

Better Automatic Program Repair by Using Bug Reports and Tests Together



Manish Motwani

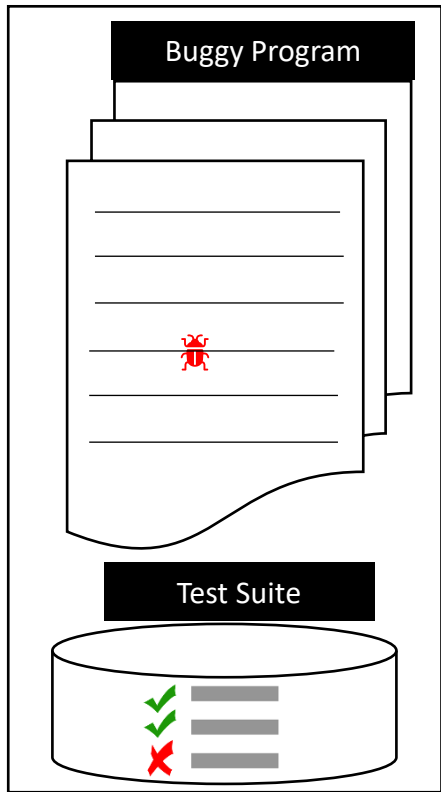
UMassAmherst



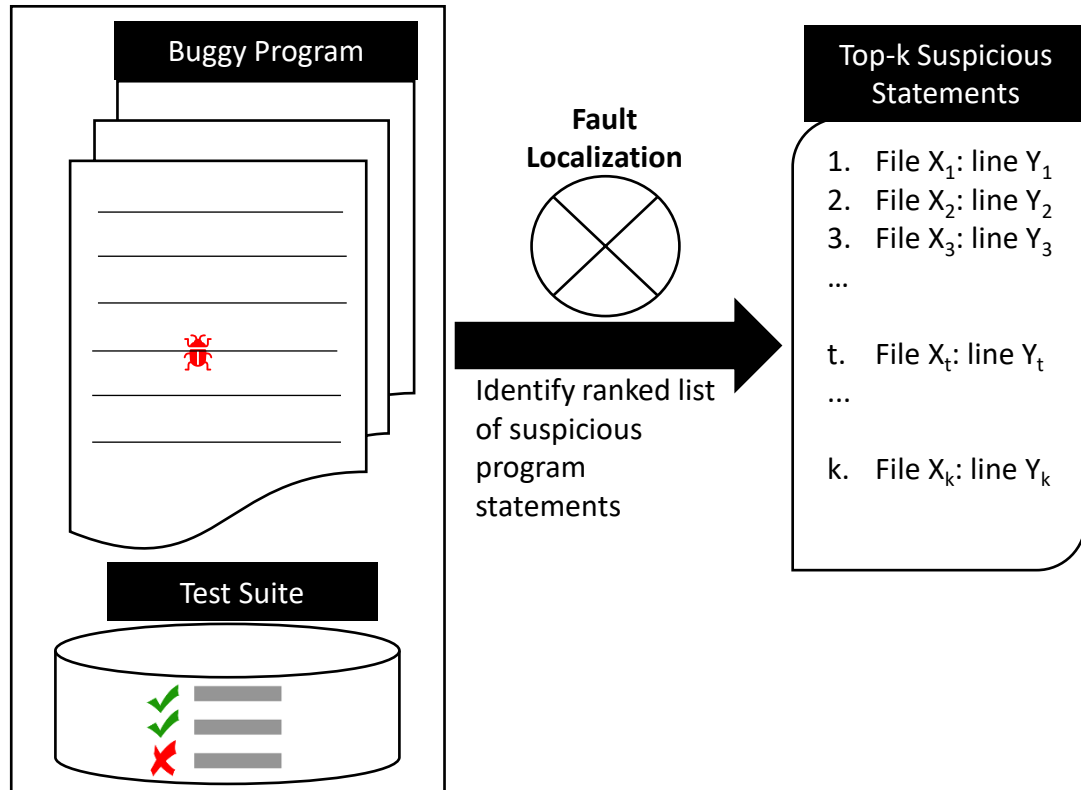
Yuriy Brun

<https://github.com/LASER-UMASS/SBIR-ReplicationPackage>

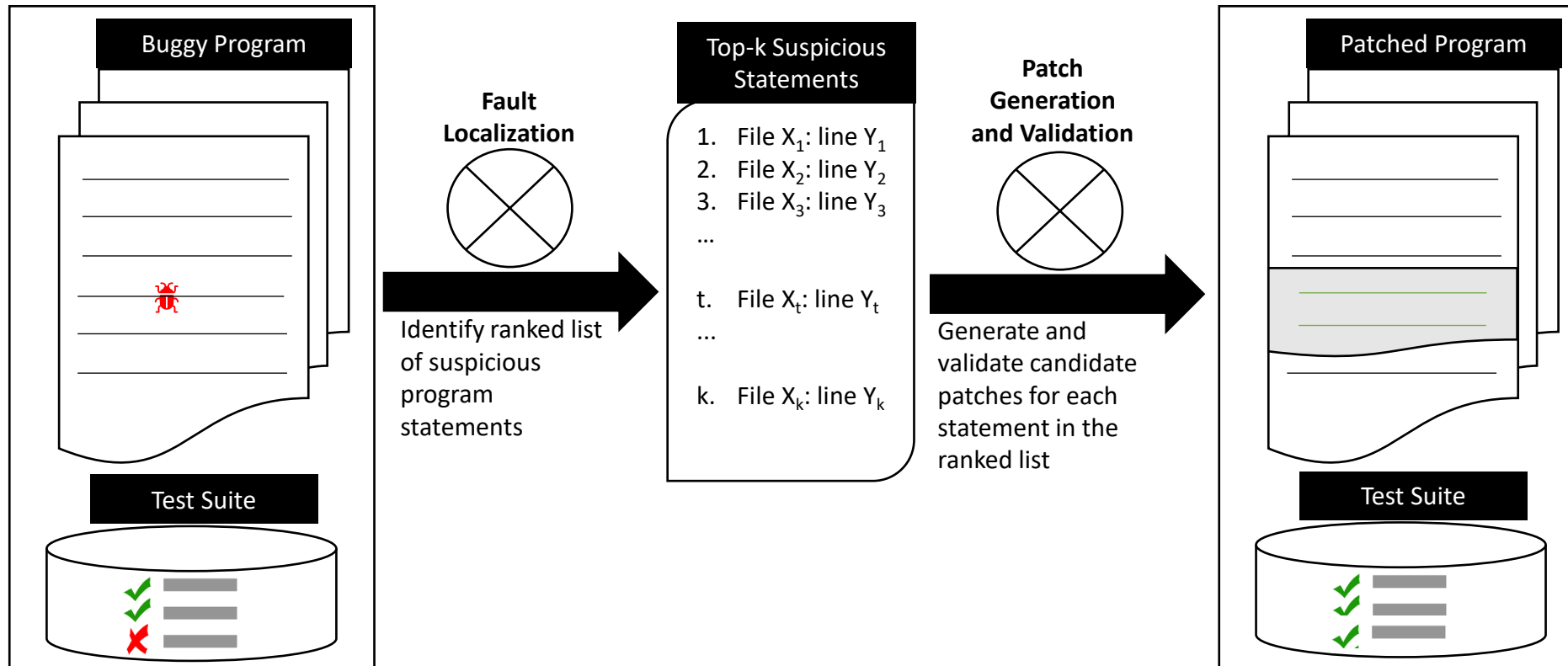
Automatic Program Repair (APR) Process



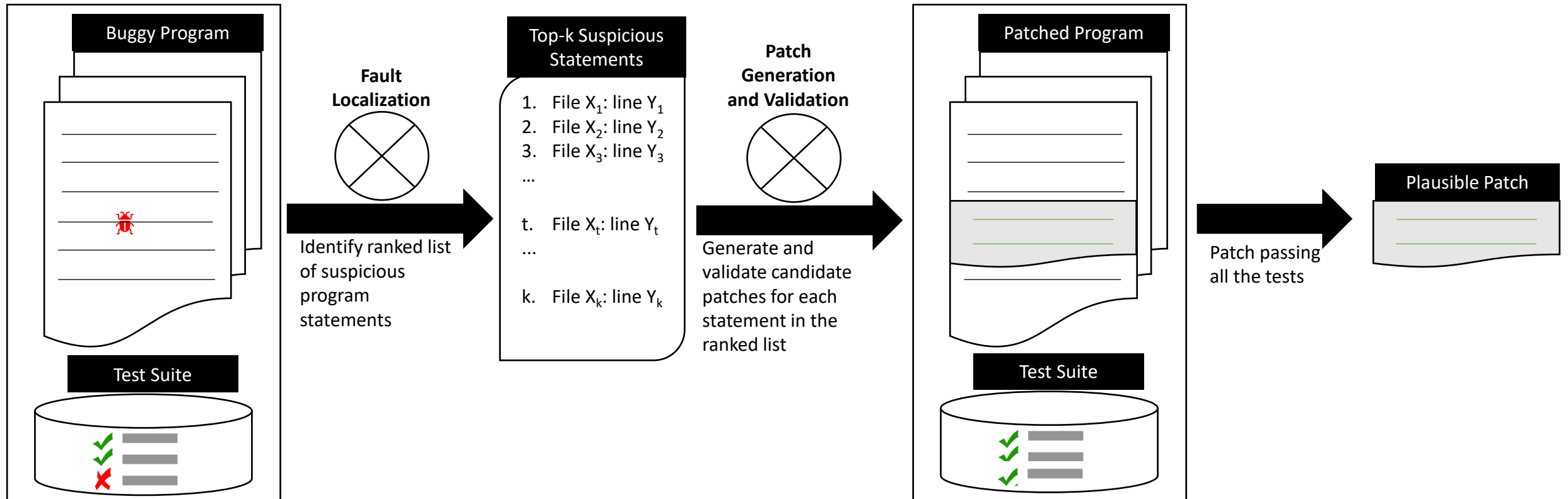
Automatic Program Repair (APR) Process



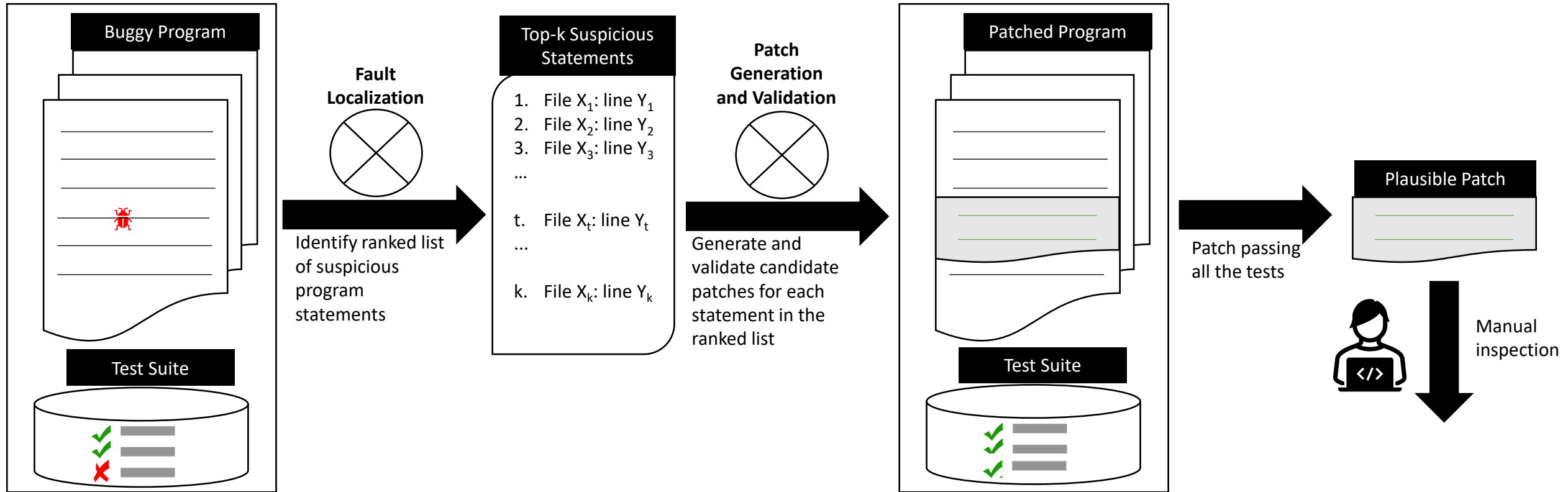
Automatic Program Repair (APR) Process



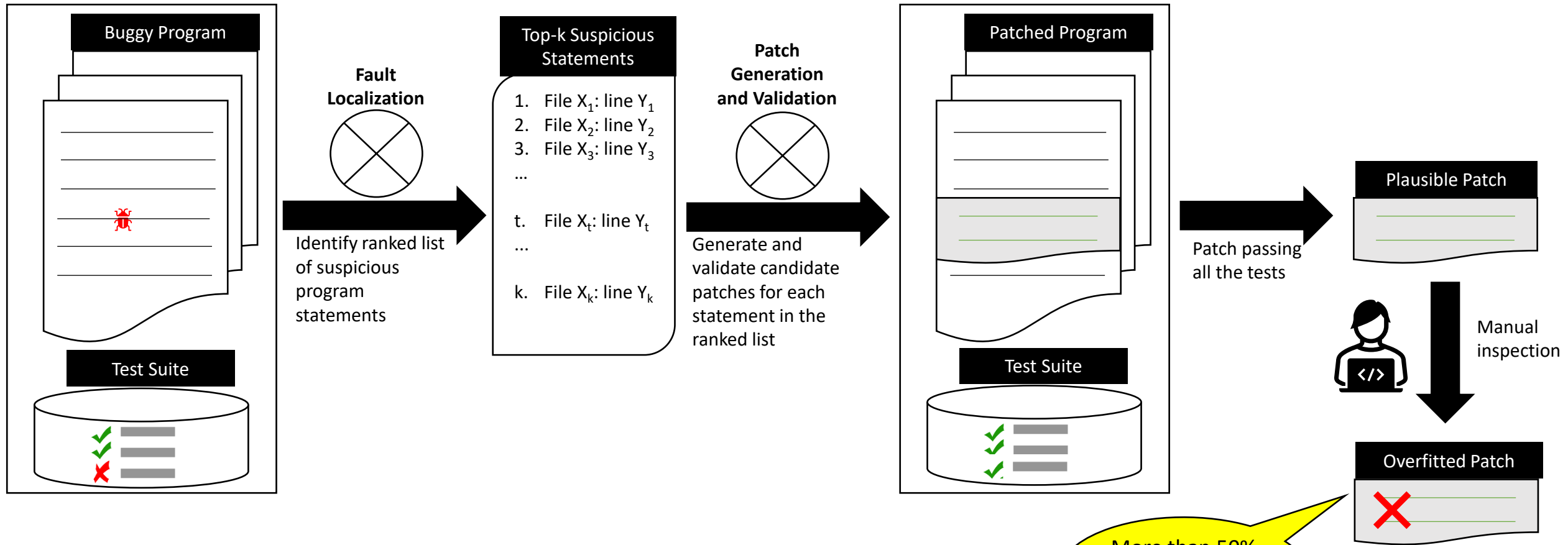
Automatic Program Repair (APR) Process



Repair Techniques Struggle to Patch Defects Correctly

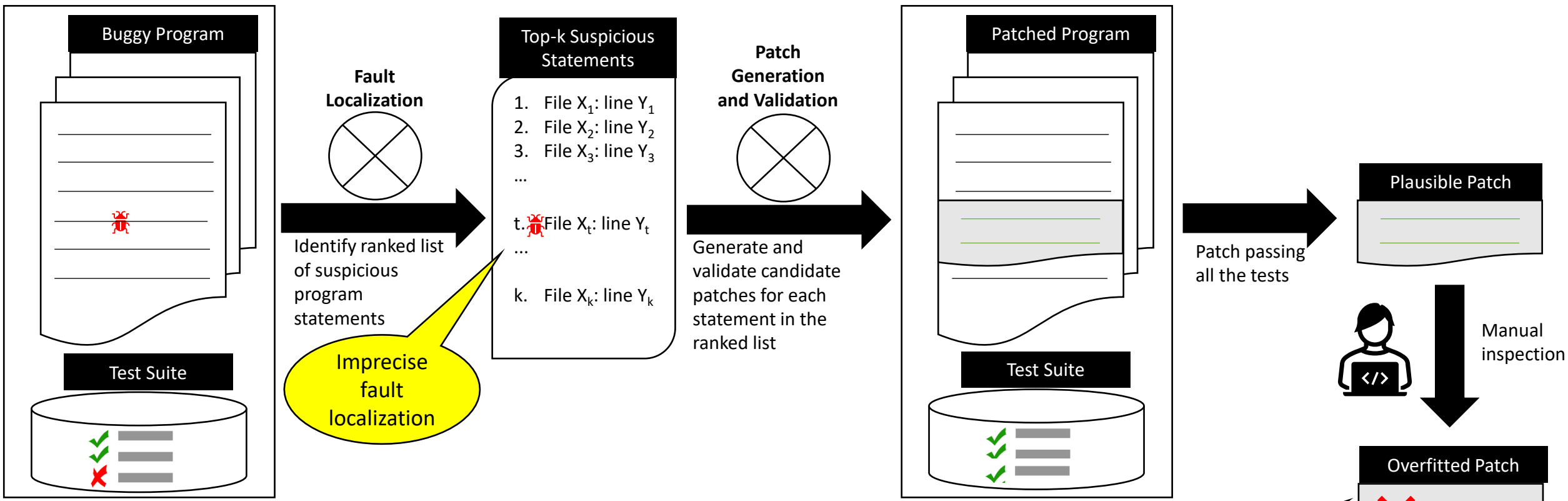


Repair Techniques Struggle to Patch Defects Correctly



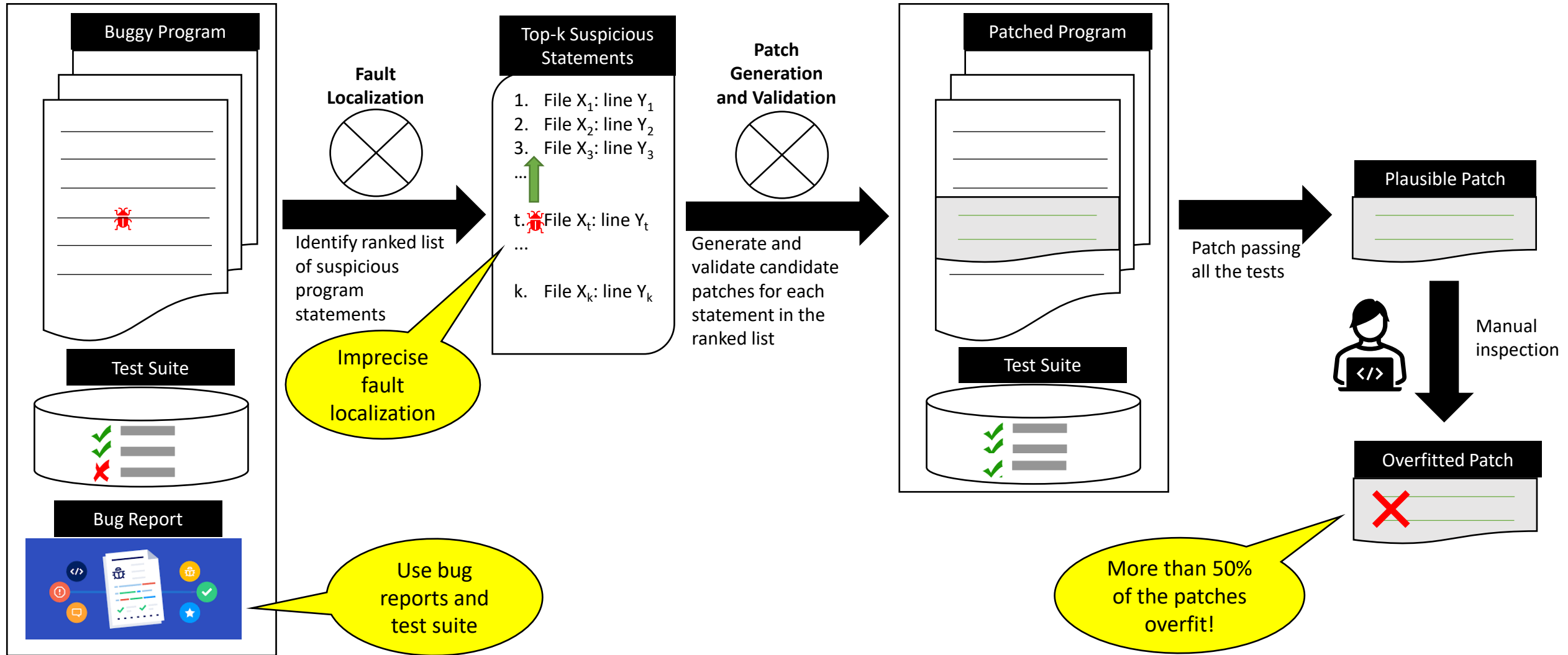
- Smith et al., Is the cure worse than the disease? Overfitting in automated program repair, ESEC/FSE, 2015
Long et al., An analysis of patch plausibility and correctness for generate-and-validate patch generation systems, ISSTA, 2015
Long et al., An analysis of the search spaces for generate and validate patch generation systems, ICSE, 2016
Le et al., Overfitting in semantics-based automated program repair, ICSE, 2018
Motwani et al. Quality of automated program repair on real-world defects, TSE, 2022
Alarcon et al., Would you fix this code for me? effects of repair source and commenting on trust in code repair, Systems, 2020

The Localization Error Problem in Automatic Program Repair

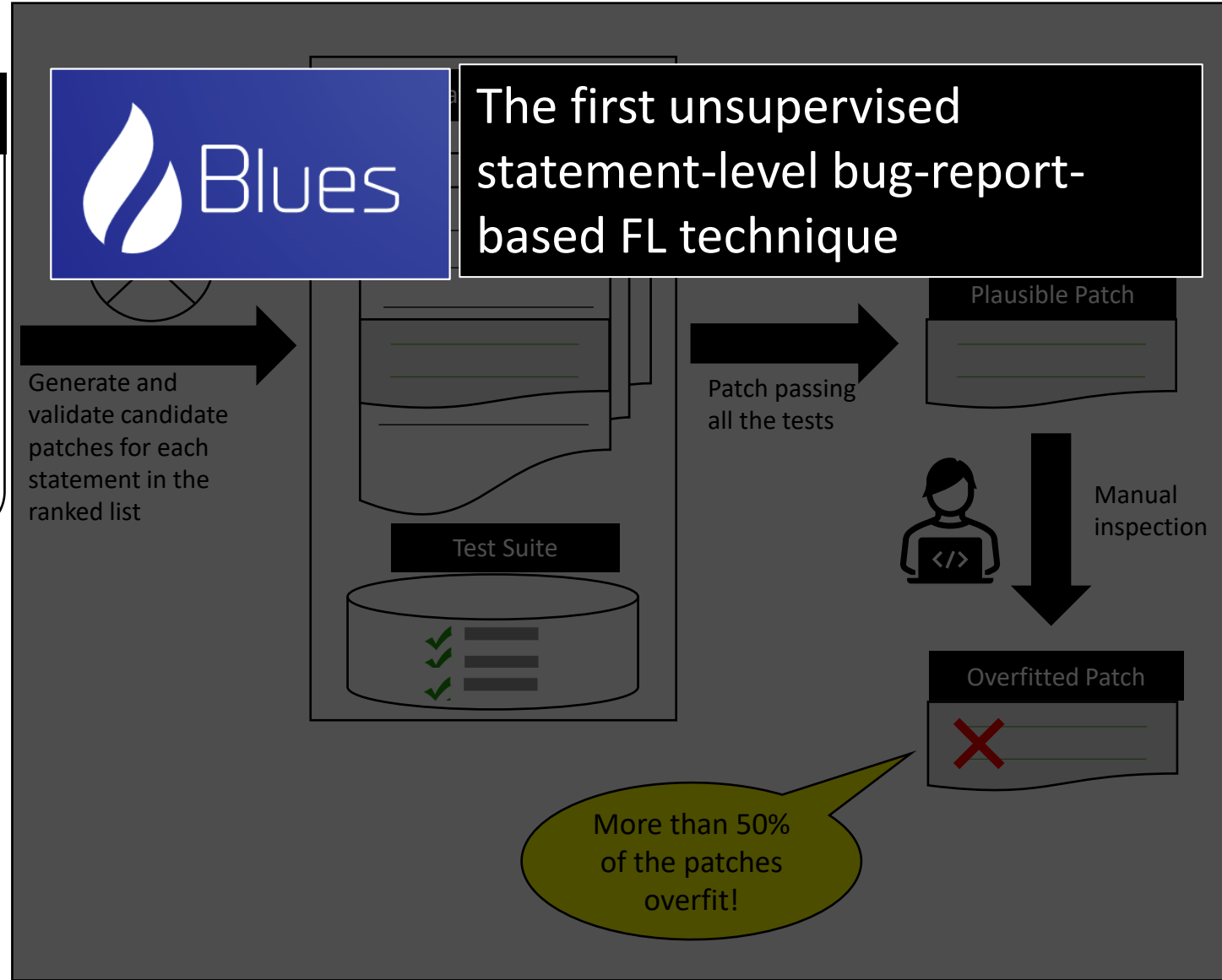
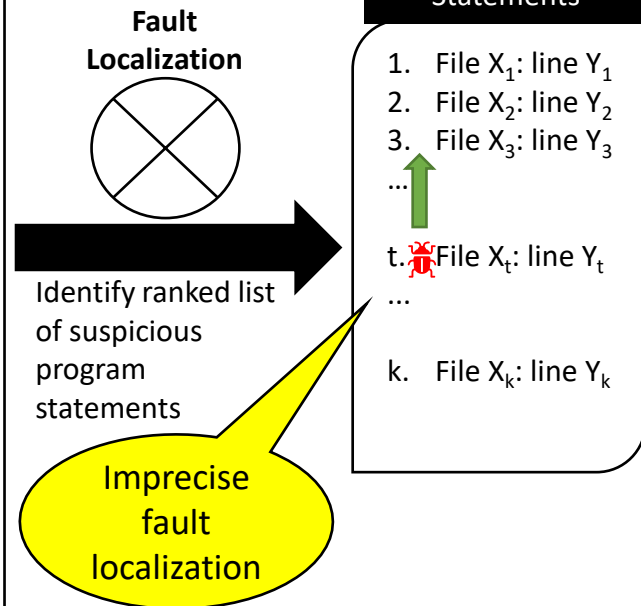
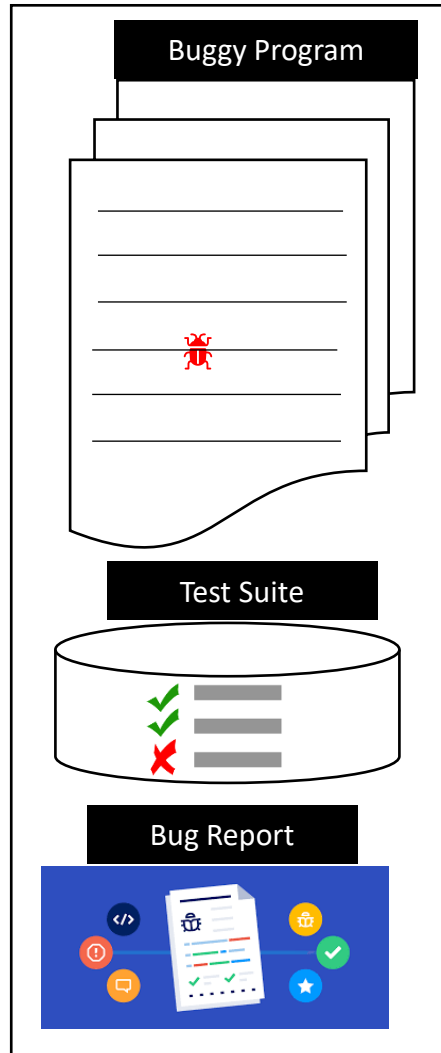


Afzal et al., SOSRepair: Expressive semantic search for real-world program repair, TSE, 2021
 Yang et al., Evaluating the strategies of statement selection in automated program repair, ISSTA, 2018
 Wen et al., An empirical analysis of the influence of fault space on search-based automated program repair, CoRR, 2017
 Bieman et al., Fault localization for automated program repair: Effectiveness, performance, repair correctness, Software Quality Journal, 2017
 Jiang et al., A manual inspection of defects4j bugs and its implications for automatic program repair, Science China Information Sciences, 2019
 Liu et al., You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems, ICST, 2019

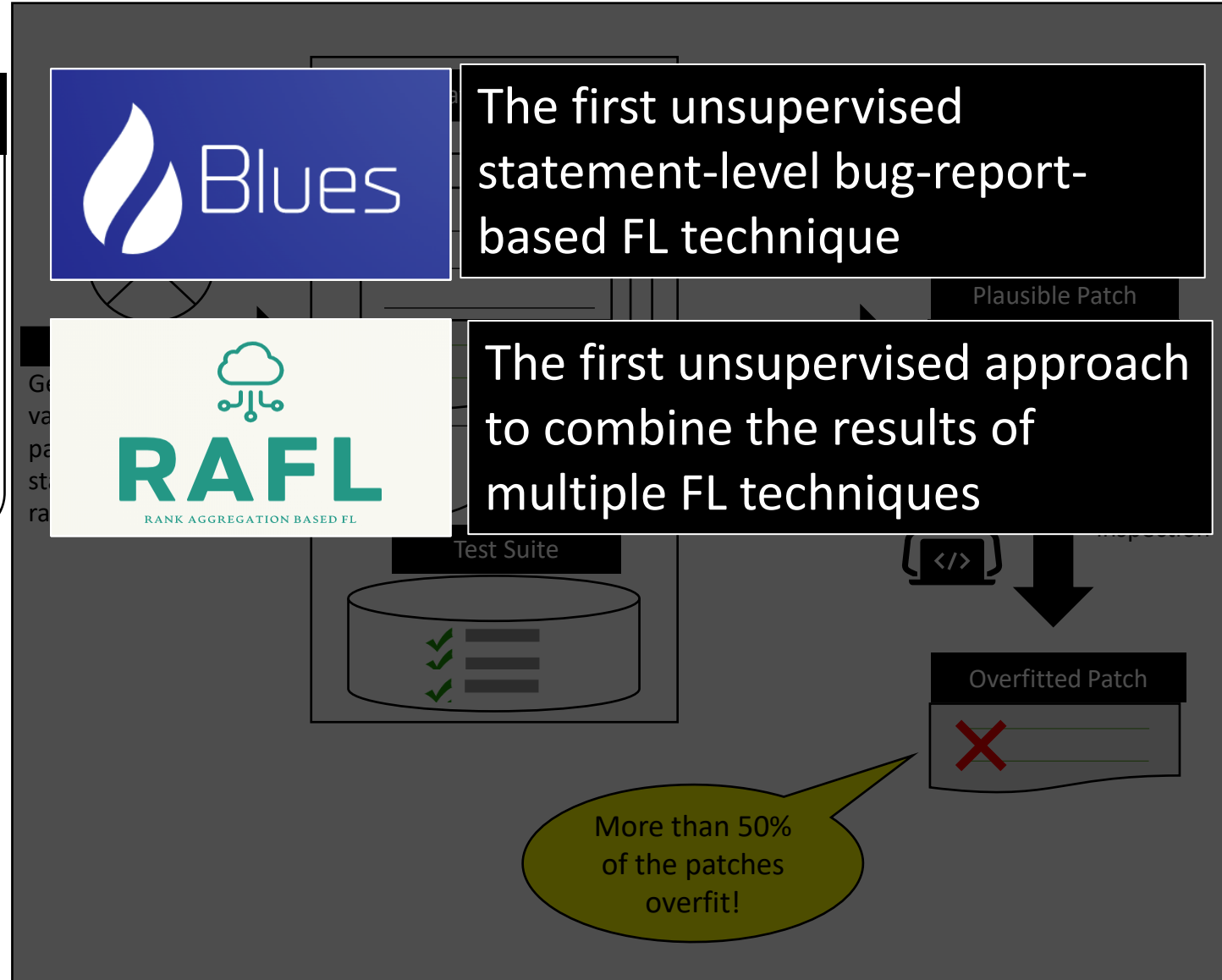
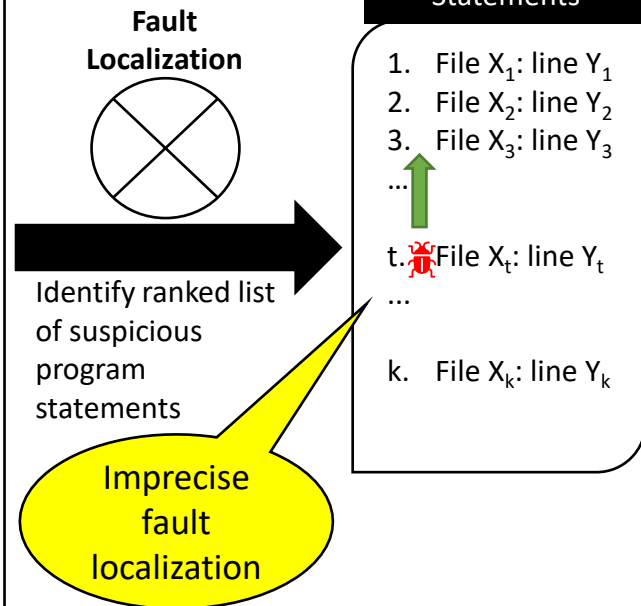
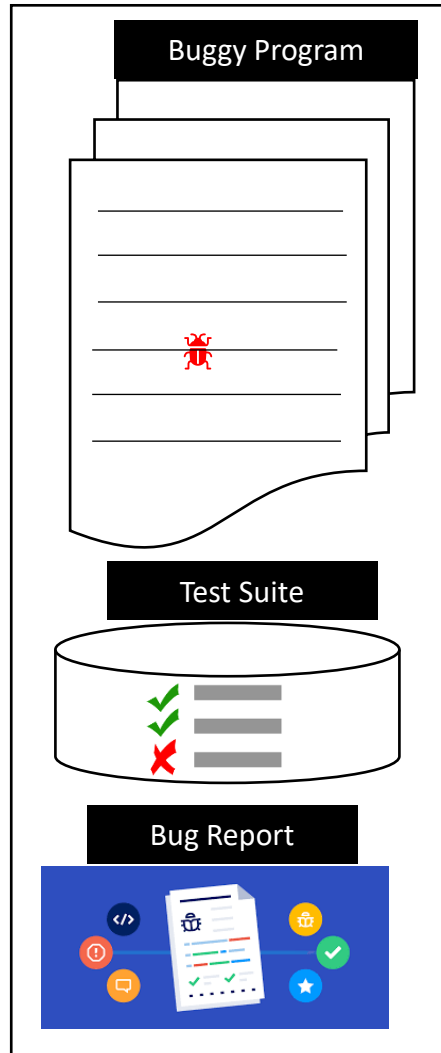
Key Insight



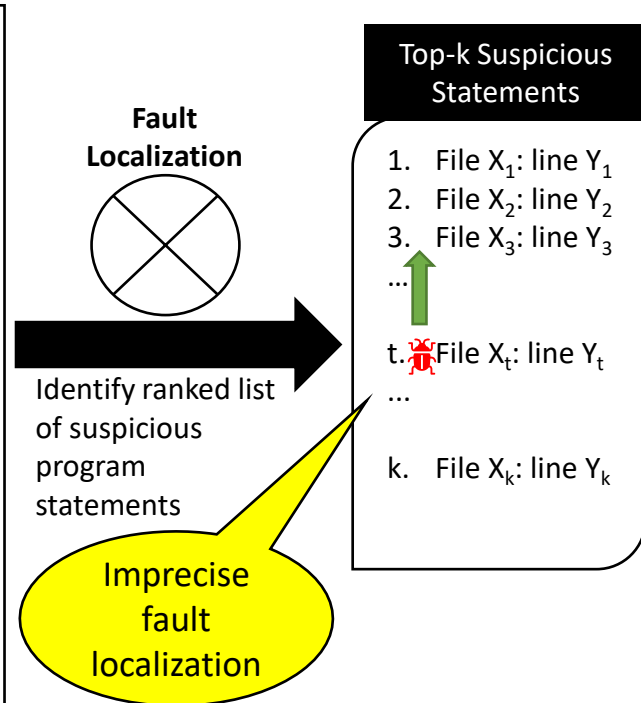
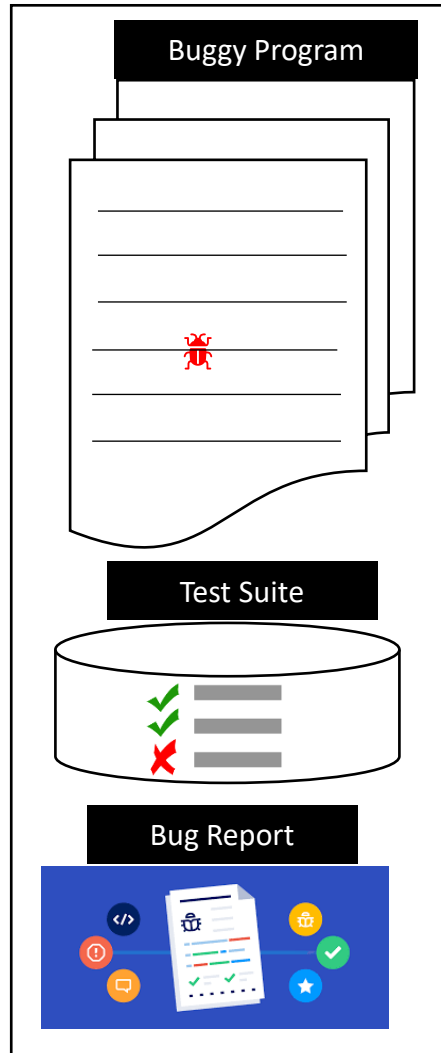
Our Solution



Our Solution



Our Solution



This section features a collage of three logos: Blues (a blue flame icon), RAFL (a green cloud and network icon with the text 'RANK AGGREGATION BASED FL'), and SBIR (an orange icon of a line graph and database cylinders). Each logo is accompanied by a text box describing its contribution to the solution. The Blues box states it is the first unsupervised statement-level bug-report-based FL technique. The RAFL box states it is the first unsupervised approach to combine the results of multiple FL techniques. The SBIR box states it combines Blues with an existing SBFL technique using the RAFL approach. A green callout bubble at the bottom right notes that more than 50% of the patches overfit.

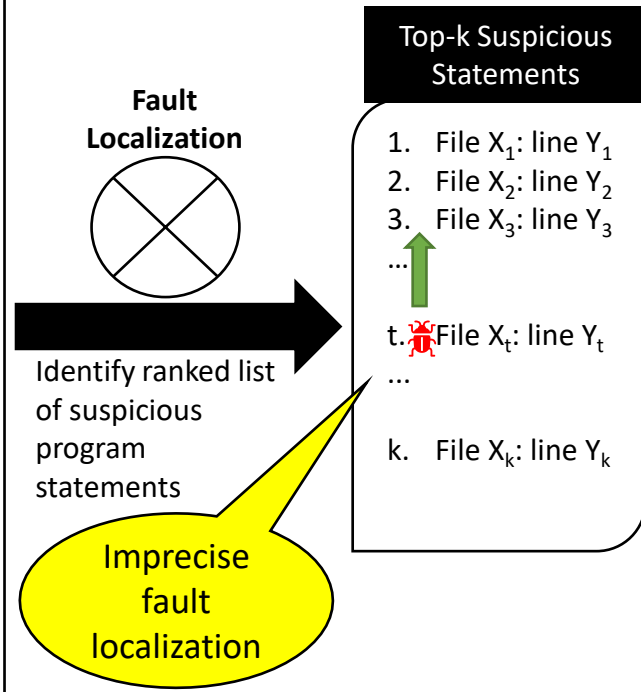
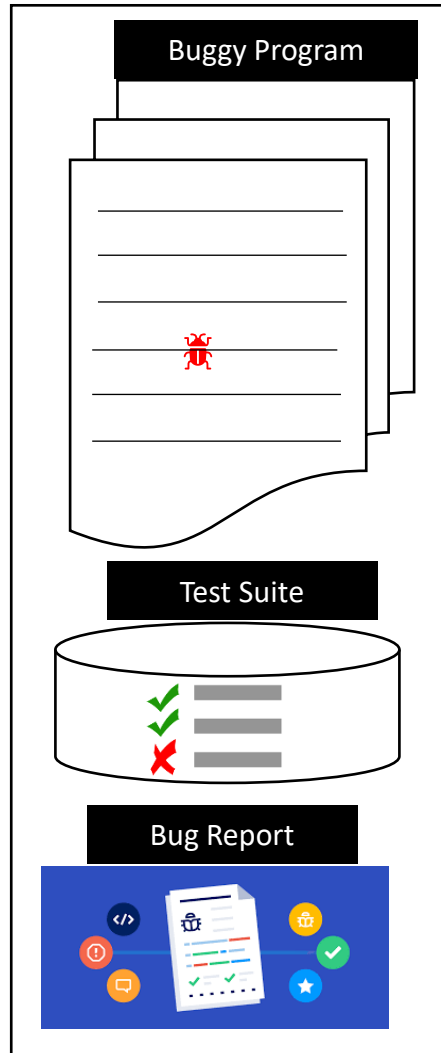
Blues
The first unsupervised statement-level bug-report-based FL technique

RAFL
RANK AGGREGATION BASED FL
The first unsupervised approach to combine the results of multiple FL techniques

SBIR
Combining Blues with an existing SBFL technique using RAFL approach

More than 50% of the patches overfit!

Our Solution

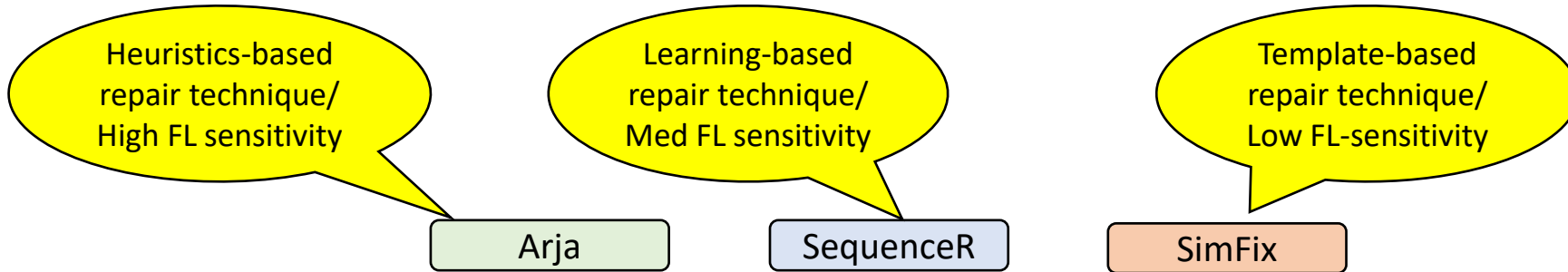


This section is a collage of software artifacts and their descriptions. It features the following items:

- Blues**: The first unsupervised statement-level bug-report-based FL technique.
- RAFL** (RANK AGGREGATION BASED FL): The first unsupervised approach to combine the results of multiple FL techniques.
- SBIR**: Combining Blues with an existing SBFL technique using RAFL approach.
- Plausible Patch**: First investigation of simultaneously using multiple software artifacts to improve program repair.

Effect of Using SBIR on Repair Performance

395 defects from 6 real-world Java projects
available in Defects4J version 1.x



Effect of Using SBIR on Repair Performance

395 defects from 6 real-world Java projects available in Defects4J version 1.x

Heuristics-based repair technique/
High FL sensitivity

Learning-based repair technique/
Med FL sensitivity

Template-based repair technique/
Low FL-sensitivity

Arja

SequenceR

SimFix

Chart-12
Closure-78
Closure-86
Lang-10
Lang-20

Closure-86

Closure-68
Closure-92

Using SBIR enables APR tools to **correctly repair** 7 new defects that **14 existing techniques¹** couldn't fix

Effect of Using SBIR on Repair Performance

395 defects from 6 real-world Java projects available in Defects4J version 1.x

Heuristics-based repair technique/
High FL sensitivity

Learning-based repair technique/
Med FL sensitivity

Template-based repair technique/
Low FL-sensitivity

Arja

SequenceR

SimFix

Using SBIR enables APR tools to **correctly repair** 7 new defects that **14 existing techniques**¹ couldn't fix

Using SBIR enables APR tools to **correctly repair** 7 new defects that **they** couldn't fix earlier

Chart-12
Closure-78
Closure-86
Lang-10
Lang-20

Closure-86

Closure-68
Closure-92

Lang-7
Lang-10
Lang-59
Math-35

(Uses
Perfect FL)

Closure-68
Closure-92
Closure-126

Effect of Using SBIR on Repair Performance

395 defects from 6 real-world Java projects available in Defects4J version 1.x

SBIR enables repair tools to correctly patch many new defects without modifying their patch generation algorithms.

Arja

Chart-12
Closure-78
Closure-86
Lang-10
Lang-20

SequenceR

Closure-86

SimFix

Closure-68
Closure-92

Using SBIR enables APR tools to **correctly repair 7 new defects** that **14 existing techniques¹** couldn't fix

Using SBIR enables APR tools to **correctly repair 7 new defects** that **they** couldn't fix earlier.

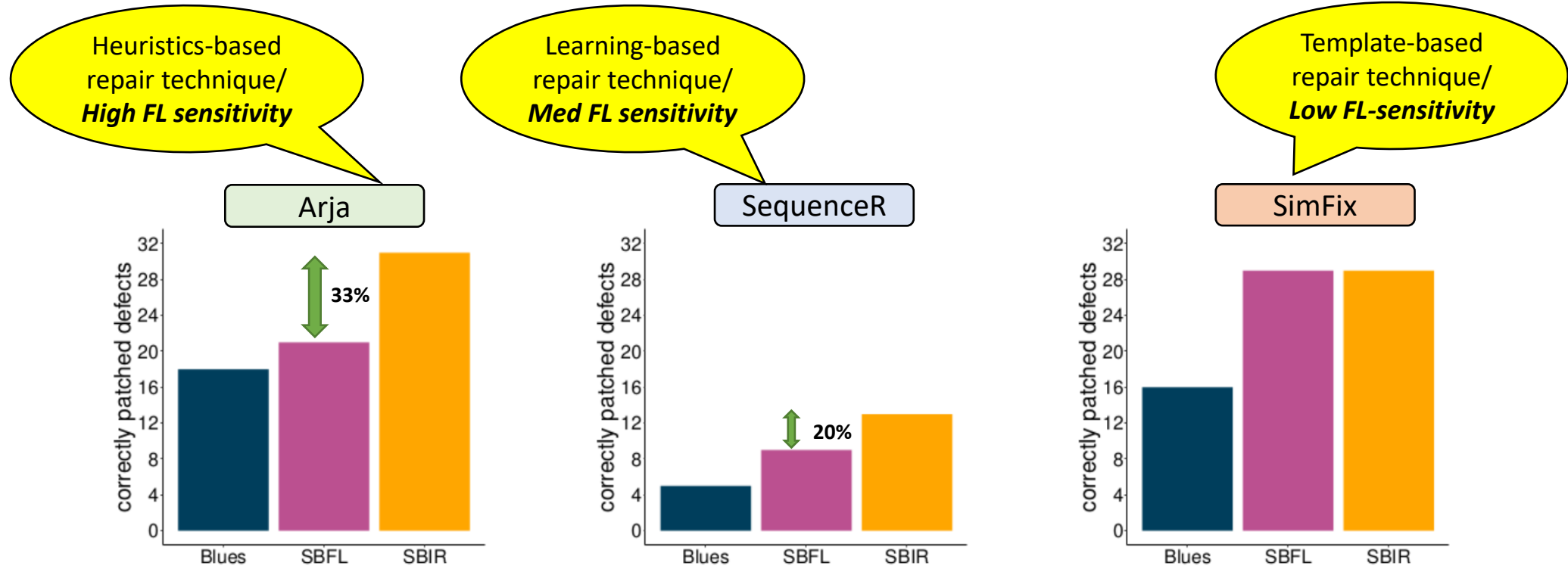
Lang-7
Lang-10
Lang-59
Math-35

(Uses Perfect FL)

Closure-68
Closure-92
Closure-126

Effect of Using SBIR on Repair Quality

689 single file edit defects from 17 real-world Java projects available in Defects4J v2.0



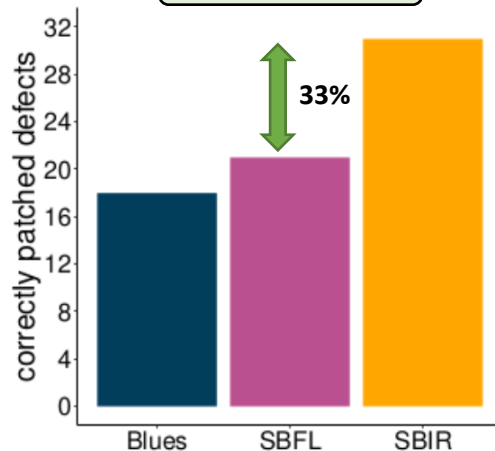
Effect of Using SBIR on Repair Quality

689 single file edit defects from 17 real-world Java projects available in Defects4J v2.0

SBIR significantly improves the quality of more FL-sensitive APR tools.

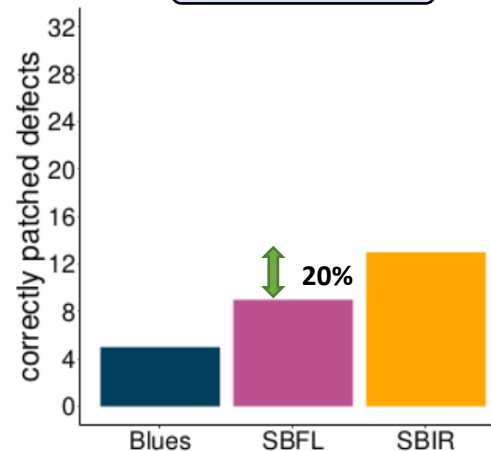
High FL sensitivity

Arja



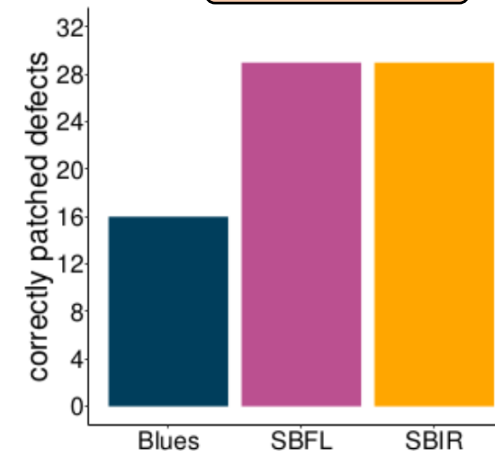
Med FL sensitivity

SequenceR



Low FL-sensitivity

SimFix



Effect of Using SBIR on Localization Error

689 single file edit defects from 17 real-world Java projects available in Defects4J v2.0

	Arja	SequenceR	SimFix
localization error assessment			
upper bound	36	24	32
↓ # of defects not correctly patched due to localization error ↓			↓ Lower is better
SBFL	15	14	2
Blues	21	20	19
SBIR	8	12	2

Effect of Using SBIR on Localization Error

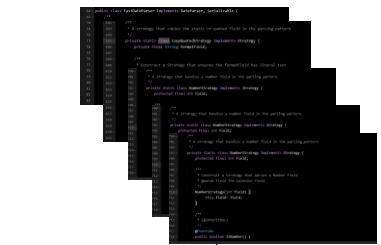
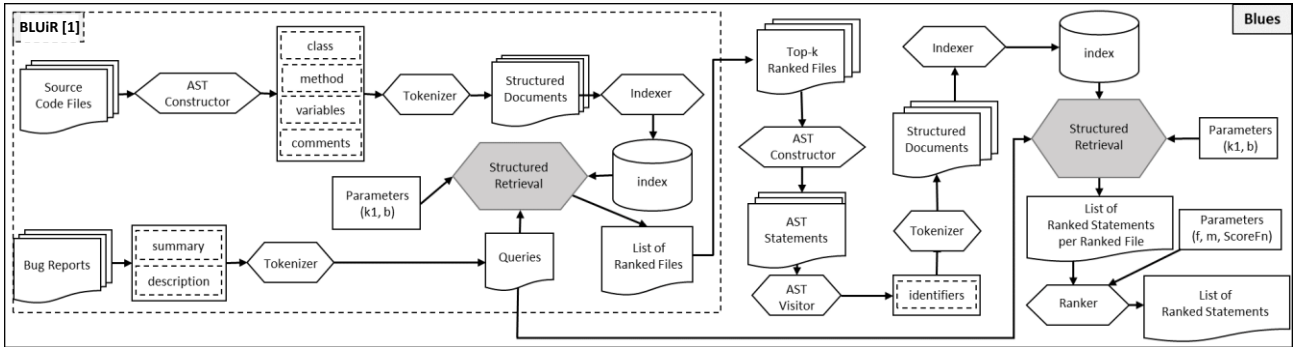
689 single file edit defects from 17 real-world Java projects available in Defects4J v2.0

	Arja	SequenceR	SimFix
localization error assessment			
upper bound	36	24	32
↓ # of defects not correctly patched due to localization error ↓			↓ Lower is better
SBFL	15	14	2
Blues	21	20	19
SBIR	8	12	2

SBIR lowers the localization error by providing repair tools an earlier opportunity to patch actual buggy statements.

Blues: Unsupervised, Statement-Level FL Using Bug Reports

Bug report



Source files



```
Lang_10 Q0 FastDateParser.java-307-266 1 0.916951 indri
Lang_10 Q0 FastDateParser.java-308-269 2 0.914638 indri
Lang_10 Q0 FastDateParser.java-307-268 3 0.897916 indri
Lang_10 Q0 FastDateParser.java-308-270 4 0.782011 indri
Lang_10 Q0 FastDateParser.java-314-280 5 0.757878 indri
Lang_10 Q0 FastDateParser.java-314-279 6 0.757878 indri
Lang_10 Q0 FastDateParser.java-309-273 7 0.757878 indri
Lang_10 Q0 FastDateParser.java-309-272 8 0.757878 indri
Lang_10 Q0 FastDateParser.java-304-253 9 0.757878 indri
Lang_10 Q0 FastDateParser.java-303-251 10 0.750024 indri
```

ranked list of suspicious program statements

[1] Saha et al., Improving bug localization using structured information retrieval, ASE 2013

Step 1: Extract terms associated with *Summary* and *Description* from bug report

Commons Lang / LANG-831

FastDateParser does not handle white-space properly

summary

description

bug report

Details

Type:	Bug	Status:	CLOSED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	3.2
Component/s:	None		
Labels:	None		

Description

The SimpleDateFormat Javadoc does not treat white-space specially, however FastDateParser treats a single white-space as being any number of white-space characters.

This means that FDP will parse dates that fail when parsed by SDP.

Activity

All [Comments](#) [Work Log](#) [History](#) [Activity](#) [Transitions](#)

Sebb added a comment - 27/Sep/12 00:10 URL: <http://svn.apache.org/viewvc?rev=1390779&view=rev> Log: LANG...

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <bugrepository name="Defects4J">
3   <bug id="Lang_10" link="https://issues.apache.org/jira/browse/LANG-831">
4     <buginformation>
5       <summary>fast date parser does not handle white space
6         properly</summary>
7       <description>the SimpleDateFormat javadoc does not treat
8         white space specially however fastdateparser treats
9         single white space being any number white space
10        characters this means that fdp will parse dates that
11        fail when parsed sdp</description>
12     </buginformation>
13     <fixedfiles>
14       <file>org.apache.commons.lang3.time.FastDateParser</file>
15     </fixedfiles>
16   </bug>
17 </bugrepository>
```

Bug report query

Step 2: Extract terms associated with *Class*, *Method*, *Identifier* and *Comment* from source files

```
68 - public class FastDateParser implements DateParser, Serializable {
68 - public class FastDateParser implements DateParser, Serializable {
68 - public class FastDateParser implements DateParser, Serializable {
69 - /**
70 -  * Required for serialization support.
71 -  *
72 -  * @see java.io.Serializable
73 -  */
74 - private static final long serialVersionUID = 1L;
75 -
76 - private static final ConcurrentMap<Locale, TimeZoneStrategy> tzsCache=
77 -     new ConcurrentHashMap<Locale, TimeZoneStrategy>(3);
78 -
79 - static final Locale JAPANESE_IMPERIAL = new Locale("ja", "JP", "JP");
80 -
81 - // defining fields
82 - private final String pattern;
83 - private final TimeZone timeZone;
```

Source files

class

method

identifier

comment

```
1 <DOC>
2 1 <DOC>
3 2 1 <DOC>
4 3 2 1 <DOC>
5 4 3 2 1 <DOC>
6 5 4 3 2 <DOCNO>org.apache.commons.lang3.time.FastDateParser.java </DOCNO>
7 6 5 4 3 <text>
8 7 6 5 4 <class>
9 8 7 6 fast date parser, key value, strategy, copy quoted strategy, text
10 9 8 7 strategy, number strategy, time zone strategy
11 10 9 8 </class>
12 11 10 9 <method>
13 12 11 10 fast date parser, init, get pattern, get time zone, get locale,
14 13 12 get parse pattern,
15 14 13 ...
16 15 14 </method>
17 16 15 14 <identifier>
18 17 16 15 fast date parser, date parser, serializable, serial version uid,
19 18 17 16 concurrent map,
20 19 18 17 locale, time zone strategy, tzs cache, concurrent hash map,
21 20 19 18 locale, time zone strategy,
22 21 20 19 locale, japanese imperial, locale, string, pattern, time zone
23 22 21 20 time zone,
24 23 22 21 ...
25 24 23 22 </identifier>
26 25 24 23 <comments>
27 26 25 24 give, access, generated, pattern, for, test, code,
28 27 26 basics, compare, another, object, for, equality, with,
29 28 27 this, object, param,
30 29 28 obj, the, object, compare, return, code, true, code,
31 30 29 equal, this, instance,
32 31 30 ...
33 32 31 </comments>
34 33 32 </text>
35 34 33 </DOC>
```

Source file document collection

Step 3: Execute IR model to rank suspicious files for given query

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <bugrepository name="Defects4J">
3   <bug id="Lang_10" link="https://issues.apache.org/jira/browse/LANG-831">
4     <buginformation>
5       <summary>fast date parser does not handle white space
6         properly</summary>
7       <description>the simpledateformat javadoc does not treat
8         white space specially however fastdateparser treats
9         single white space being any number white space
10        characters this means that fdp will parse dates that
11        fail when parsed sdp</description>
12     </buginformation>
13     <fixedfiles>
14       <file>org.apache.commons.lang3.time.FastDateParser</file>
15     </fixedfiles>
16   </bug>
17 </bugrepository>
```

summary

description

Bug report query

Execute IR model eight times

class

method

identifier

comment

```
1 <DOC>
2 <DOCNO>org.apache.commons.lang3.time.FastDateParser.java </DOCNO>
3 </DOC>
4 2 <DOCNO>org.apache.commons.lang3.time.FastDateParser.java </DOCNO>
5 3 1 <DOC>
6 4 2 <DOCNO>org.apache.commons.lang3.time.FastDateParser.java </DOCNO>
7 5 3 1 <DOC>
8 6 4 2 <DOCNO>org.apache.commons.lang3.time.FastDateParser.java </DOCNO>
9 7 5 3 <text>
10 8 6 4 <class>
11 9 7 5 3 <text>
12 10 8 6 <text>
13 11 9 7 </class>
14 12 10 8 <method>
15 13 11 9 <text>
16 14 12 10 </method>
17 15 13 11 <identifier>
18 16 14 12 <text>
19 17 15 13 </identifier>
20 18 16 14 <comments>
21 19 17 15 <text>
22 20 18 16 <text>
23 21 19 17 <text>
24 22 20 18 </comments>
25 23 21 19 <text>
26 24 22 20 </text>
27 25 23 21 </text>
28 26 24 22 </text>
29 27 25 23 </text>
30 28 26 24 </text>
31 29 27 25 </text>
32 30 28 26 </text>
33 31 29 </text>
34 30 28 </text>
35 31 29 </text>
36 30 28 </text>
37 31 29 </text>
38 30 28 </text>
39 31 29 </text>
40 30 28 </text>
41 31 29 </text>
42 30 28 </text>
43 31 29 </text>
44 30 28 </text>
45 31 29 </text>
46 30 28 </text>
47 31 29 </text>
48 30 28 </text>
49 31 29 </text>
50 30 28 </text>
51 31 29 </text>
52 30 28 </text>
53 31 29 </text>
54 30 28 </text>
55 31 29 </text>
56 30 28 </text>
57 31 29 </text>
58 30 28 </text>
59 31 29 </text>
60 30 28 </text>
61 31 29 </text>
62 30 28 </text>
63 31 29 </text>
64 30 28 </text>
65 31 29 </text>
66 30 28 </text>
67 31 29 </text>
68 30 28 </text>
69 31 29 </text>
70 30 28 </text>
71 31 29 </text>
72 30 28 </text>
73 31 29 </text>
74 30 28 </text>
75 31 29 </text>
76 30 28 </text>
77 31 29 </text>
78 30 28 </text>
79 31 29 </text>
80 30 28 </text>
81 31 29 </text>
82 30 28 </text>
83 31 29 </text>
84 30 28 </text>
85 31 29 </text>
86 30 28 </text>
87 31 29 </text>
88 30 28 </text>
89 31 29 </text>
90 30 28 </text>
91 31 29 </text>
92 30 28 </text>
93 31 29 </text>
94 30 28 </text>
95 31 29 </text>
96 30 28 </text>
97 31 29 </text>
98 30 28 </text>
99 31 29 </text>
100 30 28 </text>
101 31 29 </text>
102 30 28 </text>
103 31 29 </text>
104 30 28 </text>
105 31 29 </text>
106 30 28 </text>
107 31 29 </text>
108 30 28 </text>
109 31 29 </text>
110 30 28 </text>
111 31 29 </text>
112 30 28 </text>
113 31 29 </text>
114 30 28 </text>
115 31 29 </text>
116 30 28 </text>
117 31 29 </text>
118 30 28 </text>
119 31 29 </text>
120 30 28 </text>
121 31 29 </text>
122 30 28 </text>
123 31 29 </text>
124 30 28 </text>
125 31 29 </text>
126 30 28 </text>
127 31 29 </text>
128 30 28 </text>
129 31 29 </text>
130 30 28 </text>
131 31 29 </text>
```

BM25 Okapi

Source file document collection

$$\sum_{r \in Q} \sum_{f \in D} S(d_f, q_r)$$


r : query representation
 f : document field

Step 3: Execute IR model to rank suspicious files for given query



Step 4: Extract 57 possible AST nodes from source files and create a *statement* document collection


```
Lang_10 Q0 org.apache.commons.lang3.time.FastDateParser.java 1 0.403753 indri
Lang_10 Q0 org.apache.commons.lang3.time.FastDateFormat.java 2 0.225249 indri
Lang_10 Q0 org.apache.commons.lang3.time.FastDatePrinter.java 3 0.217153 indri
Lang_10 Q0 org.apache.commons.lang3.StringUtils.java 4 0.21257 indri
Lang_10 Q0 org.apache.commons.lang3.text.StrMatcher.java 5 0.19477 indri
Lang_10 Q0 org.apache.commons.lang3.time.DateUtils.java 6 0.173851 indri
Lang_10 Q0 org.apache.commons.lang3.text.WordUtils.java 7 0.164297 indri
Lang_10 Q0 org.apache.commons.lang3.time.DateParser.java 8 0.150949 indri
Lang_10 Q0 org.apache.commons.lang3.time.DurationFormatUtils.java 9 0.14653 indri
Lang_10 Q0 org.apache.commons.lang3.math.Fraction.java 10 0.139247 indri
```



Ranked list of
suspicious
source files

Step 4: Extract 57 possible AST nodes from source files and create a *statement* document collection

```
Lang_10 Q0 org.apache.commons.lang3.time.FastDateParser.java 1 0.403753 indri
Lang_10 Q0 org.apache.commons.lang3.time.FastDateFormat.java 2 0.225249 indri
Lang_10 Q0 org.apache.commons.lang3.time.FastDatePrinter.java 3 0.217153 indri
Lang_10 Q0 org.apache.commons.lang3.StringUtils.java 4 0.21257 indri
Lang_10 Q0 org.apache.commons.lang3.text.StrMatcher.java 5 0.19477 indri
Lang_10 Q0 org.apache.commons.lang3.time.DateUtils.java 6 0.173851 indri
Lang_10 Q0 org.apache.commons.lang3.text.WordUtils.java 7 0.164297 indri
Lang_10 Q0 org.apache.commons.lang3.time.DateParser.java 8 0.150949 indri
Lang_10 Q0 org.apache.commons.lang3.time.DurationFormatUtils.java 9 0.14653 indri
Lang_10 Q0 org.apache.commons.lang3.math.Fraction.java 10 0.139247 indri
```



Ranked list of
suspicious
source files

Step 4: Extract 57 possible AST nodes from source files and create a *statement* document collection

FastDateParser.java

```
304 boolean wasWhite= false;
305 for(int i= 0; i<value.length(); ++i) {
306     char c= value.charAt(i);
307     if(Character.isWhitespace(c)) {
308         if(!wasWhite) {
309             wasWhite= true;
310             regex.append("\\s*+");
311         }
312         continue;
313     }
314     wasWhite= false;
315     switch(c) {
316         case '\\':
```

IfStatement node
line# 308

ConditionalExpression node
line# 308

line#

AST node#

```
1 <DOC>
2   <DOCNO>FastDateParser.java-308-269 </DOCNO>
3   <identifier>
4     was, white, was, white, true, regex, append
5   </identifier>
6 </DOC>
```

```
1 <DOC>
2   <DOCNO>FastDateParser.java-308-270 </DOCNO>
3   <identifier>
4     was,white
5   </identifier>
6 </DOC>
```

statement
document collection

Step 5: Execute IR model to rank suspicious statements for given query

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <bugrepository name="Defects4J">
3   <bug id="Lang_10" link="https://issues.apache.org/jira/browse/LANG-831">
4     <buginformation>
5       <summary>fast date parser does not handle white space
6         properly</summary>
7       <description>the simpledateformat javadoc does not treat
8         white space specially however fastdateparser treats
9         single white space being any number white space
10        characters this means that fdp will parse dates that
11        fail when parsed sdp</description>
12     </buginformation>
13     <fixedfiles>
14       <file>org.apache.commons.lang3.time.FastDateParser</file>
15     </fixedfiles>
16   </bug>
17 </bugrepository>
```

summary

description

Bug report query

Execute IR model two times

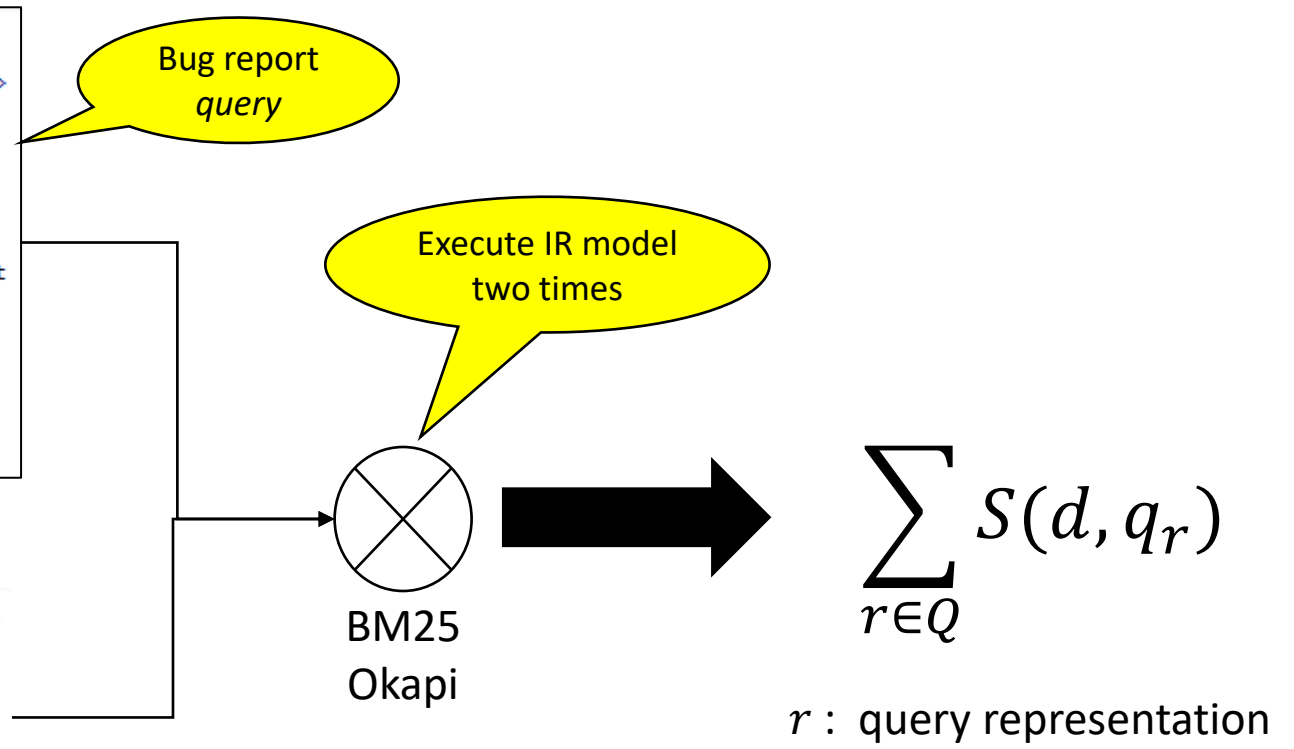
```
1 <nrc>
2   1 <DOC>
3     2   1 <DOC>
4       3   2
5         4   3   1 <DOC>
6           4     2 <DOCNO>FastDateParser.java-308-270 </DOCNO>
7             3 <identifier>
8               4   was,white
9             </identifier>
10          </DOC>
11        </DOC>
12      </DOC>
```

statement document collection

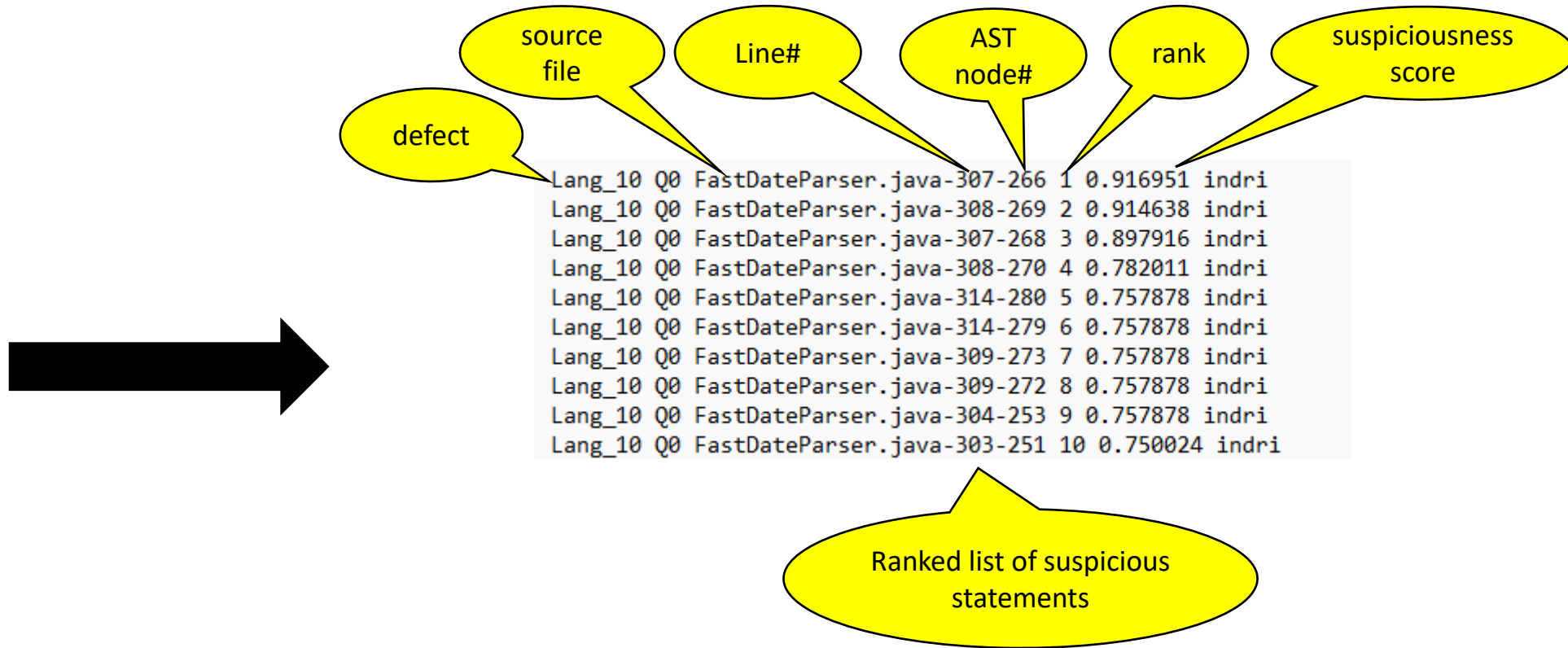


$$\sum_{r \in Q} S(d, q_r)$$

r : query representation



Step 5: Execute IR model to rank suspicious statements for given query



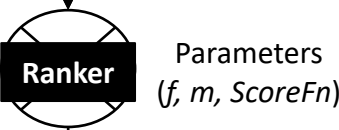
Step 6: Combine scores of ranked files and statements

```
Lang_10 Q0 org.apache.commons.lang3.time.FastDateParser.java 1 0.403753 indri  
Lang_10 Q0 org.apache.commons.lang3.time.FastDateFormat.java 2 0.225249 indri  
Lang_10 Q0 org.apache.commons.lang3.time.FastDatePrinter.java 3 0.217153 indri  
Lang_10 Q0 org.apache.commons.lang3.StringUtils.java 4 0.21257 indri  
Lang_10 Q0 org.apache.commons.lang3.text.StrMatcher.java 5 0.19477 indri  
Lang_10 Q0 org.apache.commons.lang3.time.DateUtils.java 6 0.173851 indri  
Lang_10 Q0 org.apache.commons.lang3.text.WordUtils.java 7 0.164297 indri  
Lang_10 Q0 org.apache.commons.lang3.time.DateParser.java 8 0.150949 indri  
Lang_10 Q0 org.apache.commons.lang3.time.DurationFormatUtils.java 9 0.14653 indri  
Lang_10 Q0 org.apache.commons.lang3.math.Fraction.java 10 0.139247 indri
```

```
Lang_10 Q0 FastDateParser.java-307-266 1 0.916951 indri  
Lang_10 Q0 FastDateParser.java-308-269 2 0.914638 indri  
Lang_10 Q0 FastDateParser.java-307-268 3 0.897916 indri  
Lang_10 Q0 FastDateParser.java-308-270 4 0.782011 indri  
Lang_10 Q0 FastDateParser.java-314-280 5 0.757878 indri  
Lang_10 Q0 FastDateParser.java-314-279 6 0.757878 indri  
Lang_10 Q0 FastDateParser.java-309-273 7 0.757878 indri  
Lang_10 Q0 FastDateParser.java-309-272 8 0.757878 indri  
Lang_10 Q0 FastDateParser.java-304-253 9 0.757878 indri  
Lang_10 Q0 FastDateParser.java-303-251 10 0.750024 indri
```

Ranked list of suspicious source files

Ranked list of suspicious statements per ranked file



```
Statement,Suspiciousness  
org.apache.commons.lang3.time.FastDateParser#307,1.0  
org.apache.commons.lang3.time.FastDateFormat#370,0.9722222222222222  
org.apache.commons.lang3.time.FastDatePrinter#511,0.9444444444444444  
org.apache.commons.lang3.StringUtils#5005,0.9166666666666666  
org.apache.commons.lang3.text.StrMatcher#98,0.8888888888888888  
org.apache.commons.lang3.time.DateUtils#353,0.8611111111111111  
org.apache.commons.lang3.text.WordUtils#122,0.8333333333333333
```

Final ranked list of suspicious statements

Comparison of Blues With State-Of-The-Art and Baseline

The number of defects localized within the Top- k ranked statements

171 Lang and Math defects in Defects4J v1.0

The fraction of ranked statements that must be inspected until finding a buggy statement.

↑ Higher is better

↓ Lower is better

(171 defects)	$hit@k$					EXAM
	$k = 1$	25	50	100	all	$k = all$
iFixR	26	74	95	106	135	0.048
Blues	11	79	97	108	151	0.034

statement-level IRFL technique used in iFixR repair tool

Comparison of Blues With State-Of-The-Art and Baseline

The number of defects localized within the Top- k ranked statements

171 Lang and Math defects in Defects4J v1.0

The fraction of ranked statements that must be inspected until finding a buggy statement.

↑ Higher is better

↓ Lower is better

(171 defects)	<i>hit@k</i>					EXAM
	$k = 1$	25	50	100	all	$k = \text{all}$
iFixR	26	74	95	106	135	0.048
Blues	11	79	97	108	151	0.034

statement-level IRFL technique used in iFixR repair tool

815 defects from 17 projects in Defects4J v2.0

↑ Higher is better

↓ Lower is better

(815 defects)	<i>hit@k</i>					EXAM
	$k = 1$	25	50	100	all	$k = \text{all}$
vanilla BLUiR	26	143	192	245	611	0.159
Blues	27	184	241	306	611	0.111

Does not consider suspicious file scores

Comparison of Blues With State-Of-The-Art and Baseline

The number of defects localized within the Top- k ranked statements

171 Lang and Math defects in Defects4J v1.0

The fraction of ranked statements that must be inspected until finding a buggy statement.

(171 defects)	↑ Higher is better					↓ Lower is better
	<i>hit@k</i>					EXAM
	$k = 1$	25	50	100	all	$k = \text{all}$
iFixR	26	74	95	106	135	0.048
Blues	11	79	97	108	151	0.034

statement-level IRFL technique used in

For scenarios relevant to APR ($k \geq 25$), Blues consistently outperforms the state-of-the-art and baseline.

815 defects from 17 projects in Defects4J v2.0

Does not consider suspicious file scores

(815 defects)	↑ Higher is better					↓ Lower is better
	<i>hit@k</i>					EXAM
	$k = 1$	25	50	100	all	$k = \text{all}$
vanilla BLUiR	26	143	192	245	611	0.159
Blues	27	184	241	306	611	0.111

How to Combine Multiple FL Techniques for APR?

Test Suite



Spectrum-based FL



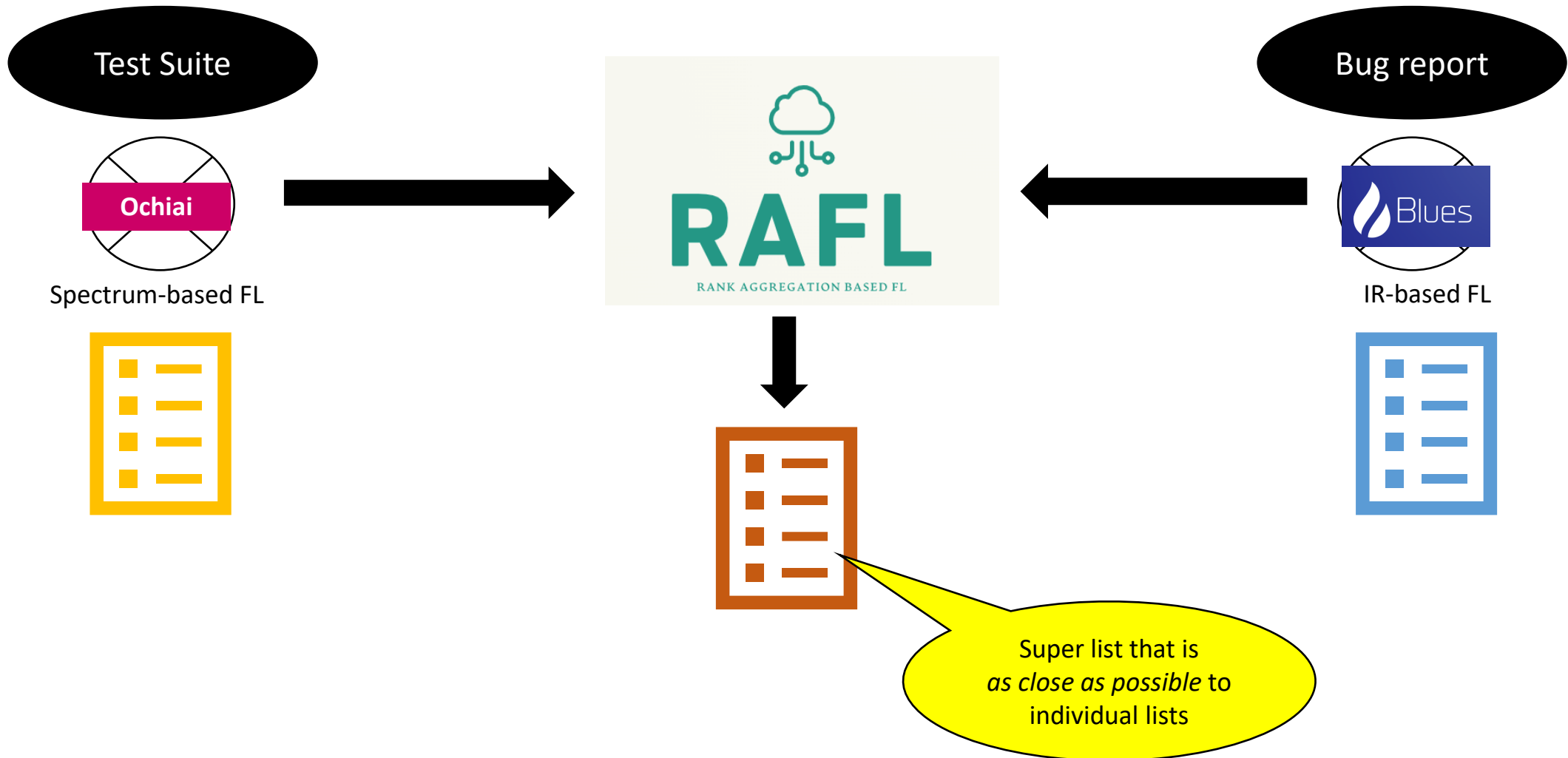
Bug report



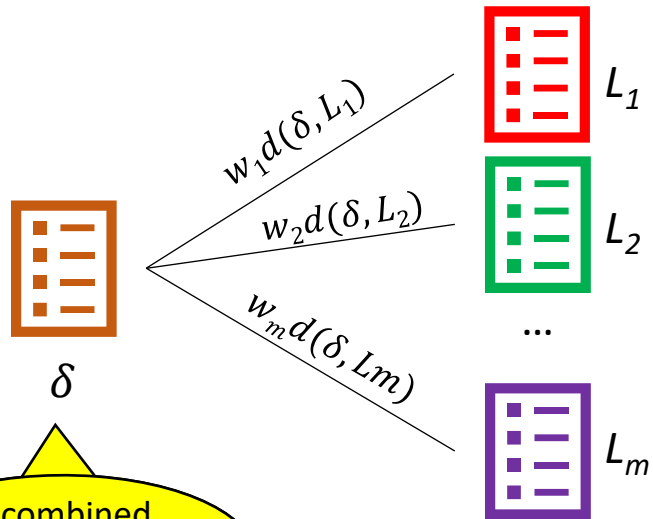
IR-based FL



RAFL: Rank-Aggregation-Based Fault Localization



Key Insight: Formulate Rank-Aggregation as an Optimization Problem



δ : combined ordered list of length $k \leq |L_i|$

L_i : i^{th} ordered list of ranked suspicious statements

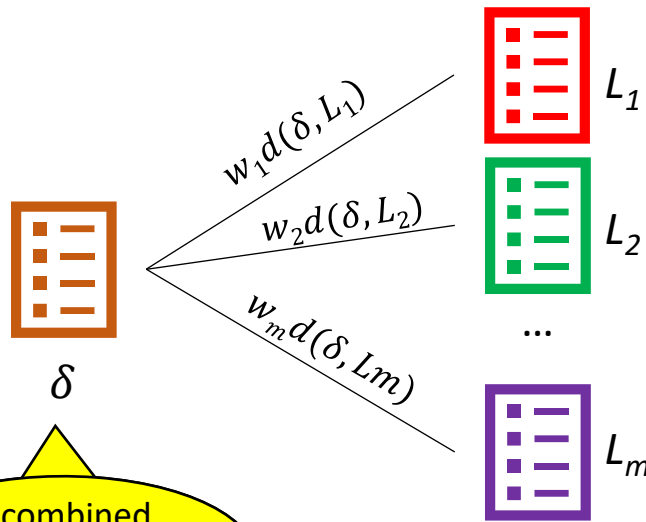
$$f(\delta) = \sum_{i=1}^m w_i d(\delta, L_i)$$

d : distance function

Objective function defined in terms of the weighted sum of distances between the combined list and individual lists

w_i : optional importance weight associated with L_i

RAFL: Rank-Aggregation-Based Fault Localization



Goal: Find a combined list (δ^*) whose sum of distances from individual lists is minimum

$$\delta^* = \arg \min f(\delta) = \arg \min \sum_{i=1}^m w_i d(\delta, L_i)$$

δ : combined ordered list of length $k \leq |L_i|$

L_i : i^{th} ordered list of ranked suspicious statements

δ^* : ordered list of length $k \leq |L_i|$ that is as close as possible to all L_i

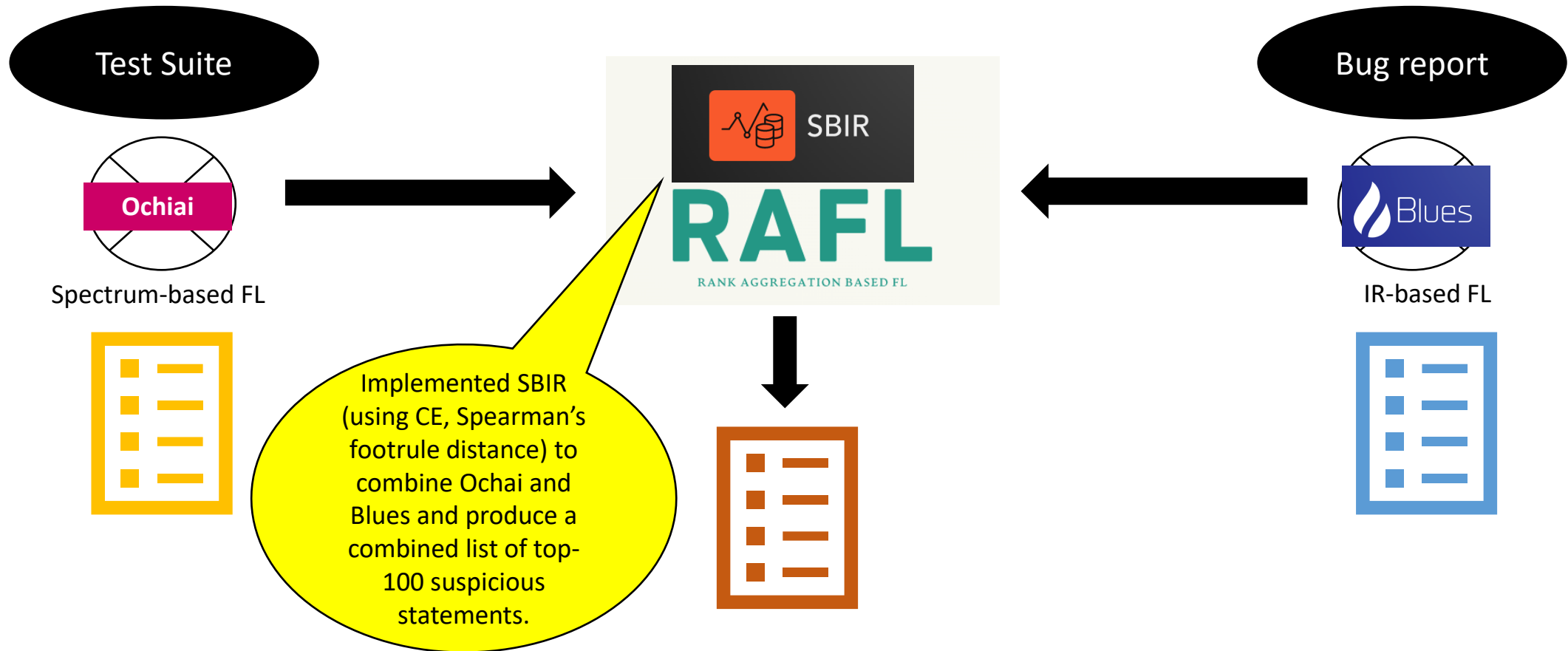
$$f(\delta) = \sum_{i=1}^m w_i d(\delta, L_i)$$

Objective function defined in terms of the weighted sum of distances between the combined list and individual lists

d : distance function

w_i : optional importance weight associated with L_i

SBIR: Combining SBFL With Blues Using RAFL



Performance of SBIR Compared to SBFL and Blues

815 defects from 17 real-world Java projects available in Defects4J v2.0

The number of defects localized within the Top- k ranked statements

The fraction of ranked statements that must be inspected until finding a buggy statement.

(815 defects)		↑ Higher is better				↓ Lower is better		
		<i>hit@k</i>				EXAM		
		$k = 1$	25	50	100	$k = 25$	50	100
SBFL		88	408	475	549	0.287	0.240	0.220
Blues		27	184	241	306	0.332	0.300	0.270
SBIR (10 seeds)	mean	101	419	489	557	0.256	0.215	0.187
	stdev	7.60	5.01	5.40	4.22	0.006	0.006	0.005
	cv	0.08	0.01	0.01	0.01	0.023	0.026	0.028

Coefficient of variation

$$cv = \frac{\text{stdev}}{\text{mean}} < 0.1 \text{ means 10 seeds' results are tightly coupled}$$

Performance of SBIR Compared to SBFL and Blues

815 defects from 17 real-world Java projects available in Defects4J v2.0

The number of defects localized within the Top- k ranked statements

The fraction of ranked statements that must be inspected until finding a buggy statement.

(815 defects)		↑ Higher is better				↓ Lower is better		
		<i>hit@k</i>				EXAM		
		$k = 1$	25	50	100	$k = 25$	50	100
SBFL		88	408	475	549	0.287	0.240	0.220
Blues		27	184	241	306	0.332	0.300	0.270
SBIR	mean	101	419	489	557	0.256	0.215	0.187
(10 seeds)	stdev	7.60	5.01	5.40	4.22	0.006	0.006	0.005
	cv	0.08	0.01	0.01	0.01	0.023	0.026	0.028

Coefficient of variation

$$cv = \frac{\text{stdev}}{\text{mean}} < 0.1 \text{ means 10 seeds' results are tightly coupled}$$

SBIR outperforms SBFL and Blues corroborating existing research^{1,2,3} on combining FL techniques.

Performance of SBIR Compared to the State-Of-The-Art FL

334 defects from 5 real-world Java projects available in Defects4J v1.0

The number of defects localized within the Top-*k* ranked statements

The fraction of ranked statements that must be inspected until finding a buggy statement.

(334 defects)		↑ Higher is better				↓ Lower is better
		<i>hit@k</i>				EXAM
family	technique	<i>k</i> = 1	25	50	100	<i>k</i> = 100
SBFL	Ochiai	30	168	196	221	0.254
	DStar	32	169	199	222	0.254
MBFL	Metallaxis	40	154	175	195	0.238
	MUSE	26	96	104	118	0.193
slicing	slicing-union	21	87	100	111	0.462
	slicing-intersection	18	71	81	91	0.481
	slicing-frequency	21	86	100	112	0.458
stack trace	stack trace	16	28	28	28	0.663
predicate switching	predicate switching	9	24	24	24	0.662
SBIR (RAFL)	mean	48	177	207	231	0.175
(10 seeds)	stdev	4.31	4.16	2.92	2.32	0.006
	cv	0.09	0.02	0.01	0.01	0.034

Coefficient of variation

$$cv = \frac{\text{stdev}}{\text{mean}} < 0.1 \text{ means 10 seeds' results are tightly coupled}$$

Performance of SBIR Compared to the State-Of-The-Art FL

334 defects from 5 real-world Java projects available in Defects4J v1.0

The number of defects localized within the Top-k ranked statements

The fraction of ranked statements that must be inspected until finding a buggy statement.

↑ Higher is better ↓ Lower is better

(334 defects)		<i>hit@k</i>				EXAM
family	technique	<i>k</i> = 1	25	50	100	<i>k</i> = 100
SBFL	Ochiai	30	168	196	221	0.254
	DStar	32	169	199	222	0.254
MBFL	Metallaxis	40	154	175	195	0.238

SBIR outperforms nine state-of-the-art FL techniques by localizing more defects and ranking buggy statements higher.

stack trace		21	30	100	112	0.158
stack trace	stack trace	16	28	28	28	0.663
predicate switching	predicate switching	9	24	24	24	0.662
SBIR (RAFL)	mean	48	177	207	231	0.175
(10 seeds)	stdev	4.31	4.16	2.92	2.32	0.006
	cv	0.09	0.02	0.01	0.01	0.034

Coefficient of variation
 $cv = \frac{stdev}{mean}$ < 0.1 means 10 seeds' results are tightly coupled

Performance of SBIR Compared to the Supervised Combining Method

815 defects from 17 real-world Java projects available in Defects4J v2.0

The number of defects localized within the Top- k ranked statements

The fraction of ranked statements that must be inspected until finding a buggy statement.

(815 defects)		↑ Higher is better				↓ Lower is better
		$E_{inspect}@k$				$EXAM_{inspect}$
technique		$k = 1$	25	50	100	$k = 100$
SBIR(RankSVM)		50	270	328	396	0.236
SBIR (RAFL)	mean	101	419	489	557	0.187
(10 seeds)	stdev	7.60	5.01	5.40	4.22	0.005
	cv	0.08	0.01	0.01	0.01	0.027

Coefficient of variation
 $cv = \frac{\text{stdev}}{\text{mean}}$ < 0.1 means 10 seeds' results are tightly coupled

Performance of SBIR Compared to the Supervised Combining Method

815 defects from 17 real-world Java projects available in Defects4J v2.0

The number of defects localized within the Top- k ranked statements

The fraction of ranked statements that must be inspected until finding a buggy statement.

(815 defects)		↑ Higher is better				↓ Lower is better
		$E_{inspect}@k$				$EXAM_{inspect}$
technique		$k = 1$	25	50	100	$k = 100$
SBIR(RankSVM)		50	270	328	396	0.236
SBIR (RAFL)	mean	101	419	489	557	0.187
(10 seeds)	stdev	7.60	5.01	5.40	4.22	0.005
	cv	0.08	0.01	0.01	0.01	0.027

Coefficient of variation
 $cv = \frac{\text{stdev}}{\text{mean}}$ < 0.1 means 10 seeds' results are tightly coupled

SBIR using unsupervised RAFL outperforms SBIR using supervised RankSVM, which is used in many state-of-the-art combining FL techniques^{1,2,3,4}.

Effect of Using SBIR on Repair Performance

395 defects from 6 real-world Java projects
available in Defects4J version < 2.0

SBIR enables repair tools to correctly patch many new defects without modifying their patch generation algorithms.

Arja

Chart-12
Closure-78
Closure-86
Lang-10
Lang-20

SequenceR

Closure-86

SimFix

Closure-68
Closure-92

Using SBIR enables APR tools to **correctly repair** 7 new defects that **14 existing techniques¹** couldn't fix

Using SBIR enables APR tools to **correctly repair** 7 new defects that **they** couldn't fix earlier

Lang-7
Lang-10
Lang-59
Math-35

(Uses
Perfect FL)

Closure-68
Closure-92
Closure-126

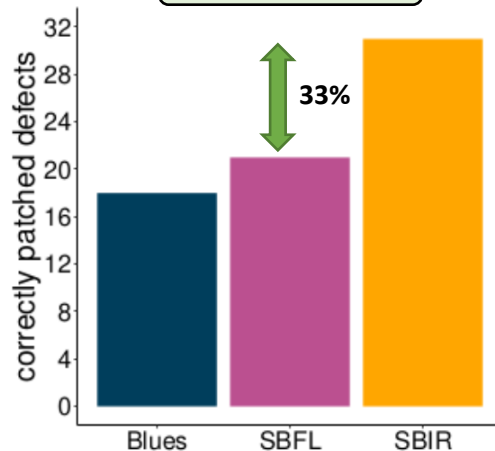
Effect of Using SBIR on Repair Quality

689 single file edit defects from 17 real-world Java projects available in Defects4J v2.0

SBIR significantly improves the quality of more FL-sensitive APR tools.

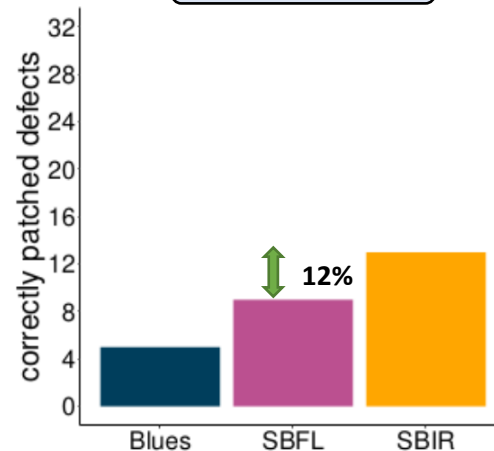
High FL sensitivity

Arja



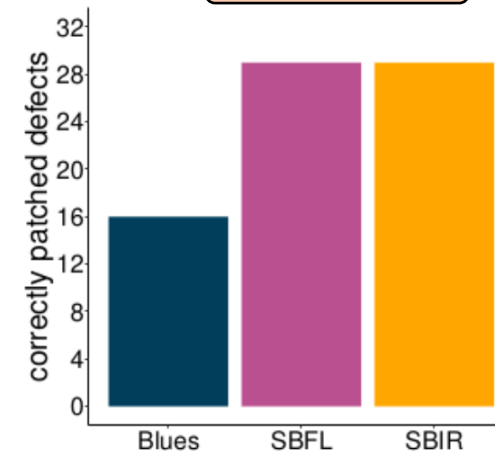
Med FL sensitivity

SequenceR



Low FL-sensitivity

SimFix



Effect of Using SBIR on Localization Error

689 single file edit defects from 17 real-world Java projects available in Defects4J v2.0

	Arja	SequenceR	SimFix
localization error assessment			
upper bound	36	24	32
↓ # of defects not correctly patched due to localization error ↓			↓ Lower is better
SBFL	15	14	2
Blues	21	20	19
SBIR	8	12	2

SBIR lowers the localization error by providing repair tools an earlier opportunity to patch actual buggy statements.



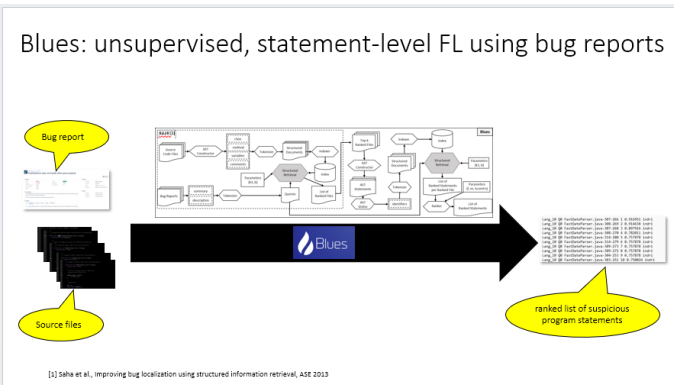
scan me for paper



Contributions



scan me for artifact



<https://github.com/LASER-UMASS/Blues>



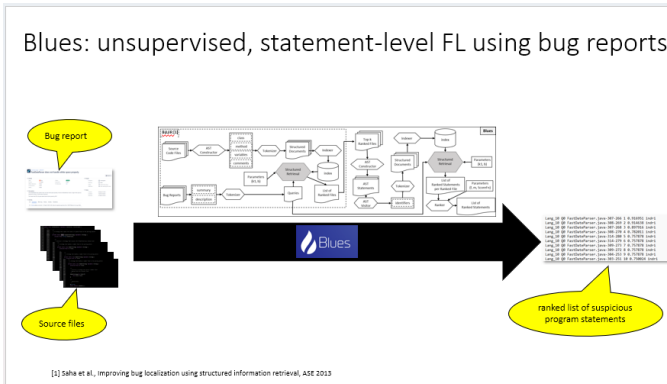
scan me for paper



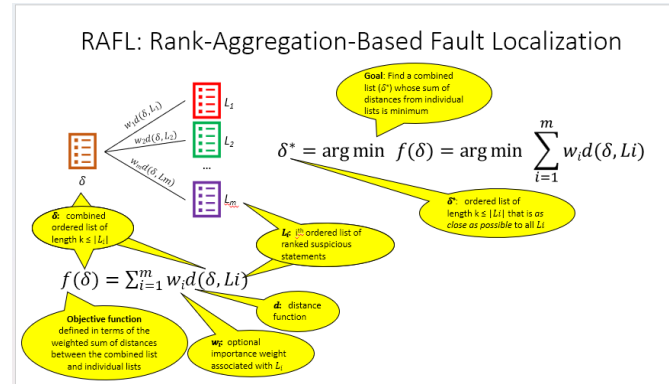
Contributions



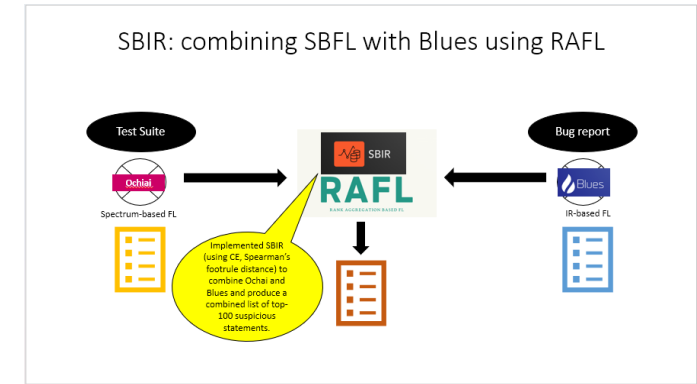
scan me for artifact



<https://github.com/LASER-UMASS/Blues>



<https://github.com/LASER-UMASS/RAFL>



<https://github.com/LASER-UMASS/SBIR-ReplicationPackage>



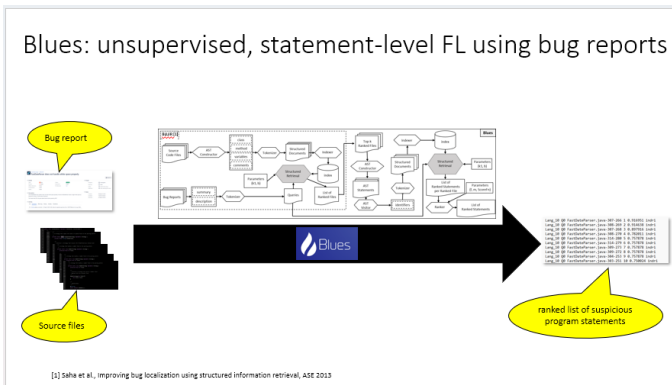
scan me for paper



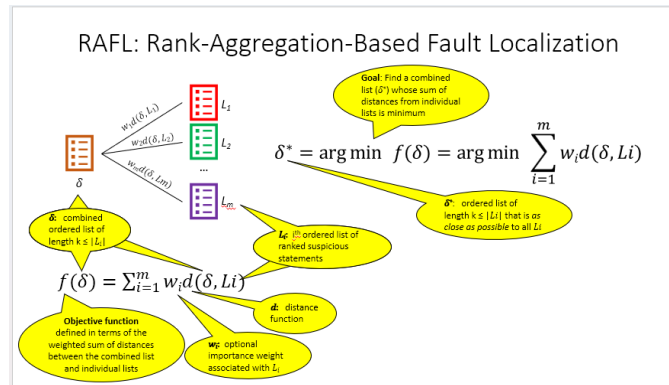
Contributions



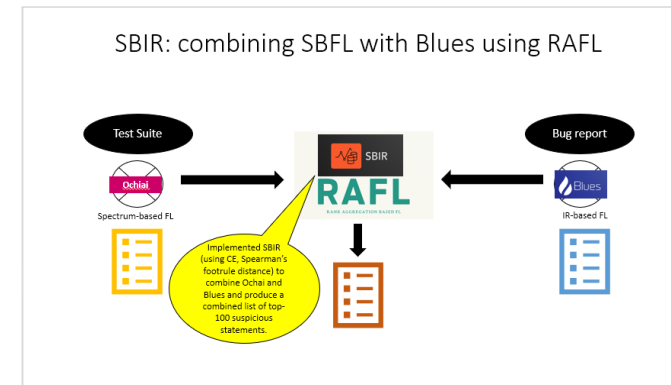
scan me for artifact



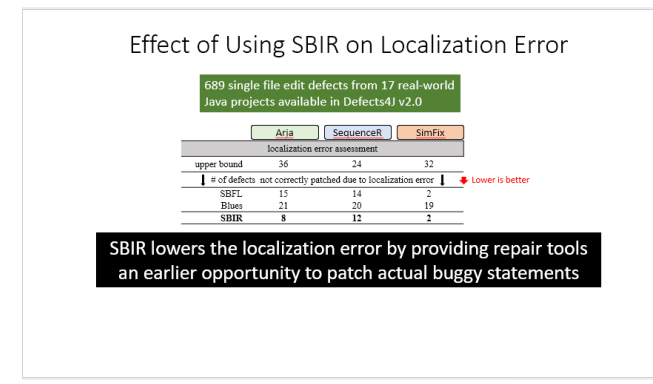
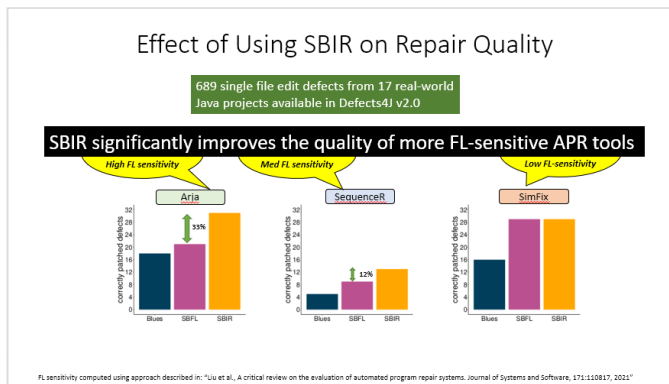
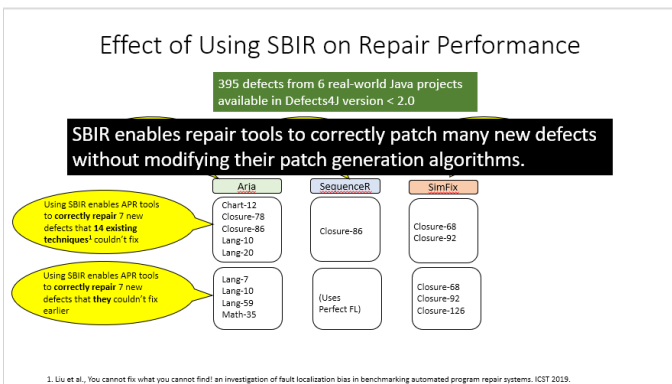
<https://github.com/LASER-UMASS/Blues>



<https://github.com/LASER-UMASS/RAFL>



<https://github.com/LASER-UMASS/SBIR-ReplicationPackage>



<https://manishmotwani3.github.io/>