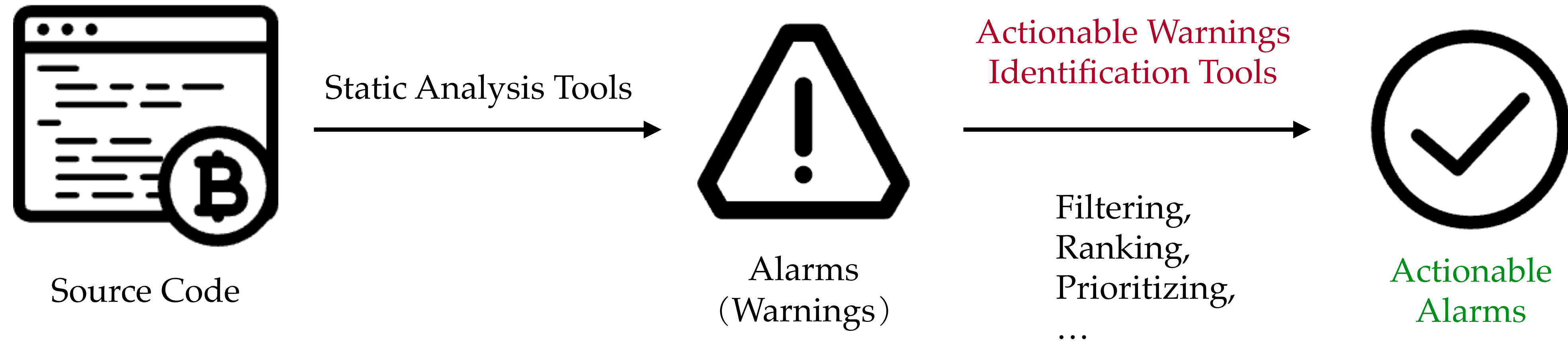


# Detecting False Alarms from Automatic Static Analysis Tools: How Far are We?

A brief literature review on **AWI**

# From Static Analysis to AWI

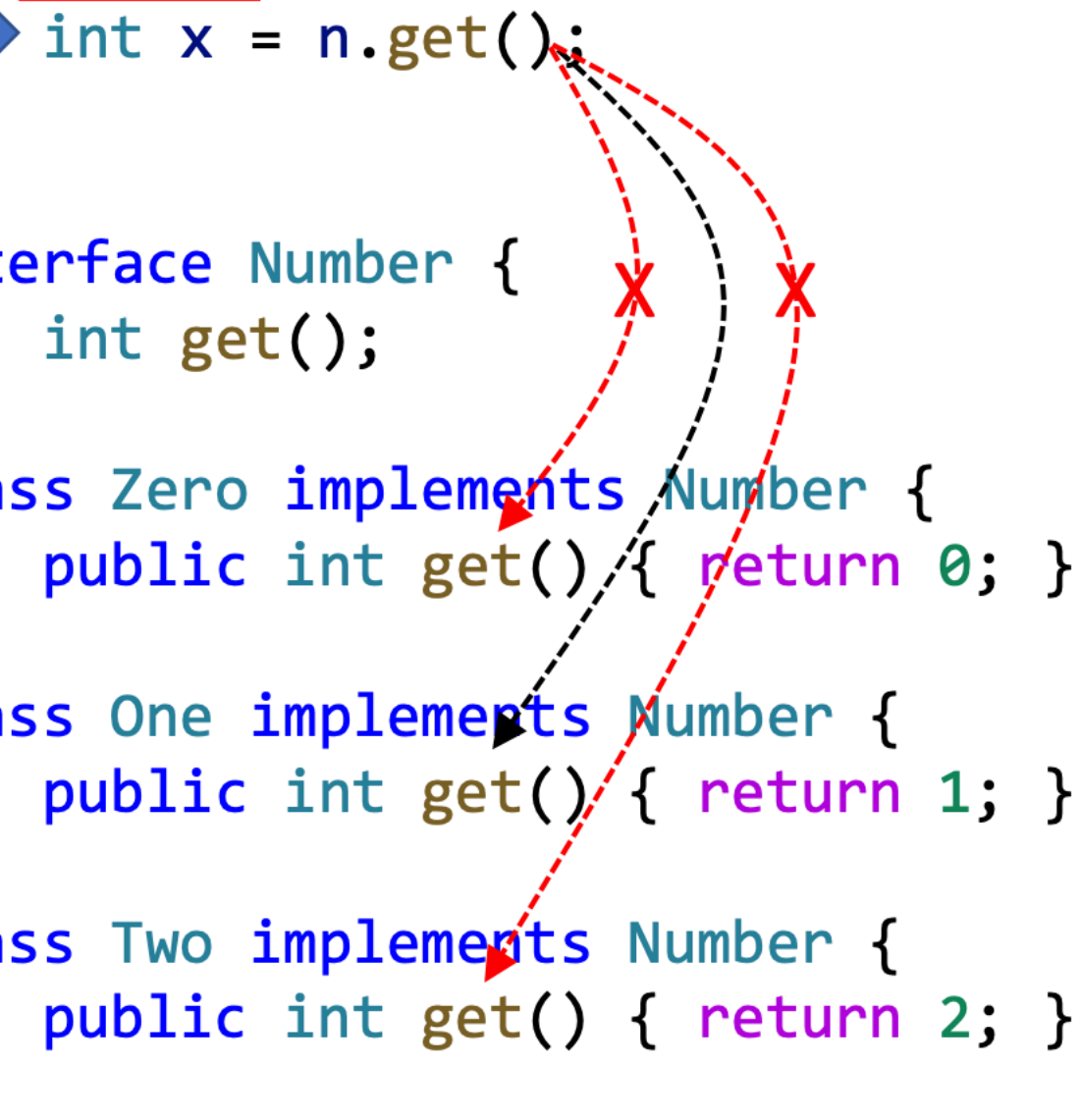


# Over-approximation!

for most static(may) analyses

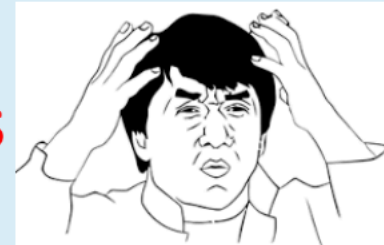
## Problem of CHA

```
void foo() {  
    Number n = new One();  
    int x = n.get();  
}  
  
interface Number {  
    int get();  
}  
  
class Zero implements Number {  
    public int get() { return 0; }  
}  
  
class One implements Number {  
    public int get() { return 1; }  
}  
  
class Two implements Number {  
    public int get() { return 2; }  
}
```



CHA: ~~based on~~ only considers  
class hierarchy

- 3 call targets
- 2 false positives



Constant propagation

- x = NAC **imprecise**

The larger number of **false alarms** produced poses a barrier to adoption.

# Practice to Describe / Model Alarms

Reproduced by

doi:10.1007/s10664-021-09948-6



## Is There A “Golden” Feature Set for Static Warning Identification?

— An Experimental Evaluation

- **RQ1:** Is there a common set of features that takes effect in warning identification for most project revisions?
- **RQ2:** What is the performance (i.e., AUC and time cost) of the common feature set in warning identification?

# Practice to Describe / Model Alarms

«”Golden” Feature Set...»

Table 1: Overview of the collected features

Id	Name; Meaning	Cat.	Reference	#Rev.
F1, F2, F3*	file name, package name, project name; <i>the file name, package name, project name where the warning locates</i>	fChr	[6, 7, 14, 26]	17,15
F4	file type; <i>file extension name</i>	fChr	[7, 17]	1
F5*	SA tool name	fChr	[15]	n/a
(F6-F8)*	location, file, project warnings for tool; <i>warnings reported for the same location, file, project by each tool</i>	fChr	[15, 26]	n/a
F9, F10, F11*	latest file, package, project modification; <i>latest modification revision</i>	fHst	[7]	15,9
F12, F13, F14*	file, package, project staleness; <i>amount of time between current revision and last modification revision of file, package, project</i>	fHst	[7, 17]	12,11
F15	file age; <i>number of days the file has existed</i>	fHst	[17]	18
F16, F17	file creation, deletion revision	fHst	[7, 11]	24,8
F18	developers; <i>set of developers who have made changes to the file</i>	fHst	[7]	26
F19, F20, F21	method, file, package size; <i>number of non-comment source code statements in method, file, package</i>	cChr	[7, 17]	11,16,15
F22	comment length; <i>number of comment lines in file</i>	cChr	[15]	11
F23	comment-code ratio; <i>ratio of comment length and code length in file</i>	cChr	[15]	18
F24, F25	method, file depth; <i>depth of warned line in method, file</i>	cChr	[15]	24,20
F26, F27	method callers, callees; <i>number of callers, callees of warned method</i>	cChr	[15]	9,17
F28, F29	methods in file, package; <i>number of methods in file, package</i>	cChr	[7]	18,12
F30, F31	classes in file, package; <i>number of (inner) classes in file, package</i>	cChr	[7]	16,22
F32	indentation; <i>spaces indenting warned line</i>	cChr	[17]	16
F33	complexity; <i>cyclomatic complexity</i>	cChr	[7]	8
F34-F39	added, changed, deleted, growth, total, percentage of lines of code in file during the past 3 months	cHst	[17]	10-17
F40-F45	added, changed, deleted, growth, total, percentage of lines of code in file during the past 25 revisions	cHst	[7]	9-22
F46-F51	added, changed, deleted, growth, total, percentage of lines of code in package during the past 3 months	cHst	[17]	12-18
F52-F57	added, changed, deleted, growth, total, percentage of lines of code in package during the past 25 revisions	cHst	[7]	10-17
(F58-F63)*	added, changed, deleted, growth, total, percentage of lines of code in project during the past 3 months	cHst	[17]	n/a
(F64-F69)*	added, changed, deleted, growth, total, percentage of lines of code in project during the past 25 revisions	cHst	[7]	n/a
F70-F73	call name, class, parameter signature, return type; <i>name of method being called, name of class containing the method</i>	cAnl	[5]	14,11,24,15
F74, F75	new type, new concrete type; <i>class, or concrete type of object being created</i>	cAnl	[5]	9,9
F76	operator; <i>operator for the binary operation</i>	cAnl	[5]	14
F77, F78	field access class, field; <i>class containing the field being accessed, name of field being accessed</i>	cAnl	[5]	3,1
F79	catch; <i>whether a catch statement is present</i>	cAnl	[5]	12
F80-F83	field name, type, visibility, is static/final	cAnl	[5]	0,4,2,1
F84-F86	method visibility, return type, is static/final/abstract/protected	cAnl	[5]	19,16,0
F87, F88	class visibility, is abstract/interfact/array class	cAnl	[5]	8,0
F89-F92, F93*	warning pattern, type, priority, rank, range (Google warning descriptors)	wChr	[6, 7, 11, 17]	18,18,24,13
F94-F96	warnings in method, file, package; <i>number of warnings in method, file, package</i>	wChr	[6, 7, 17]	0,17,18
F97	warning modifications; <i>number of times the warning's line number has changed</i>	wHst	[7]	11
F98	warning open revision	wHst	[7, 11]	15
F99, F100	warning lifetime by revision, by time; <i>number of revisions, amount of time between current revision and open revision</i>	wHst	[7, 11, 26]	25,14
F101*	developer idea; <i>four different idea: fix, not a problem, ignore, analyse</i>	wHst	[26]	n/a
F102	size context for a warning type; <i>number of warnings from a warning type normalized by S; S is total number of warnings</i>	wCmb	[6, 7]	16
F103-F105	size context in method, file, package; <i>number of warnings in the method, file, package normalized by S</i>	wCmb	[6, 7]	14,11,15
F106	warning context for warning type; <i>difference of actionable and unactionable warnings for a warning type normalized by S</i>	wCmb	[6, 7]	20
F107-F109	warning context in method, file, package; <i>difference of actionable and unactionable warnings for the method, file, package normalized by S</i>	wCmb	[6, 7]	30,26,10
F110, F111	fix, non-fix change removal rate; <i>warnings in a type that is removed by bug-fix commit, non-bug-fix commit normalized by S</i>	wCmb	[12]	14,12
F112	defect likelihood for warning pattern; $D(P_{ij}) = T_{ij}/(T_{ij}+F_{ij})$ , where $P_{ij}$ denotes the $j_{th}$ warning pattern in $C_i$ warning type, $T_{ij}$ is number of actionable warnings for pattern $P_{ij}$ , $F_{ij}$ is number of unactionable warnings for pattern $P_{ij}$	wCmb	[18]	23
F113	variance of likelihood; <i>variance of <math>D(P_{ij})</math> for each warning pattern <math>P_{ij}</math> in a warning type <math>C_i</math></i>	wCmb	[18]	17
F114	defect likelihood for warning type; $D(C_i) = D(P_{i1})*S_{i1} + \dots + D(P_{in})*S_{in}$ , where $S_{ij} = T_{ij}+F_{ij}$	wCmb	[18]	13
F115	discretization of defect likelihood; <i>discretization of <math>D(C_i)</math> for each warning type <math>C_i</math> in the project</i>	wCmb	[18]	23
F116	average lifetime for warning type; <i>average value for feature F100 for a warning type</i>	wCmb	[11]	22

Note: The features marked with \* denote they will not be experimentally investigated in this paper.  
#Rev. is the number of project revisions in which the features are selected (details are in Section 6.1).

Table 4: Commonly-selected features (RQ1)

Cat.	Feature	#Rev. (Revision Information)
wCmb  <b>6/15=40%</b>	F107: warning context in method	30( <i>lu:5, tm:5, mv:5, dr:5, po:5, ph:5</i> )
	F108: warning context in file	26( <i>lu:2, tm:5, mv:5, dr:4, po:5, ph:5</i> )
	F112: defect likelihood for warning pattern	23( <i>lu:3, tm:5, mv:2, dr:3, po:5, ph:5</i> )
	F115: discretization of defect likelihood	23( <i>lu:5, tm:5, mv:1, dr:4, po:5, ph:3</i> )
	F116: average lifetime for warning type	22( <i>lu:5, tm:3, mv:1, dr:3, po:5, ph:5</i> )
cChr  <b>5/15=33%</b>	F106: warning context for warning type	20( <i>lu:5, tm:4, mv:2, dr:0, po:5, ph:4</i> )
	F24: method depth	24( <i>lu:5, tm:5, mv:1, dr:4, po:4, ph:5</i> )
	F31: classes in package	22( <i>lu:4, tm:5, mv:3, dr:2, po:5, ph:3</i> )
	F25: file depth	20( <i>lu:5, tm:2, mv:3, dr:2, po:5, ph:3</i> )
	F23: comment-code ratio	18( <i>lu:0, tm:3, mv:3, dr:5, po:4, ph:3</i> )
wChr  <b>4/7=57%</b>	F28: methods in file	18( <i>lu:0, tm:4, mv:4, dr:1, po:4, ph:5</i> )
	F91: warning priority	24( <i>lu:5, tm:4, mv:4, dr:3, po:5, ph:3</i> )
	F89: warning pattern	18( <i>lu:4, tm:4, mv:1, dr:2, po:5, ph:2</i> )
	F90: warning type	18( <i>lu:0, tm:5, mv:3, dr:1, po:5, ph:4</i> )
	F96: warnings in package	18( <i>lu:1, tm:5, mv:0, dr:2, po:5, ph:5</i> )
fHst  <b>3/8=37%</b>	F18: developers	26( <i>lu:5, tm:5, mv:1, dr:5, po:5, ph:5</i> )
cAnl  <b>2/19=10%</b>	F16: file creation revision	24( <i>lu:5, tm:5, mv:2, dr:4, po:5, ph:3</i> )
	F15: file age	18( <i>lu:1, tm:3, mv:2, dr:3, po:5, ph:4</i> )
	F72: parameter signature	24( <i>lu:5, tm:5, mv:2, dr:2, po:5, ph:5</i> )
cHst  <b>2/24=8%</b>	F84: method visibility	19( <i>lu:5, tm:5, mv:0, dr:2, po:4, ph:3</i> )
	F40: added lines of code in file during the past 25 revisions	22( <i>lu:5, tm:5, mv:1, dr:3, po:5, ph:3</i> )
	F46: added lines of code in package during the past 3 months	18( <i>lu:1, tm:4, mv:2, dr:2, po:4, ph:5</i> )
wHst  <b>1/4=25%</b>	F99: warning lifetime by rev	25( <i>lu:5, tm:4, mv:4, dr:3, po:5, ph:4</i> )
fChr(0)		

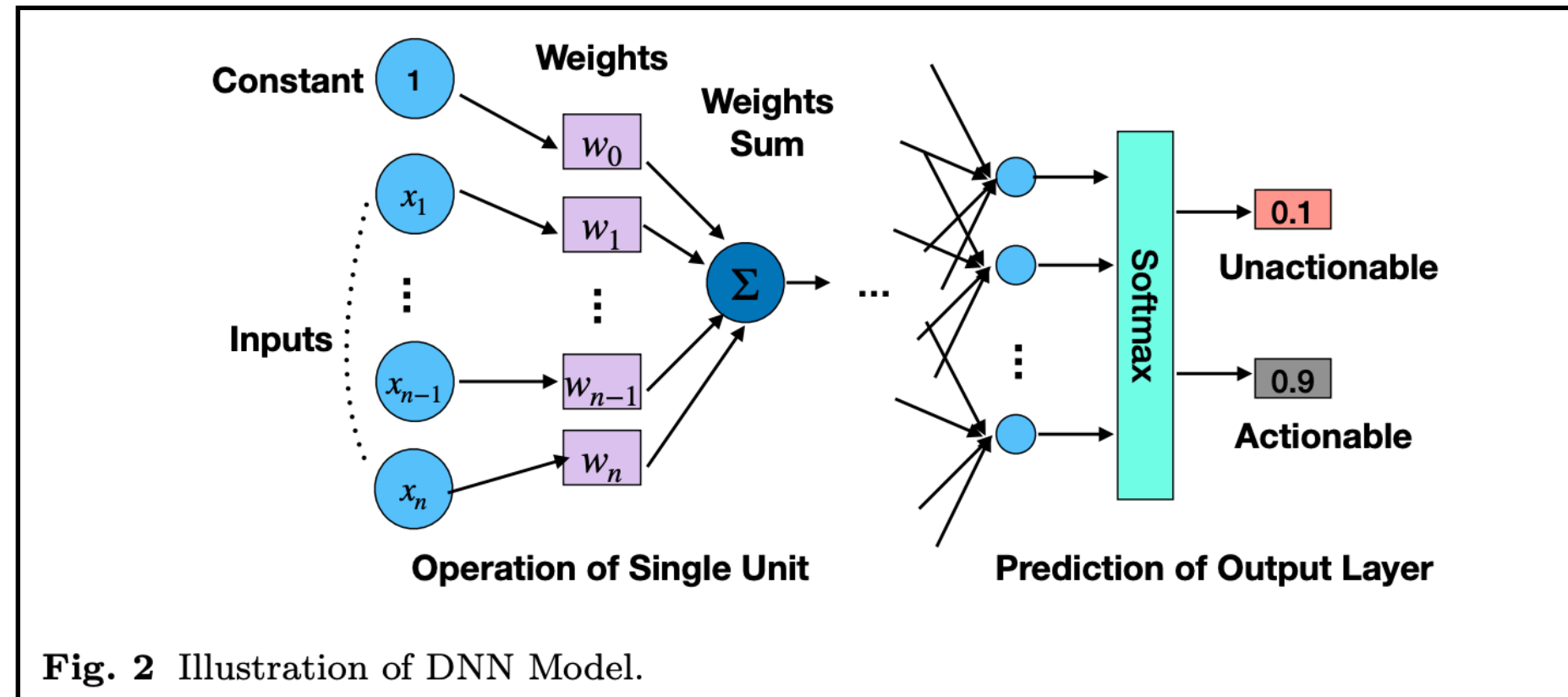
P:X denotes the feature is selected in X revisions of project P.

Projects are abbreviated as follows: **lu**:Lucene-solr, **tm**:Tomcat, **mv**:Maven, **dr**:Derby, **po**:Poi, **ph**:Phoenix.





# Learning to Recognize Actionable Static Code Warnings (is Intrinsically Easy)



**Table 4** Summary results of recall, false alarm and AUC on nine datasets. Cells in gray denote the “best” results for each row, where “best” means within  $d$  difference to the best value (and  $d$  is calculated as per §3.2.)

	Project	DNN weighted	CNN	DNN	Random Forest	Decision Tree	SVM linear
Recall ( $d = 3\%$ )	derby	96.6%	96.9%	94.0%	92.0%	94.8%	97.8%
	mvn	97.6%	95.0%	92.0%	78.9%	94.7%	97.0%
	lucence	95.3%	98.1%	91.3%	96.8%	96.6%	87.1%
	phoenix	95.2%	93.0%	89.3%	88.7%	86.8%	96.1%
	cass	81.3%	98.8%	68.1%	76.8%	75.7%	90.3%
	jmeter	94.3%	93.9%	89.2%	96.9%	92.7%	93.3%
	tomcat	98.0%	95.0%	96.4%	91.8%	87.6%	98.2%
	ant	91.1%	93.1%	84.1%	78.7%	87.0%	95.0%
False Alarm ( $d=2\%$ )	commons	81.1%	97.8%	73.3%	66.7%	92.0%	99.5%
	derby	1.2%	10.8%	0.5%	0.3%	0.5%	1.3%
	mvn	1.4%	6.8%	0.4%	0.5%	0.4%	1.2%
	lucence	5.9%	5.8%	3.2%	1.4%	3.2%	6.9%
	phoenix	3.0%	8.7%	1.4%	1.3%	0.7%	3.5%
	cass	1.2%	7.0%	0.4%	2.5%	1.3%	1.4%
	jmeter	3.1%	48.6%	1.4%	1.3%	0.4%	2.1%
	tomcat	2.1%	8.8%	1.3%	0.4%	4.3%	3.2%
AUC ( $d=1\%$ )	ant	0.5%	6.7%	0.5%	0.4%	0.5%	0.5%
	commons	3.1%	8.6%	1.4%	0.2%	1.4%	5.8%
	derby	99.7%	97.2%	99.6%	99.7%	97.1%	99.5%
	mvn	99.9%	99.1%	99.9%	99.6%	96.8%	99.6%
	lucence	98.7%	98.7%	98.8%	99.6%	96.6%	97.3%
	phoenix	98.5%	97.8%	98.8%	98.6%	92.7%	98.8%
	cass	97.0%	98.0%	96.9%	98.6%	88.0%	99.7%
	jmeter	98.7%	82.3%	97.7%	99.7%	95.9%	98.8%
	tomcat	100.0%	98.3%	99.7%	99.6%	92.1%	99.6%
	ant	98.9%	97.3%	97.7%	98.7%	93.3%	99.7%
	commons	96.0%	99.2%	97.7%	98.7%	96.1%	99.0%



# Supports of “Golden” Feature Set

Understanding static code warnings: An incremental AI approach

Xueqi Yang<sup>a,\*</sup>, Zhe Yu<sup>a</sup>, Junjie Wang<sup>b</sup>, Tim Menzies<sup>a</sup>

<sup>a</sup> *Department of Computer Science, North Carolina State University, Raleigh, NC, USA*

<sup>b</sup> *Institute of Software Chinese Academy of Sciences, Beijing, China*

## **Documenting Evidence of a Reproduction of ‘Is There A “Golden” Feature Set for Static Warning Identification? — An Experimental Evaluation’**

Xueqi Yang

xyang37@ncsu.edu

North Carolina State University

Raleigh, North Carolina, USA

Tim Menzies

timmm@ieee.org

North Carolina State University

Raleigh, North Carolina, USA

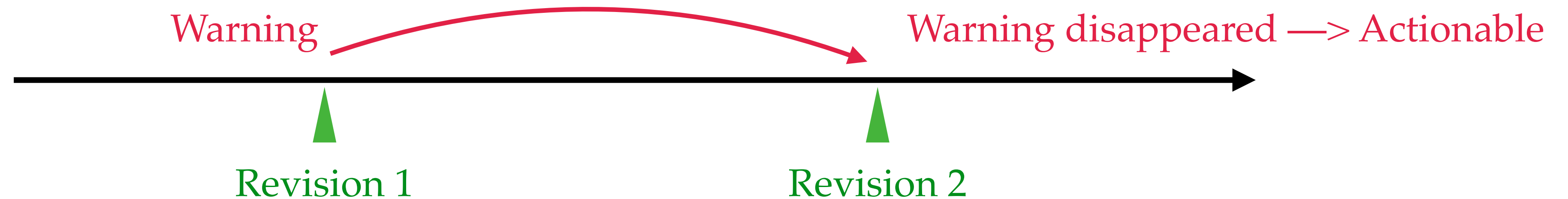
• • • • •



# Detecting False Alarms from Automatic Static Analysis Tools: How Far are We?

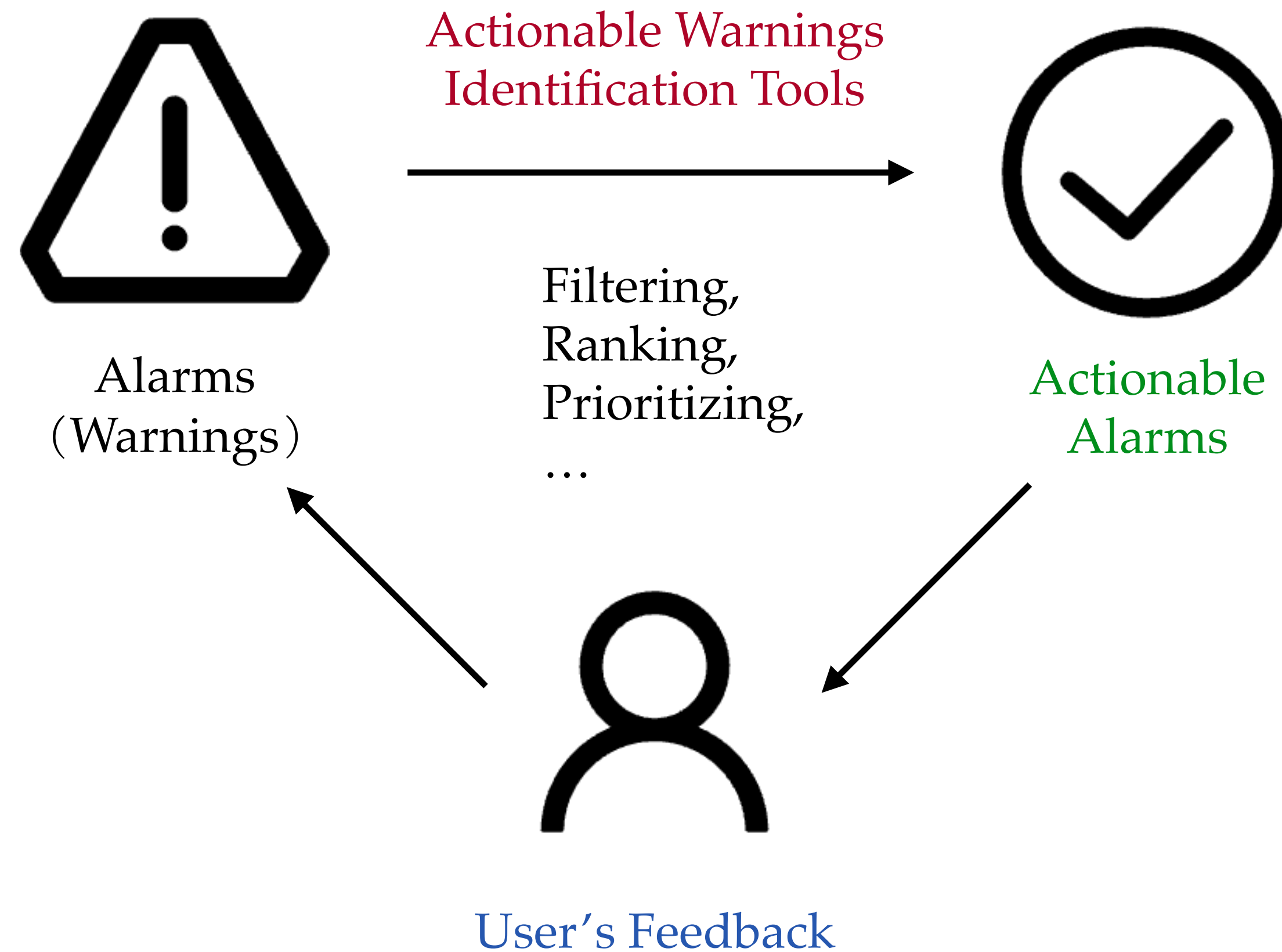
Table 3: Effectiveness of an SVM using the Golden Features after removing the leaked features and removing the duplicate warnings between the training and testing dataset. The numbers in parentheses are the F1 obtained by the baseline classifier that predicts all warnings are actionable.

Project	All Golden Features		- leaked features		- data duplication		- leak, duplication	
	F1	AUC	F1	AUC	F1	AUC	F1	AUC
ant	0.94 (0.09)	1.00	0.11 (0.09)	0.67	-	-	-	-
cassandra	0.92 (0.24)	1.00	0.45 (0.24)	0.86	0.9 (0.41)	0.99	0.29 (0.41)	0.54
commons	0.65 (0.10)	0.99	0.16 (0.10)	0.65	0.75 (0.25)	0.97	0.11 (0.25)	0.49
derby	0.95 (0.09)	1.00	0.39 (0.09)	0.93	0.97 (0.28)	0.97	0.30 (0.28)	0.59
jmeter	0.94 (0.38)	0.99	0.53 (0.38)	0.76	1.00 (0.14)	1.00	0.25 (0.14)	1.00
lucene-solr	0.87 (0.51)	0.97	0.59 (0.51)	0.74	0.87 (0.53)	0.98	0.23 (0.53)	0.62
maven	0.86 (0.07)	1.00	0.27 (0.07)	0.9	0.95 (0.24)	0.99	0.27 (0.24)	0.58
tomcat	0.93 (0.37)	1.00	0.48 (0.37)	0.73	0.95 (0.70)	1.00	0.65 (0.70)	0.39
phoenix	0.89 (0.25)	1.00	0.42 (0.25)	0.78	0.83 (0.37)	0.99	0.40 (0.37)	0.63
Average	0.88 (0.23)	1.00	0.38 (0.23)	0.76	0.90 (0.37)	0.99	0.31 (0.37)	0.59

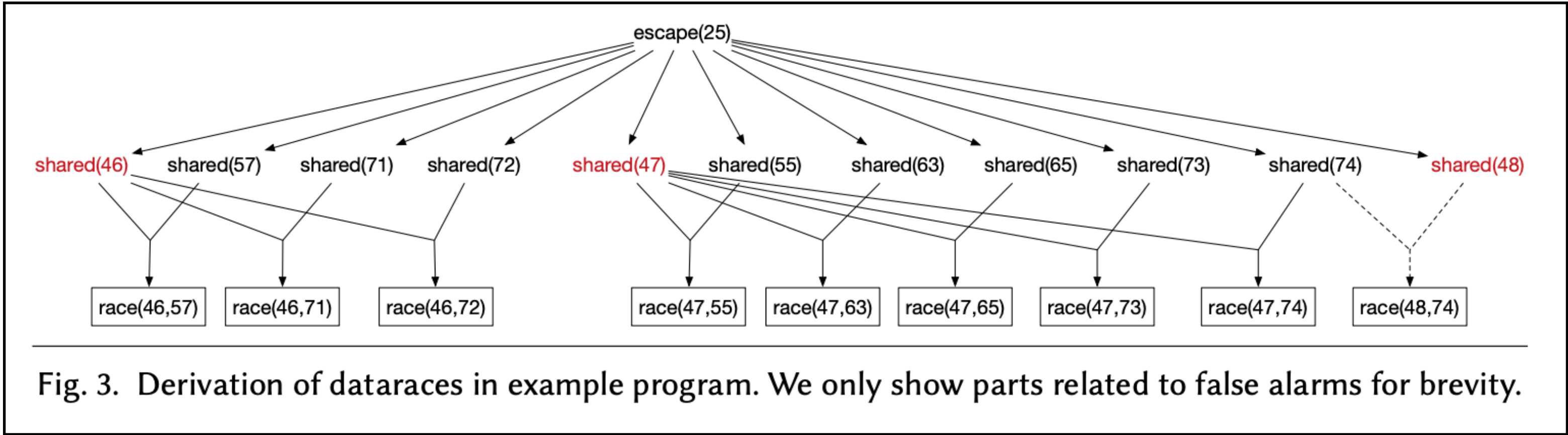


# Programmer: the ~~artificial~~ intelligence

## A Step Backward



# Effective **interactive resolution** of static analysis alarms



Alarms  
(Warnings)

Optimization  
Problem Solver



a Set of Questions

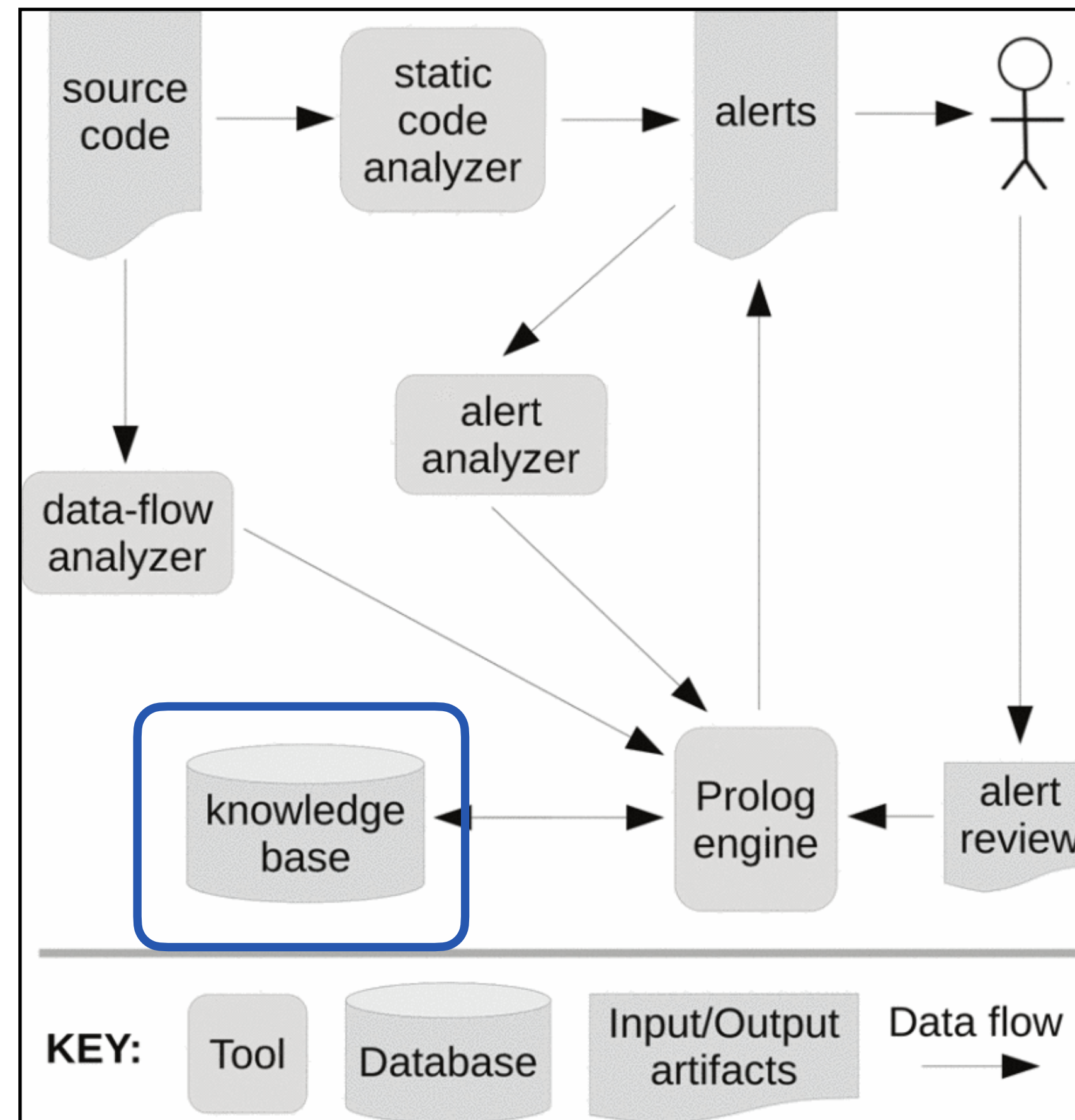
Analysier



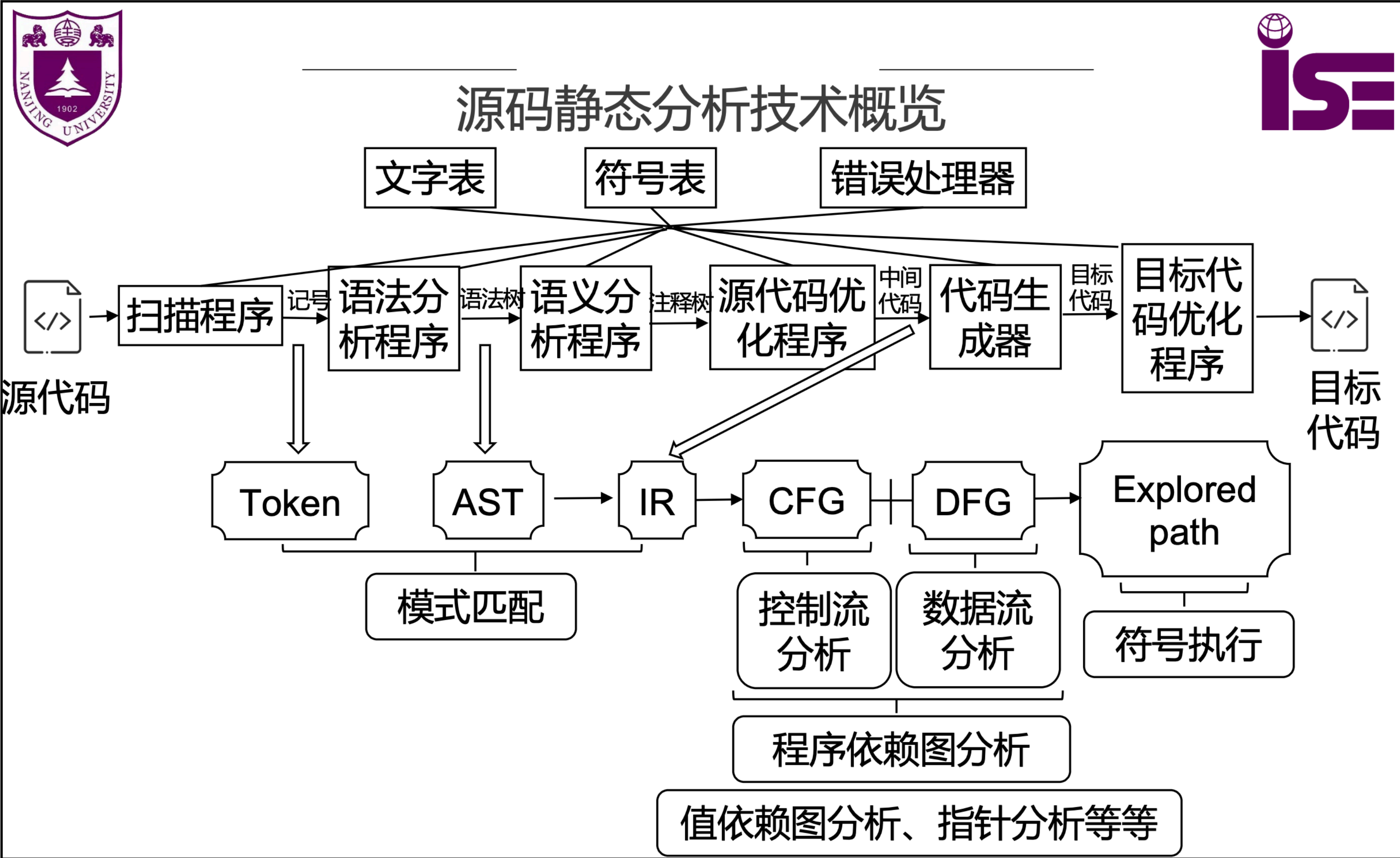
the Most Likely  
Actionable Alarms



# Dynamic Filtering and Prioritization of Static Code Analysis Alerts



# Another Eye on Approaches of Static Analysis



# How Far are We?

## Conclusion

- AWI, a frontier field.
  - open metrics?
  - application in real world?
- The problem itself seems non-trivial...



# Our Group

紫东华策史

- 蔡之恒 (201250127)
- 林浩然 (201250184)
- 沈霁昀 (201250048)
- 熊丘桓 (201250172)

