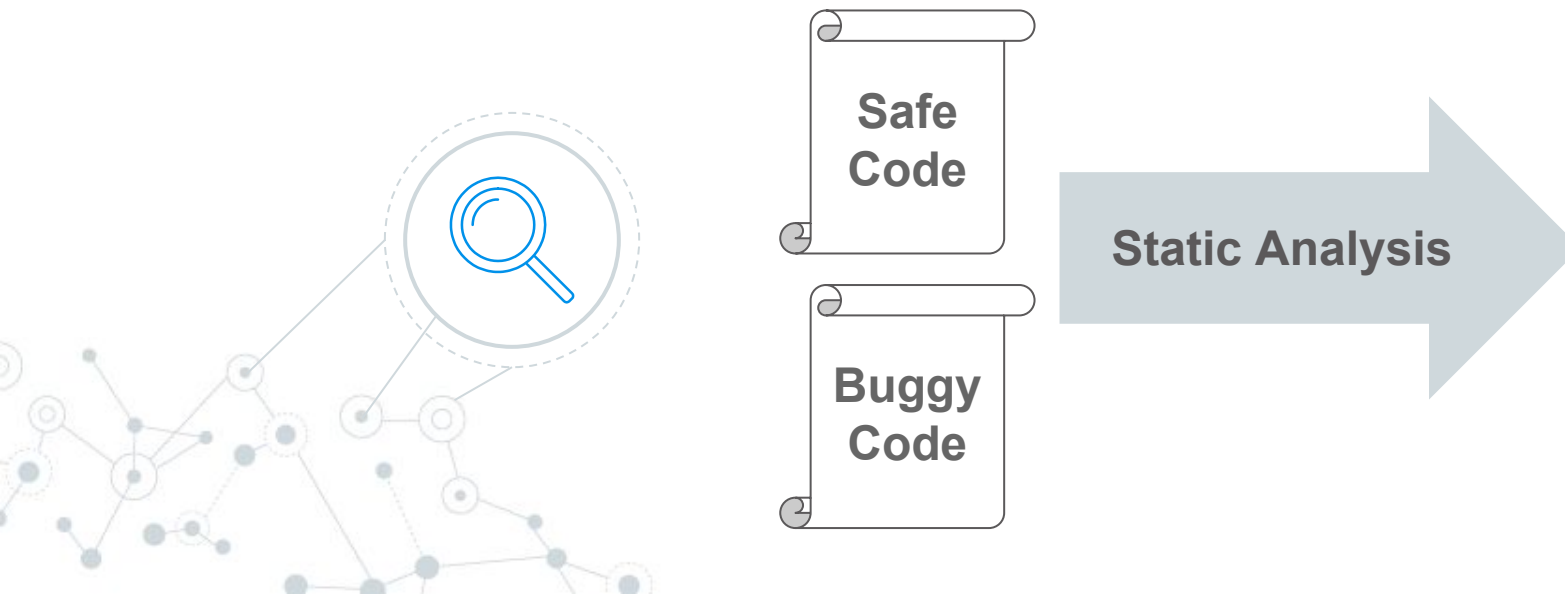
Coverage

What kind of flaws can a tool find ?

Recall

What proportion of flaws can a tool find ?

How smart is a tool ?



Evaluation Metrics

Precision

How much can I trust a tool ?

Coverage

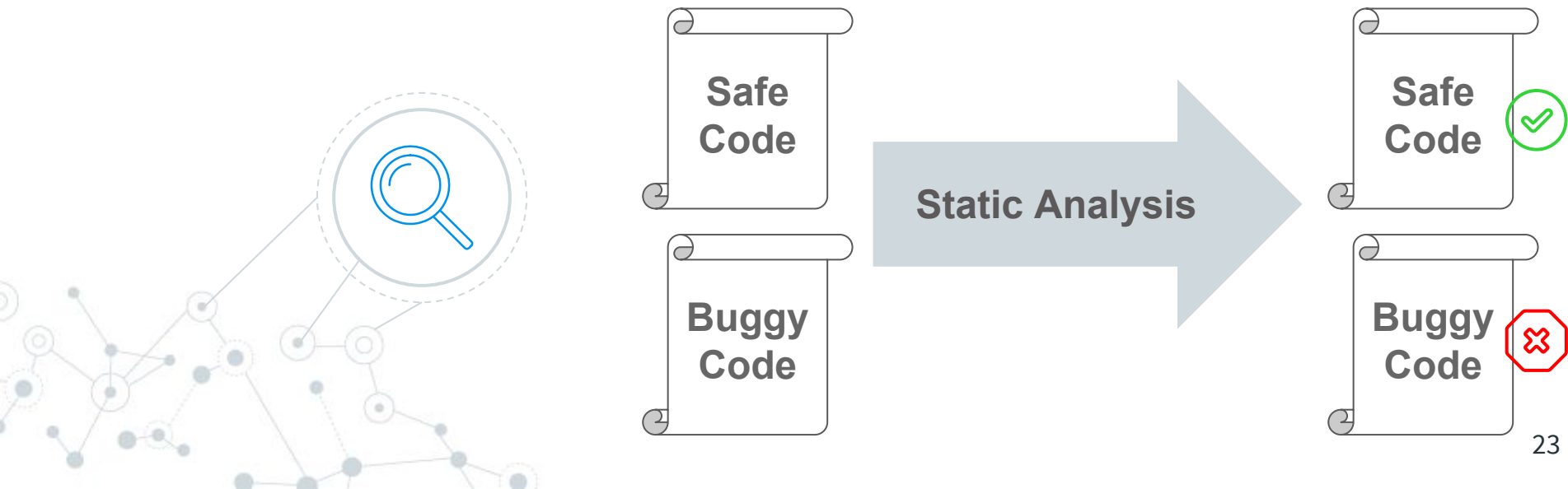
What kind of flaws can a tool find ?

Recall

What proportion of flaws can a tool find ?

Discrimination

How smart is a tool ?



Evaluation Metrics

Precision

How much can I trust a tool ?

Recall

What proportion of flaws can a tool find ?

Coverage

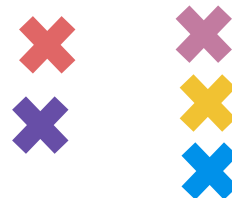
What kind of flaws can a tool find ?

Discrimination

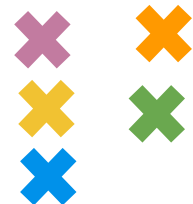
How smart is a tool ?

How similar are unrelated tools ?

Bugs report



Bugs report



Evaluation Metrics

Precision

How much can I trust a tool ?

Recall

What proportion of flaws can a tool find ?

Coverage

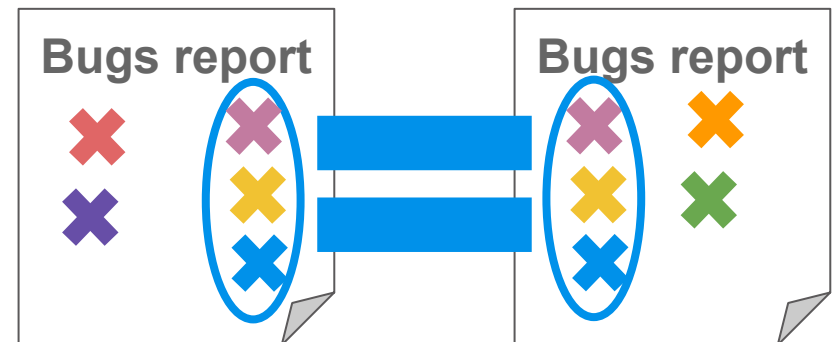
What kind of flaws can a tool find ?

Discrimination

How smart is a tool ?

Overlap

How similar are unrelated tools ?



A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines, with some nodes highlighted in blue and others in grey.

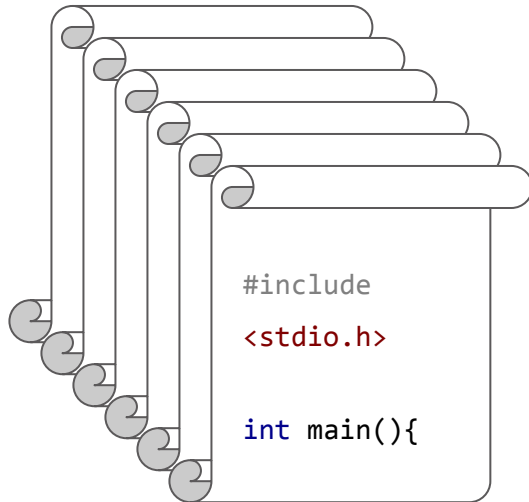
4. **Test Cases**

Static Analysis Tool Exposition (SATE)



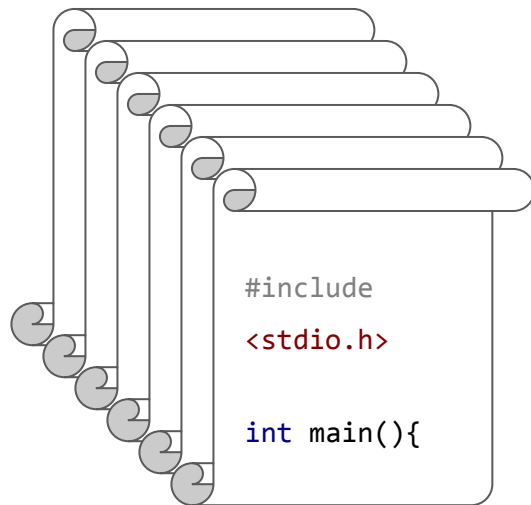
Design of Test Cases

**Statistical
significance**

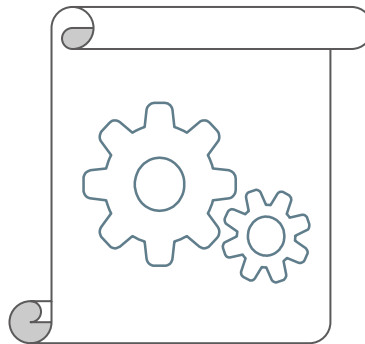


Design of Test Cases

Statistical significance

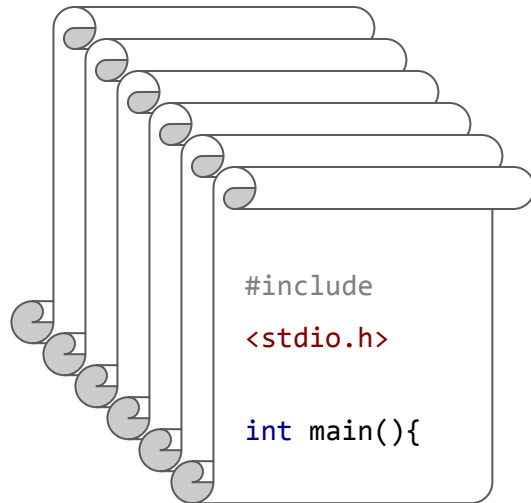


Relevance

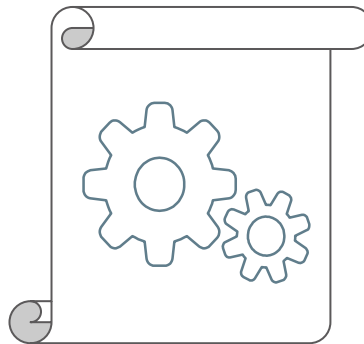


Design of Test Cases

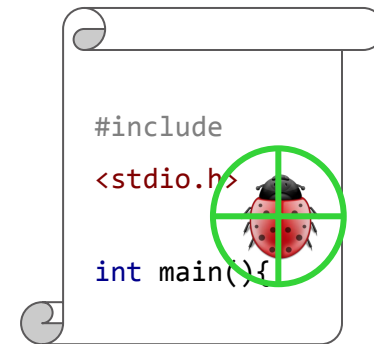
Statistical significance



Relevance

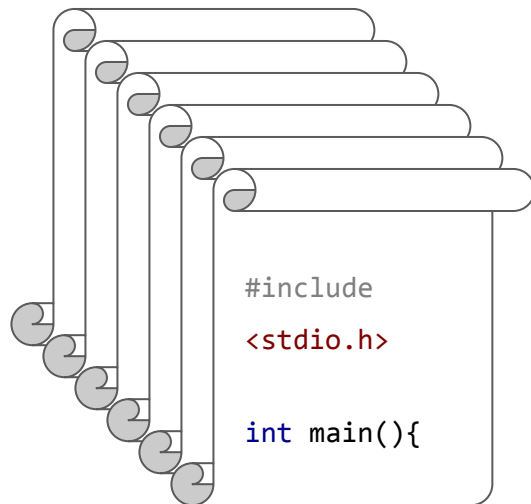


Ground Truth

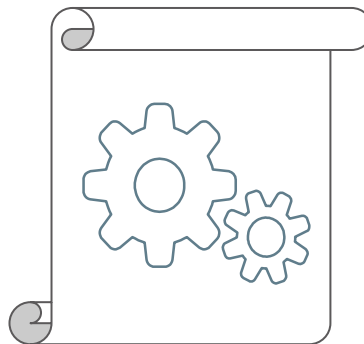


Design of Test Cases

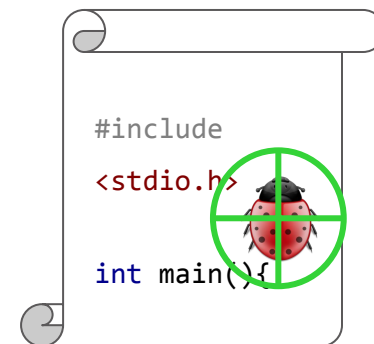
Statistical significance



Relevance



Ground Truth

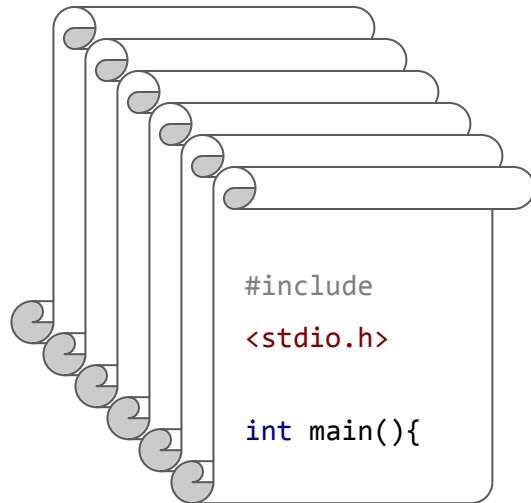


- © Types of Test Cases:
 - Software with Common Vulnerability Enumeration (CVE)
 - Production Software
 - Synthetic Test Cases

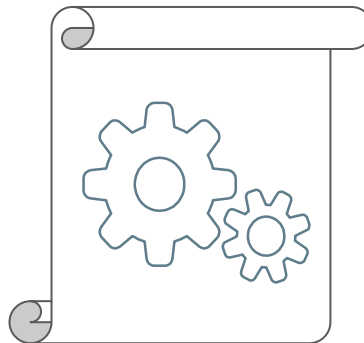
Design of Test Cases

Software w/ CVEs

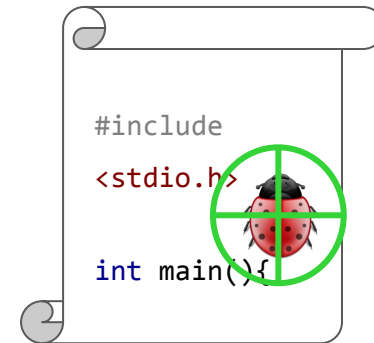
Statistical
significance



Relevance



Ground Truth



© Types of Test Cases:

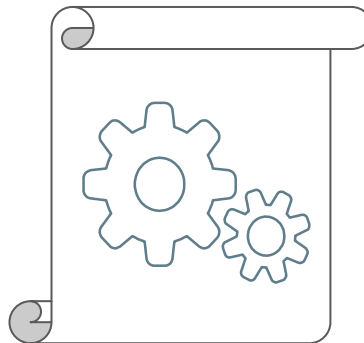
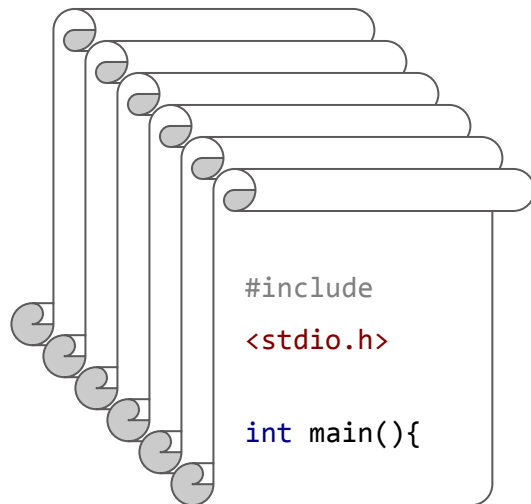
- Software with Common Vulnerability Enumeration (CVE)
- Production Software
- Synthetic Test Cases

Design of Test Cases

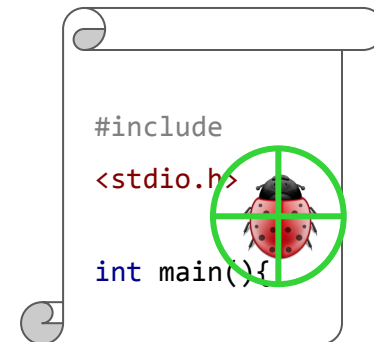
Production Software

**Statistical
significance**

Relevance



Ground Truth



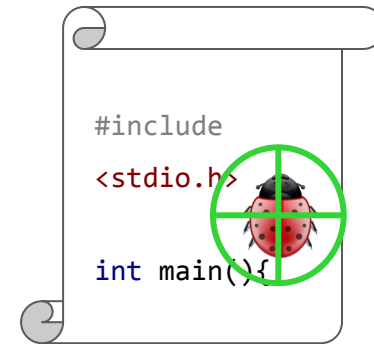
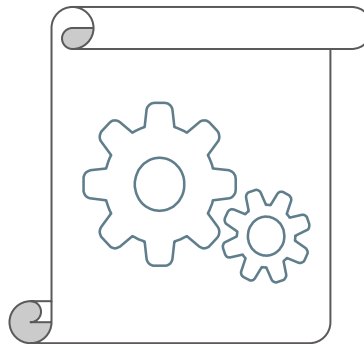
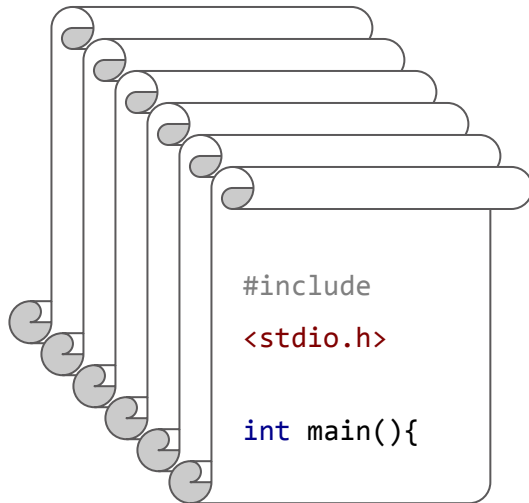
- © Types of Test Cases:
 - Software with Common Vulnerability Enumeration (CVE)
 - **Production Software**
 - Synthetic Test Cases

Design of Test Cases

Synthetic Cases
















**Statistical
significance**

Ground Truth



- © Types of Test Cases:
 - Software with Common Vulnerability Enumeration (CVE)
 - Production Software
 - Synthetic Test Cases

Mapping Metrics to Data

Question	Production Software	Software w/ CVEs	Synthetic Test Cases
Coverage			
Recall			
Precision			
Discrimination			
Overlap			



Applicable - Metric can be computed



Limited - Some limitations with the calculation



N/A - Not Applicable

5. **Results**



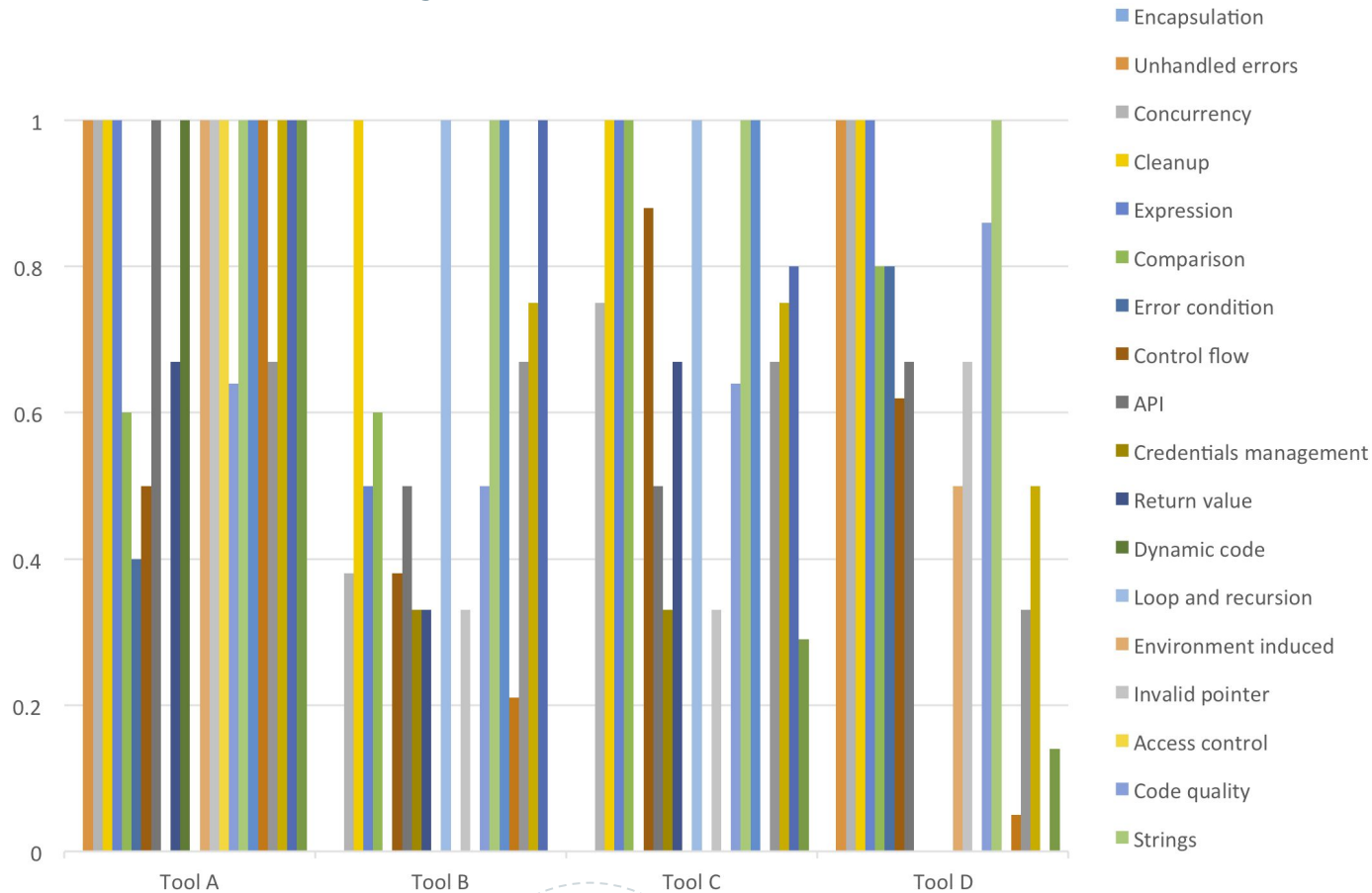
A background network diagram consisting of numerous small circular nodes connected by thin lines, forming a complex web-like structure. The nodes are light gray with darker gray outlines, and the lines are thin and light gray.

3,480,195

Warnings to analyze* !

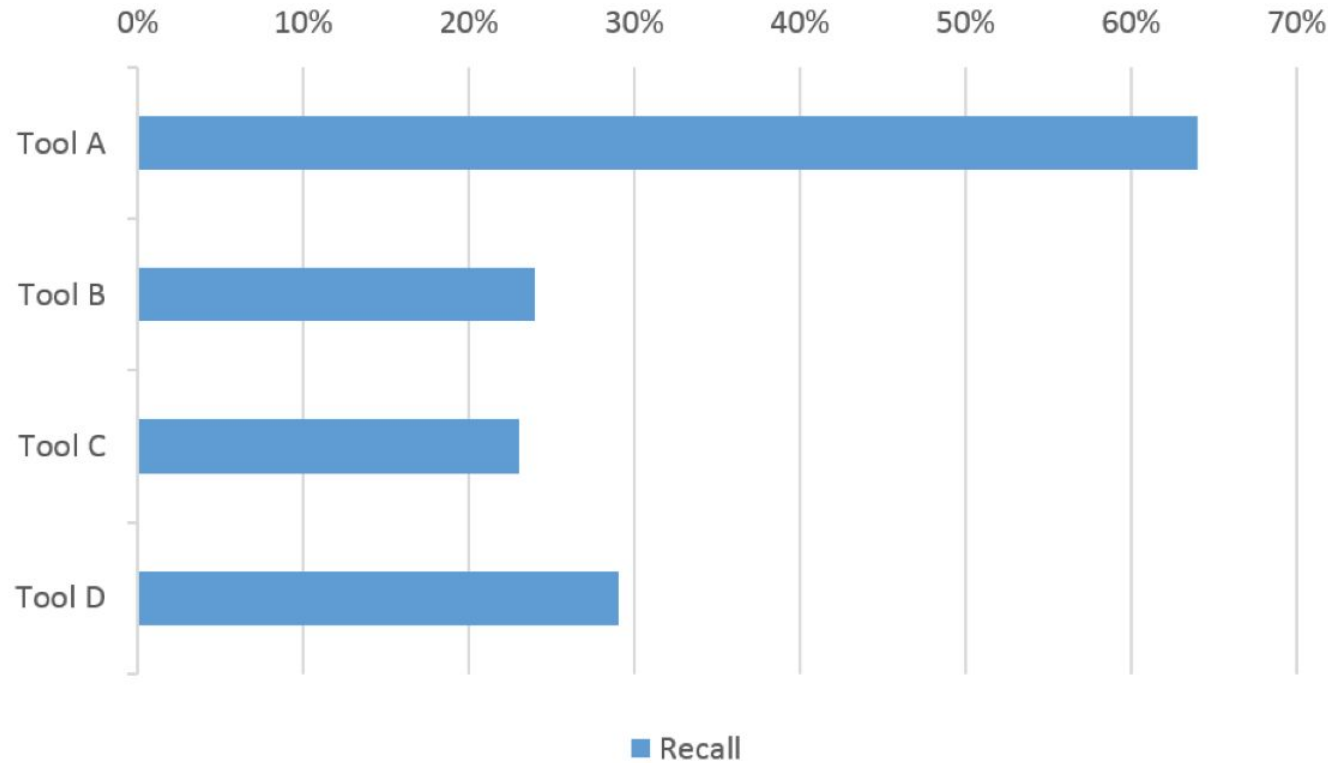
Coverage Spectrum per Tool

For Synthetic Java



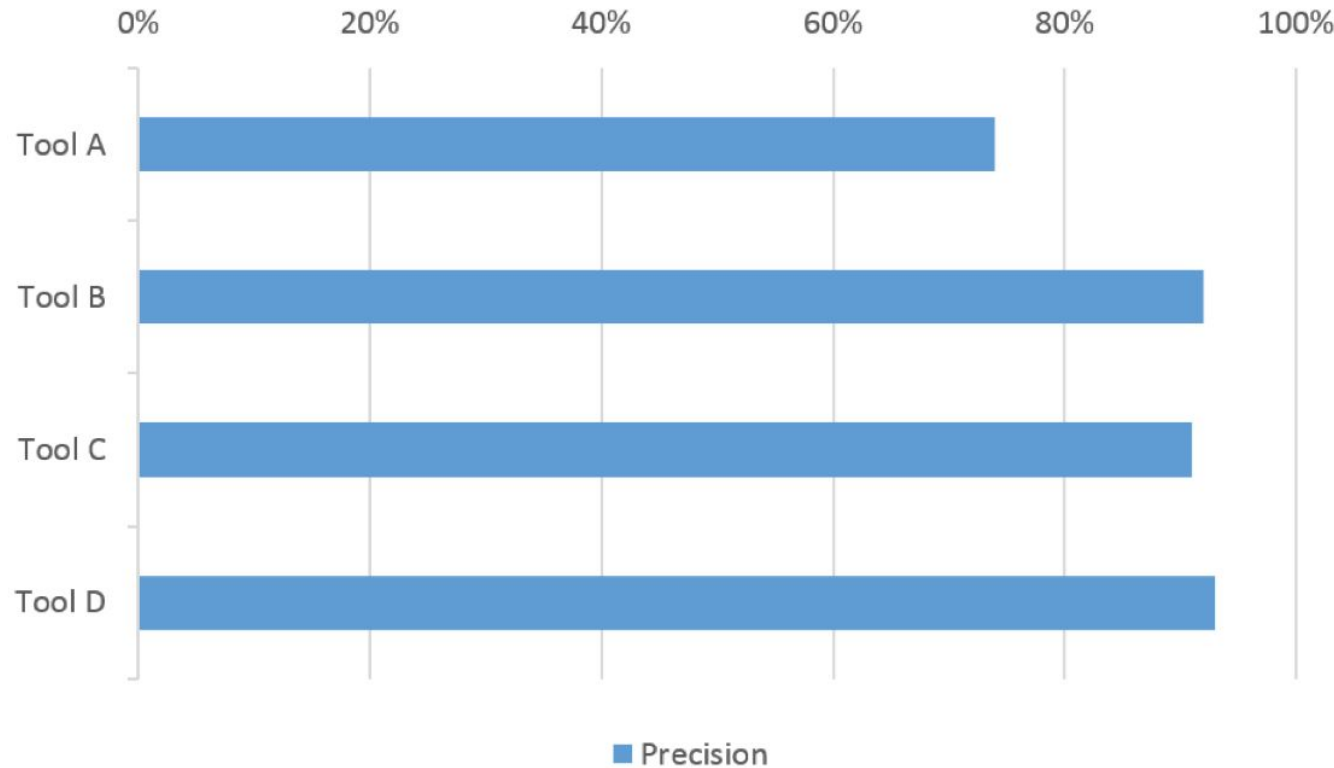
Recall per Tool

For Synthetic Java



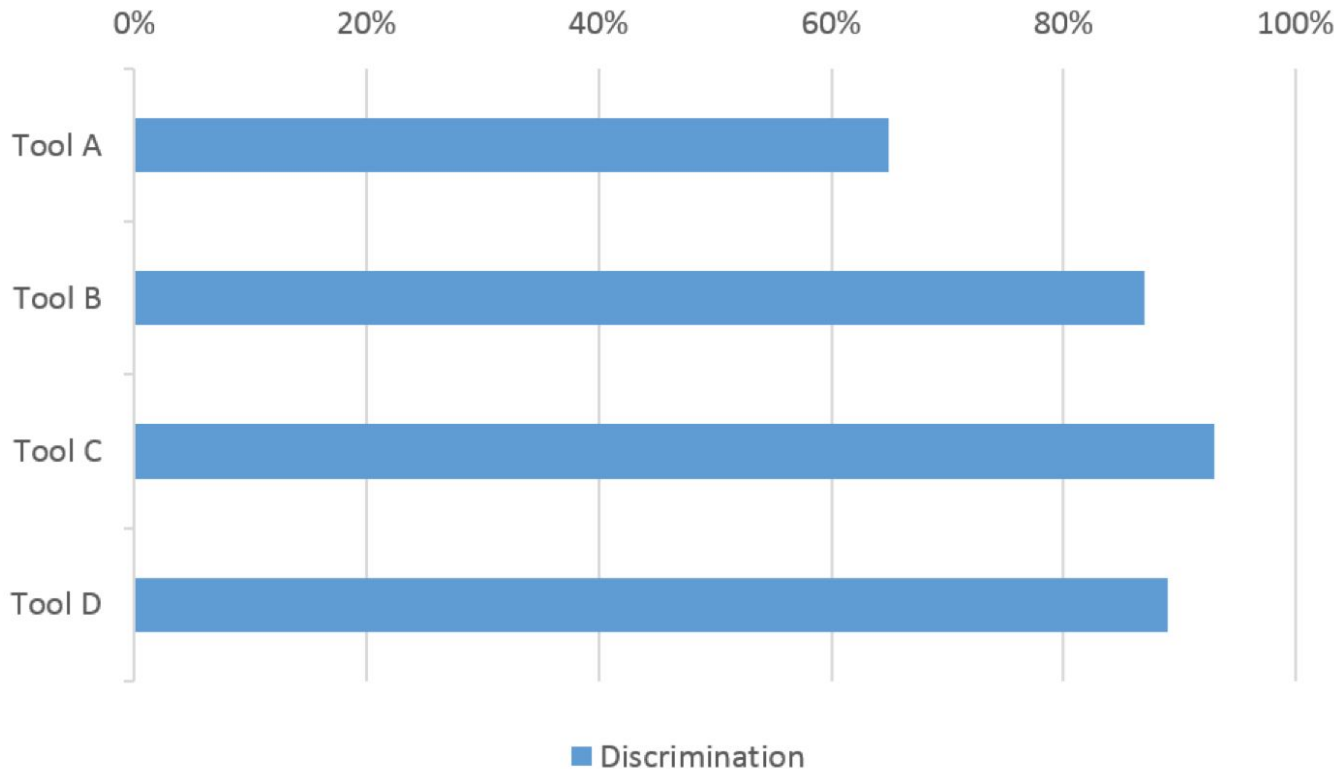
Precision per Tool

For Synthetic Java

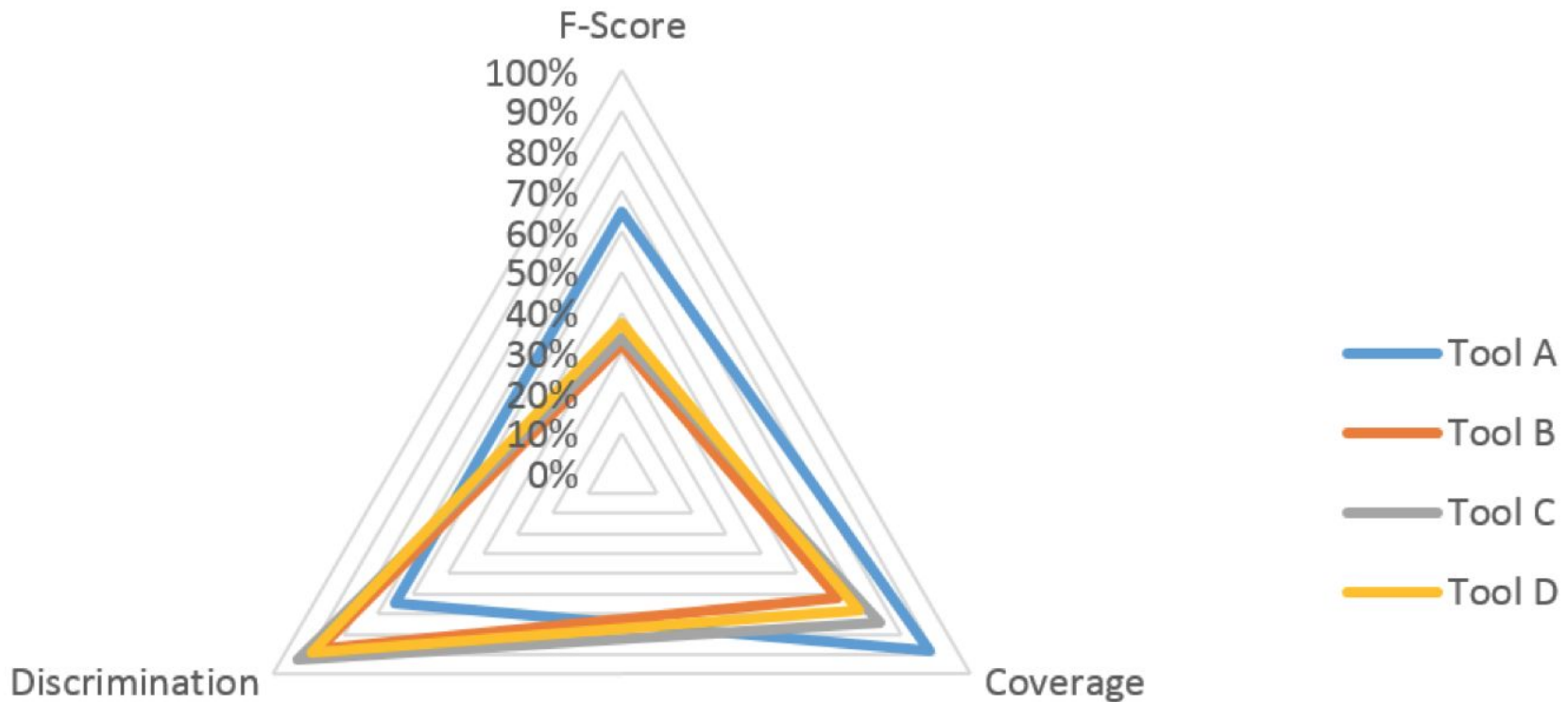


Discrimination per Tool

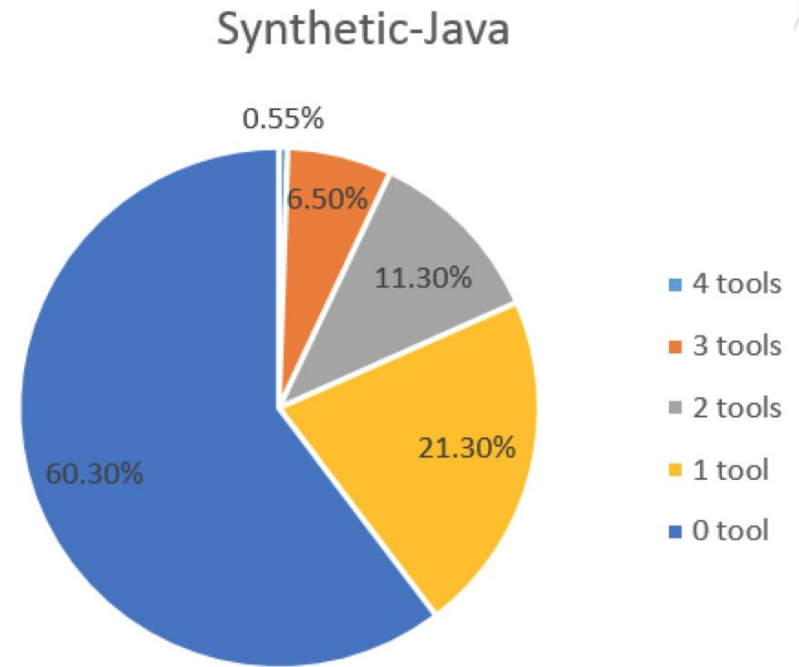
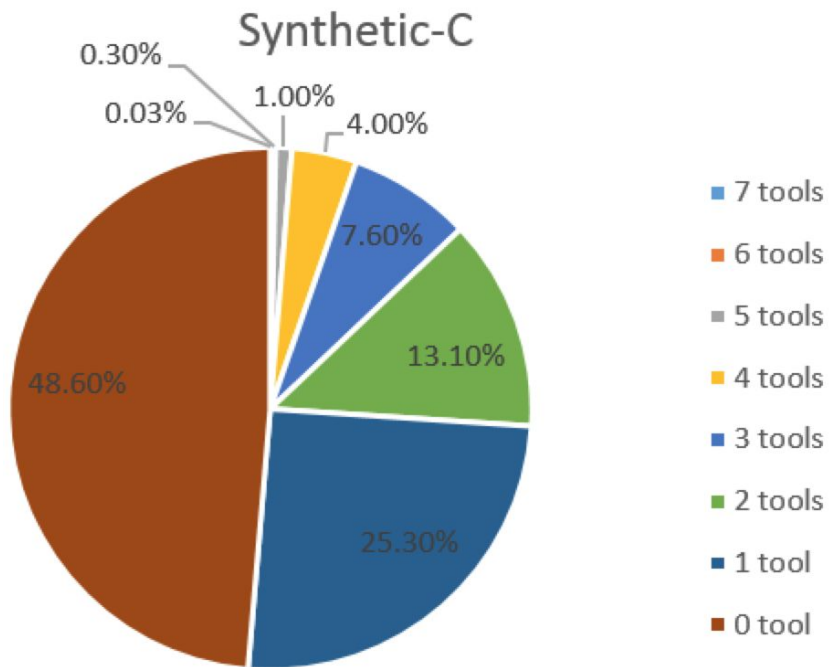
For Synthetic Java



Combination of Tool Metrics



Findings' Overlap



Code Complexity

```
1. char * data;  
2.  
3.  
4. data = NULL;  
5.  
6. char myString[] = "myString";  
7. data = strdup(myString);  
8.  
9.  
10.  
11. delete [] data;  
12.
```

```
1. char * data;  
2. char * *dataPtr1 = &data;  
3. char * *dataPtr2 = &data;  
4. data = NULL;  
5. char * data = *dataPtr1;  
6. char myString[] = "myString";  
7. data = strdup(myString);  
8. *dataPtr1 = data;  
9. {  
10.     char * data = *dataPtr2;  
11.     delete [] data;  
12. }
```

Code Complexity

```
1. char * data;  
2.  
3.  
4. data = NULL;  
5.  
6. char myString[] = "myString";  
7. data = strdup(myString);  
8.  
9.  
10.   
11. delete [] data;  
12.
```

A blue circle highlights the `strdup` function call in line 7. A blue arrow points from this circle to another blue circle in line 11 that highlights the `delete` statement. This illustrates a memory leak because the memory allocated by `strdup` is never freed.

```
1. char * data;  
2. char * *dataPtr1 = &data;  
3. char * *dataPtr2 = &data;  
4. data = NULL;  
5. char * data = *dataPtr1;  
6. char myString[] = "myString";  
7. data = strdup(myString);  
8. *dataPtr1 = data;  
9. {  
10. char * data = *dataPtr2;  
11. delete [] data;  
12. }
```

A blue circle highlights the `strdup` function call in line 7. A blue arrow points from this circle to another blue circle in line 11 that highlights the `delete` statement. This illustrates a memory leak because the memory allocated by `strdup` is never freed.

CWE 762: Mismatched Memory Management Routines

Complexity vs. Tool Effectiveness

```
1. char * data;  
2.  
3.  
4. data = NULL;  
5.  
6. char myString[] = "myString";  
7. data = strdup(myString);  
8.  
9.  
10.  
11. delete [] data;  
12.
```



Found by tool X



Found by tool Y

```
1. char * data;  
2. char * *dataPtr1 = &data;  
3. char * *dataPtr2 = &data;  
4. data = NULL;  
5. char * data = *dataPtr1;  
6. char myString[] = "myString";  
7. data = strdup(myString);  
8. *dataPtr1 = data;  
9. {  
10.     char * data = *dataPtr2;  
11.     delete [] data;  
12. }
```



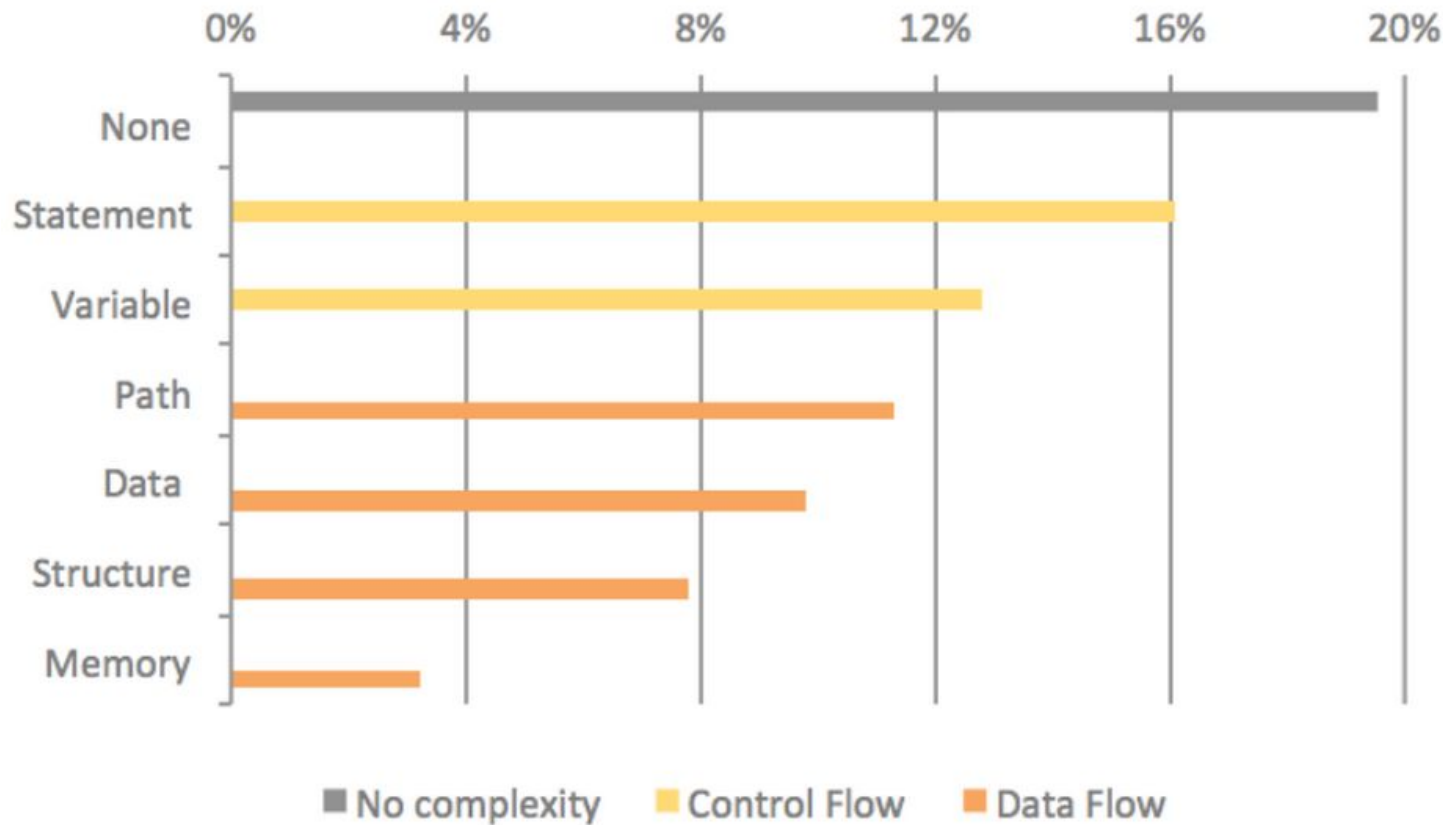
Found by tool X



Missed by tool Y

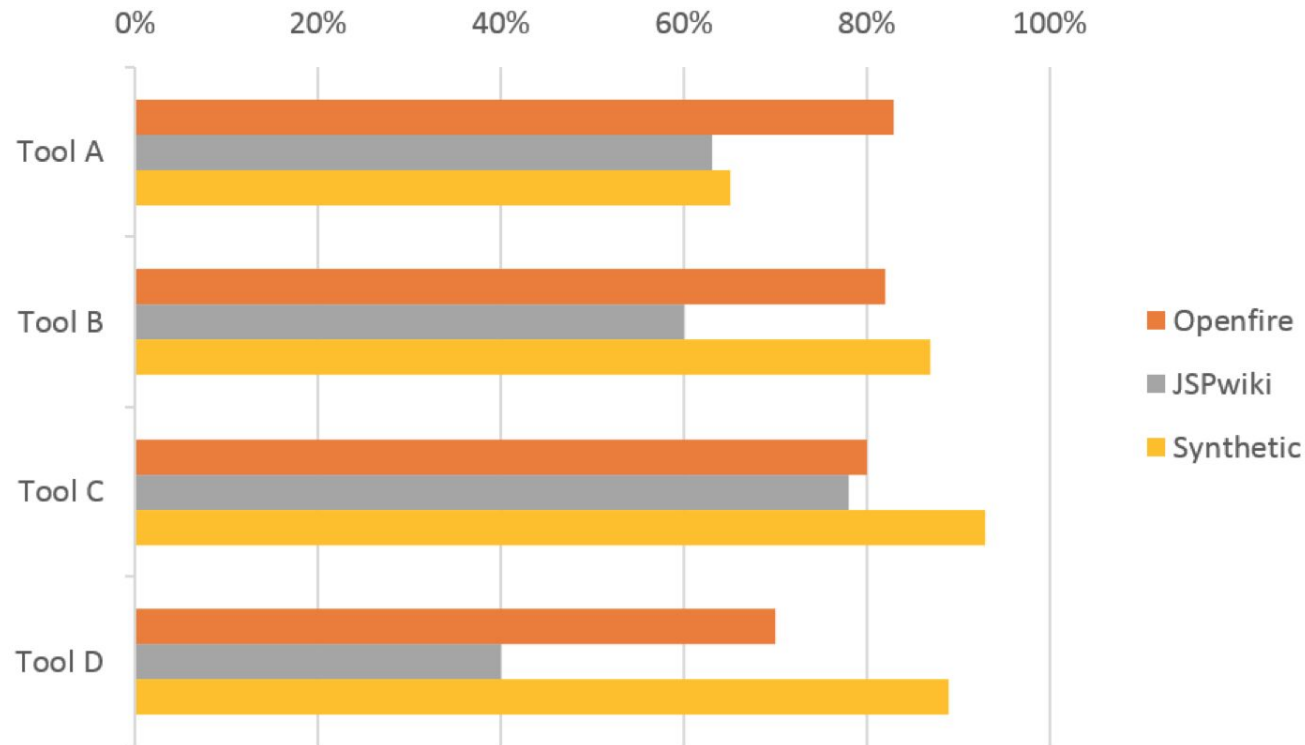
Recall per Complexity

For Synthetic C



Precision per Tool

On Production Software vs. Synthetic Java



5.

Conclusion



Conclusion

- ◎ Tools need evaluation!
- ◎ Test cases need improvement
- ◎ Testing procedure needs more metrics:
 - Usability
 - Integration
 - Impact



Thanks!

Any questions?

Find us at:

<http://samate.nist.gov>

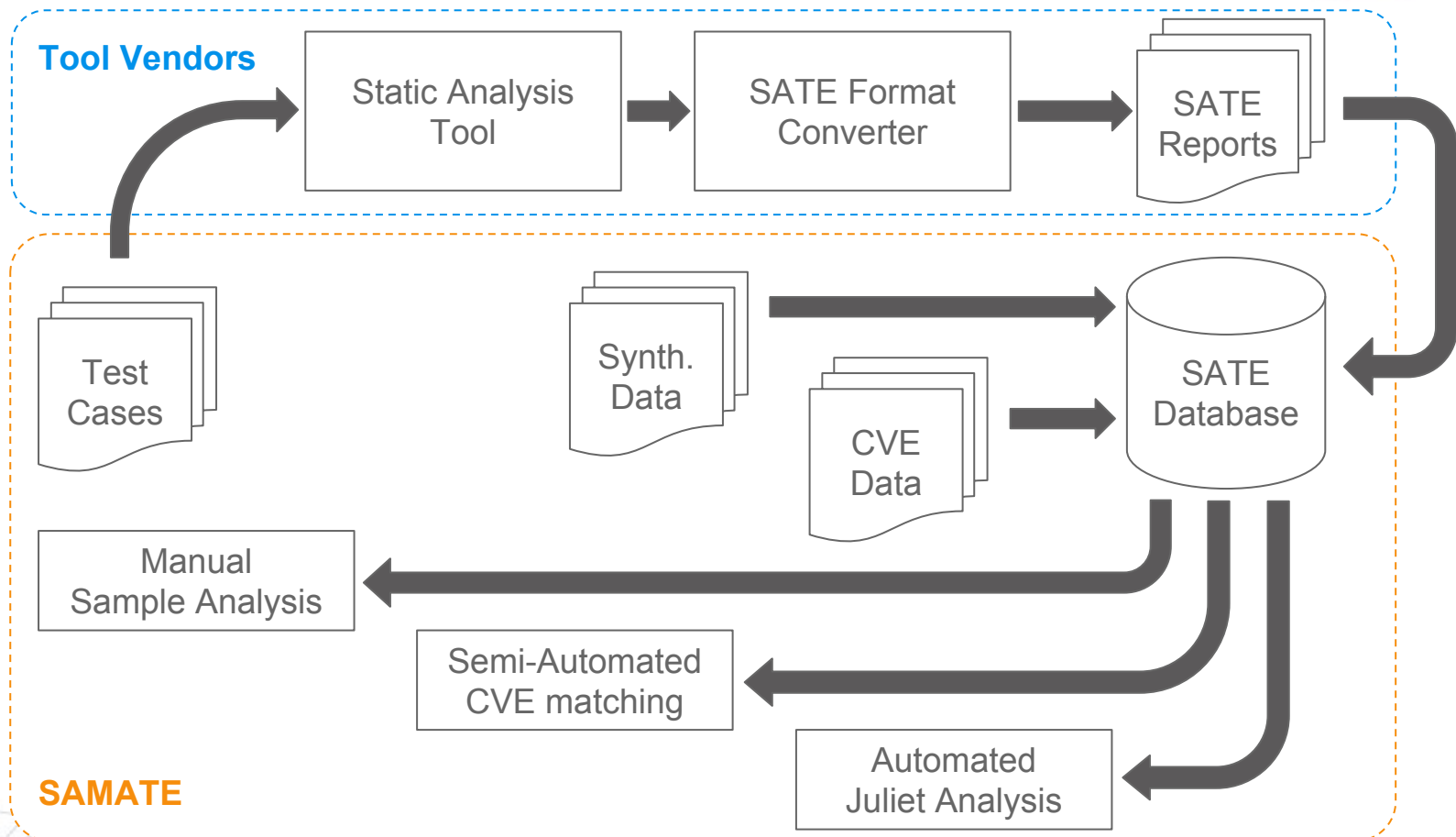
samate@nist.gov





SATE


The Art of Collecting Data



Evaluation Metrics



Question	Metrics
What proportion of defects can a tool find ?	Recall / Coverage
How noisy is a tool ?	Precision / Discrimination
How similar are unrelated tools ?	Overlap



Complexity

- ◎ Different kinds of complexities in the **Synthetic Test Cases**
 - None

No complexity

```
int main()
{
    char buf[15];

    cin >> buf;
    cout << "echo: " << buf << endl;

    return 0;
}
```

Complexity

◎ Different kinds of complexities in the **Synthetic Test Cases**

- None
- **Control Flow**

Control Flow complexity

```
int main()
{
    char buf[15] = "COUFLESS2015";

    if (1) cin >> buf;
    cout << "echo: " << buf << endl;

    return 0;
}
```

Complexity

◎ Different kinds of complexities in the **Synthetic Test Cases**

- None
- Control Flow
- **Data Flow**

Data Flow complexity

```
char *strcpy(char *str1, char *str2)
{
    while (*str2)
        *str1++ = *str2++;
    return str1;
}

int main(int argc, char **argv)
{
    char *buffer = (char *)malloc(16 * sizeof(char));
    strcpy(buffer, argv[1]);
    printf("%s\n", buffer);
    return 0;
}
```