Better Automatic Program Repair by Using Bug Reports and Tests Together

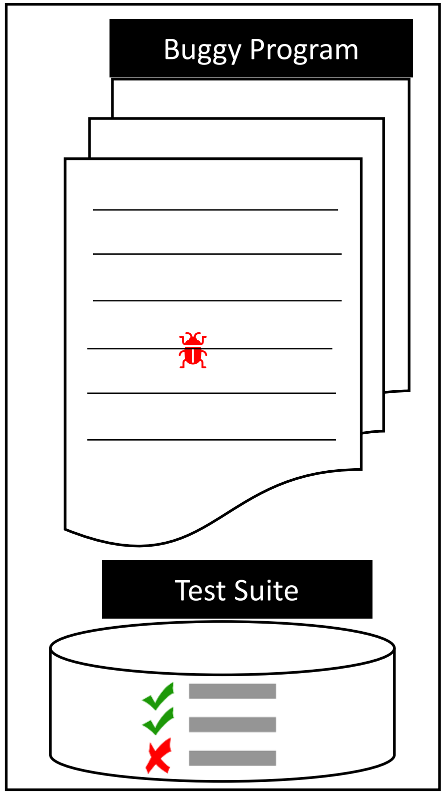
Manish Motwani

UMassAmherst

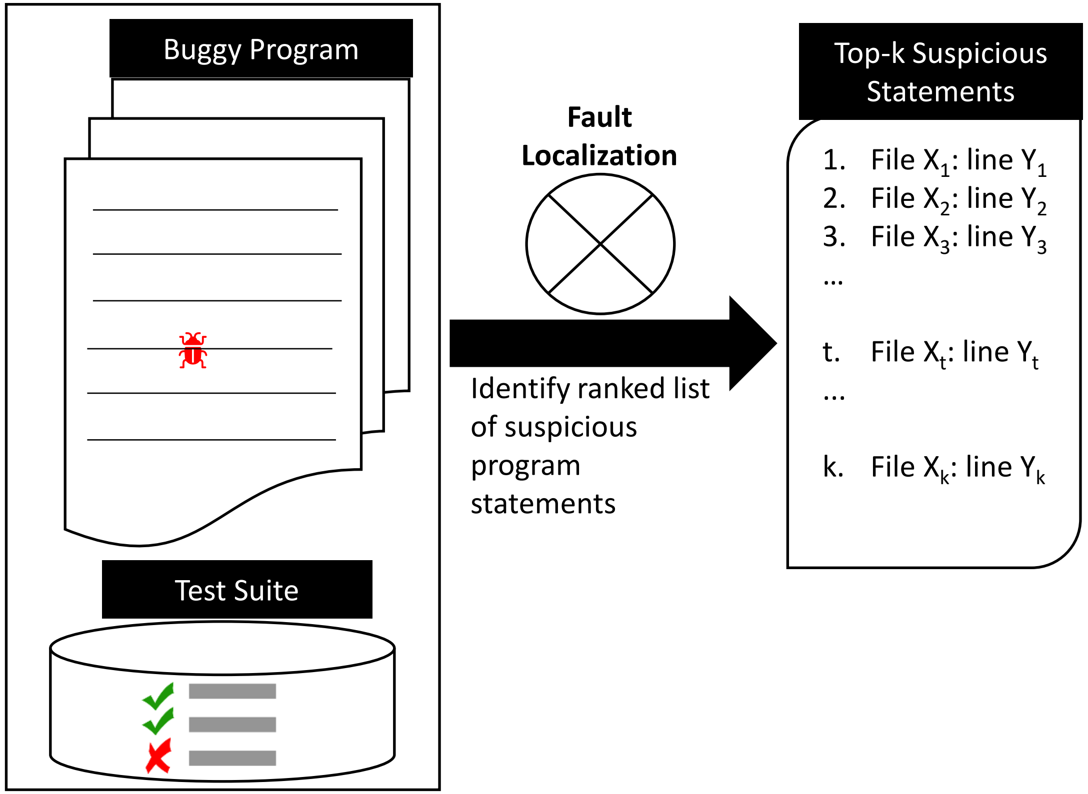
Yuriy Brun

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

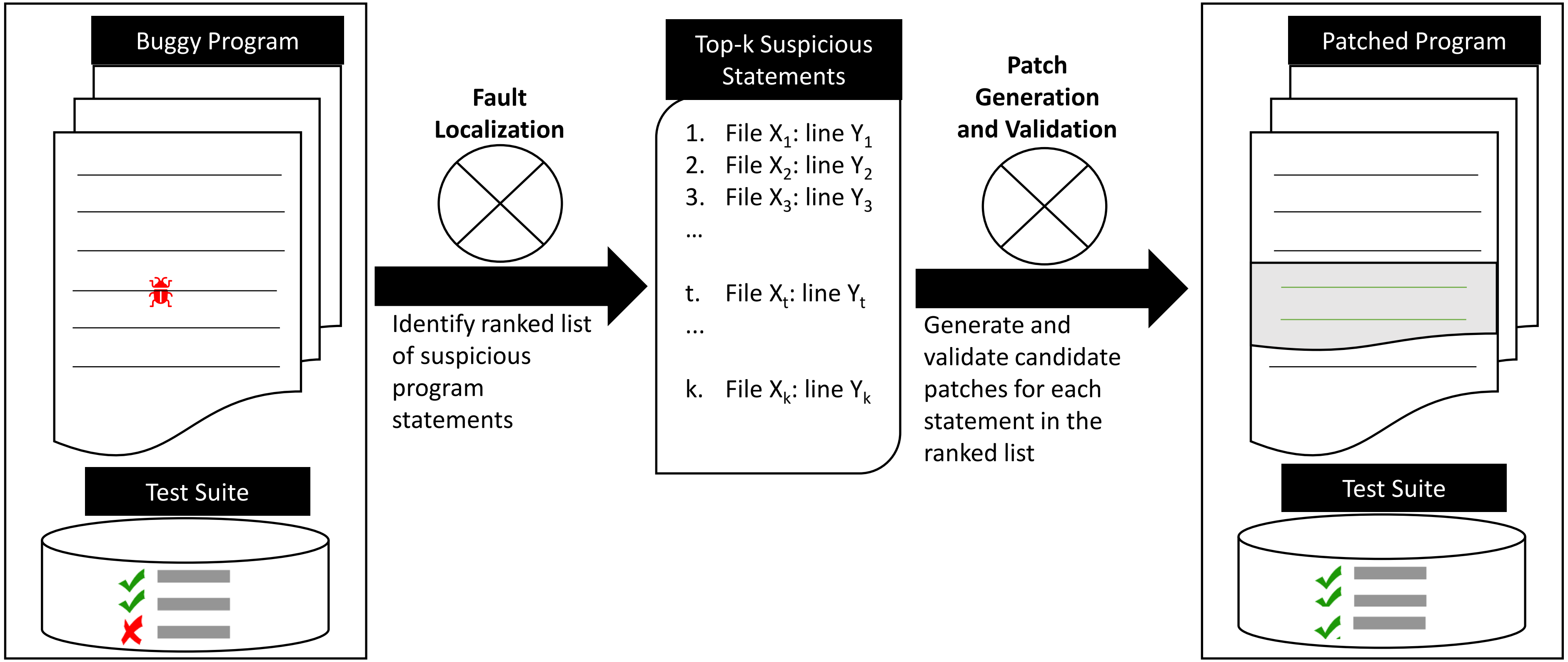
Automatic Program Repair (APR) Process



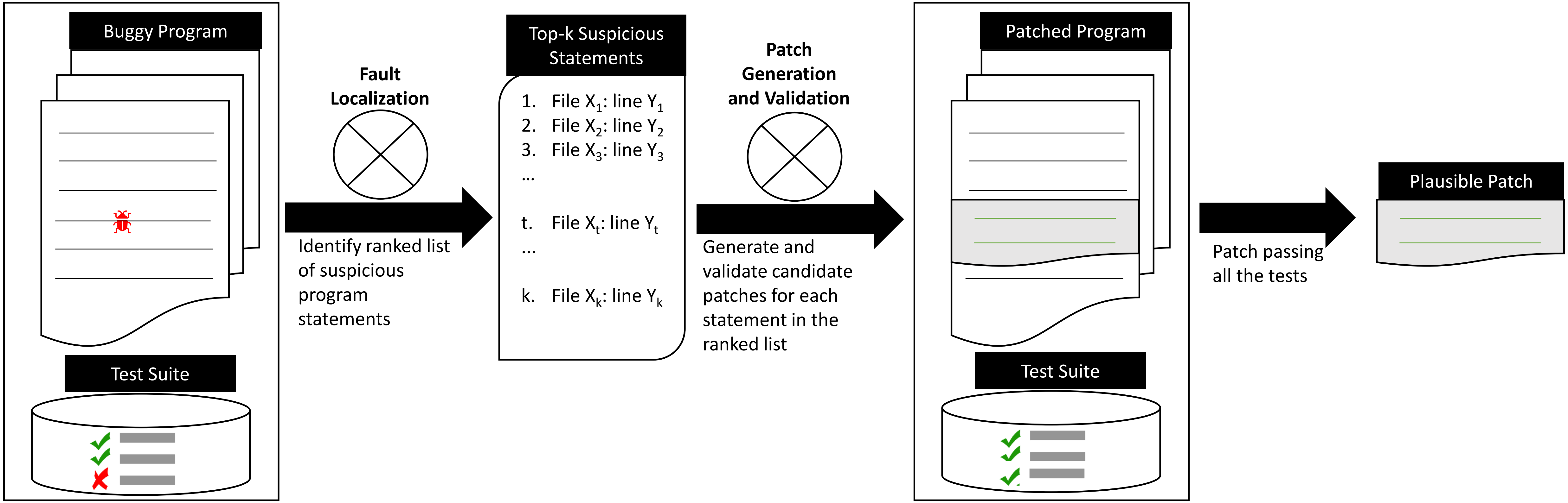
Automatic Program Repair (APR) Process



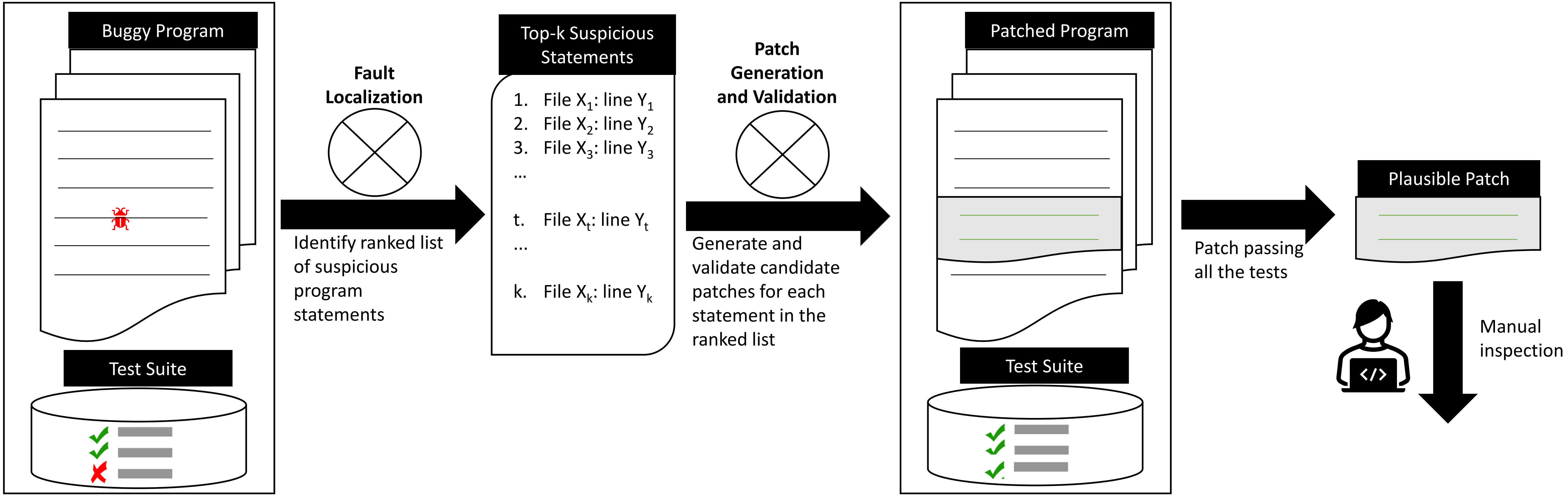
Automatic Program Repair (APR) Process



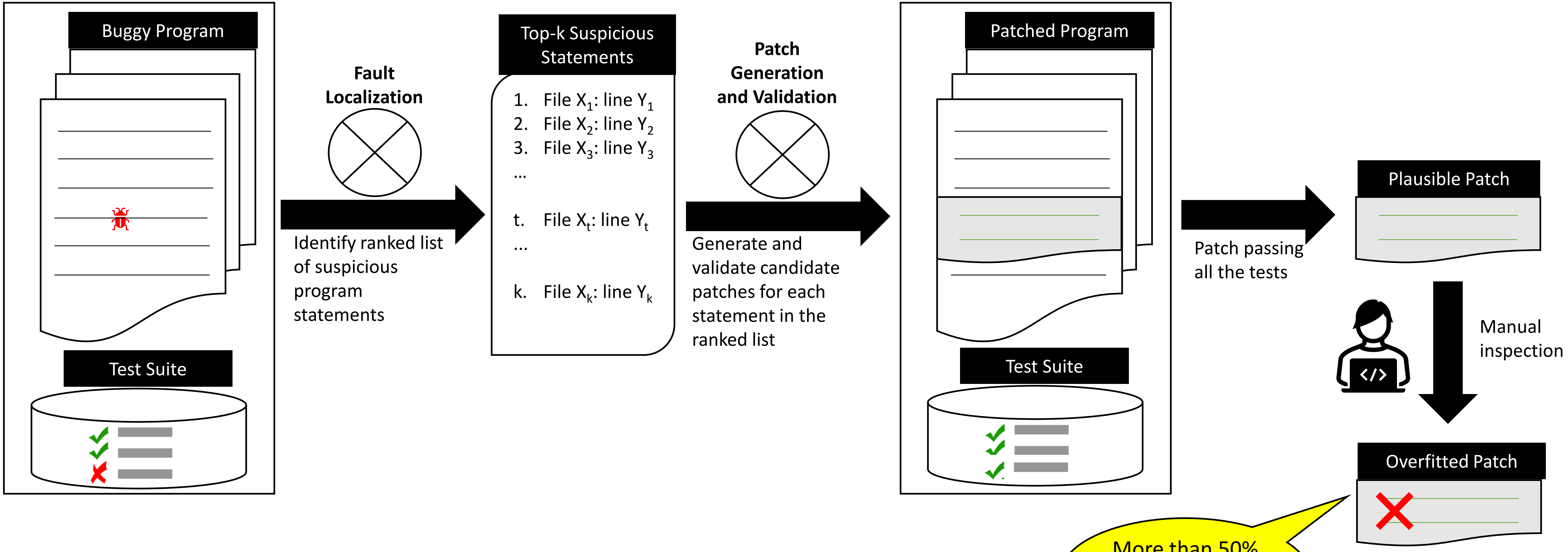
Automatic Program Repair (APR) Process

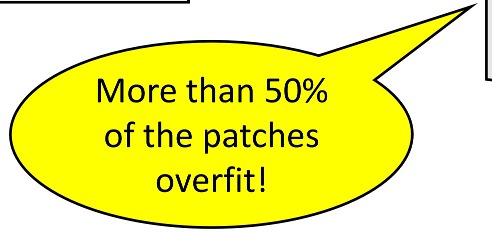


Repair Techniques Struggle to Patch Defects Correctly



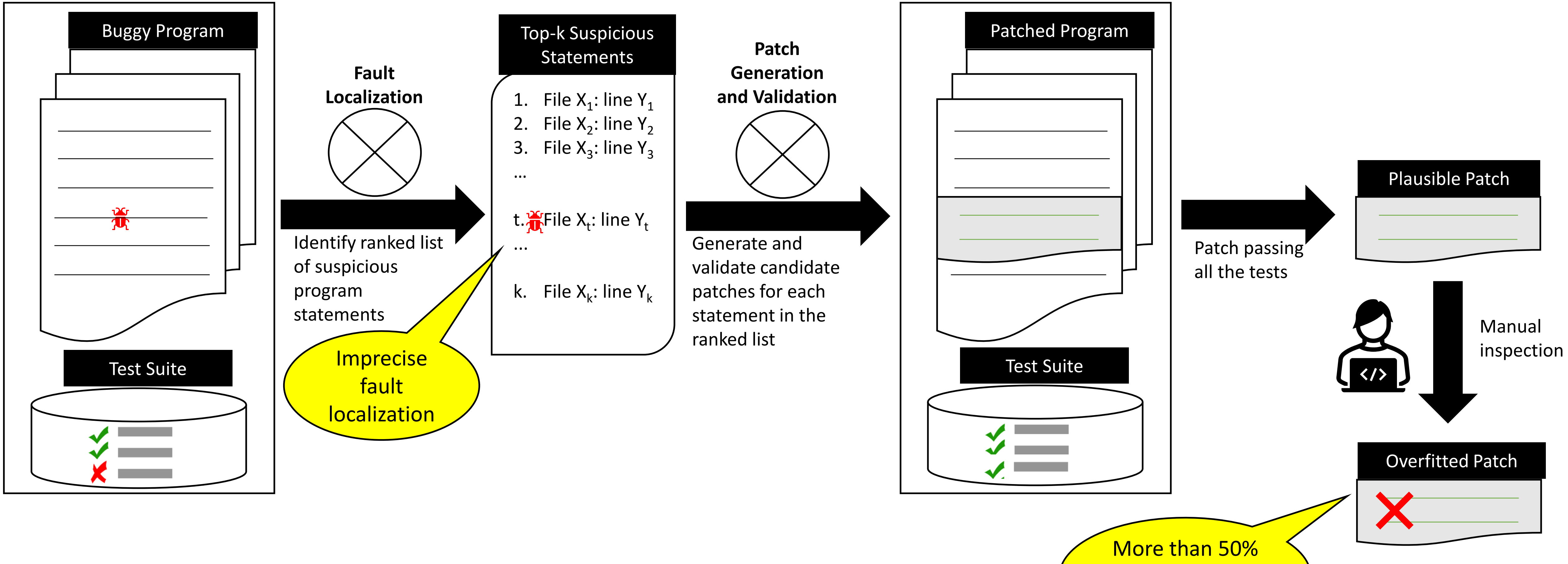
Repair Techniques Struggle to Patch Defects Correctly





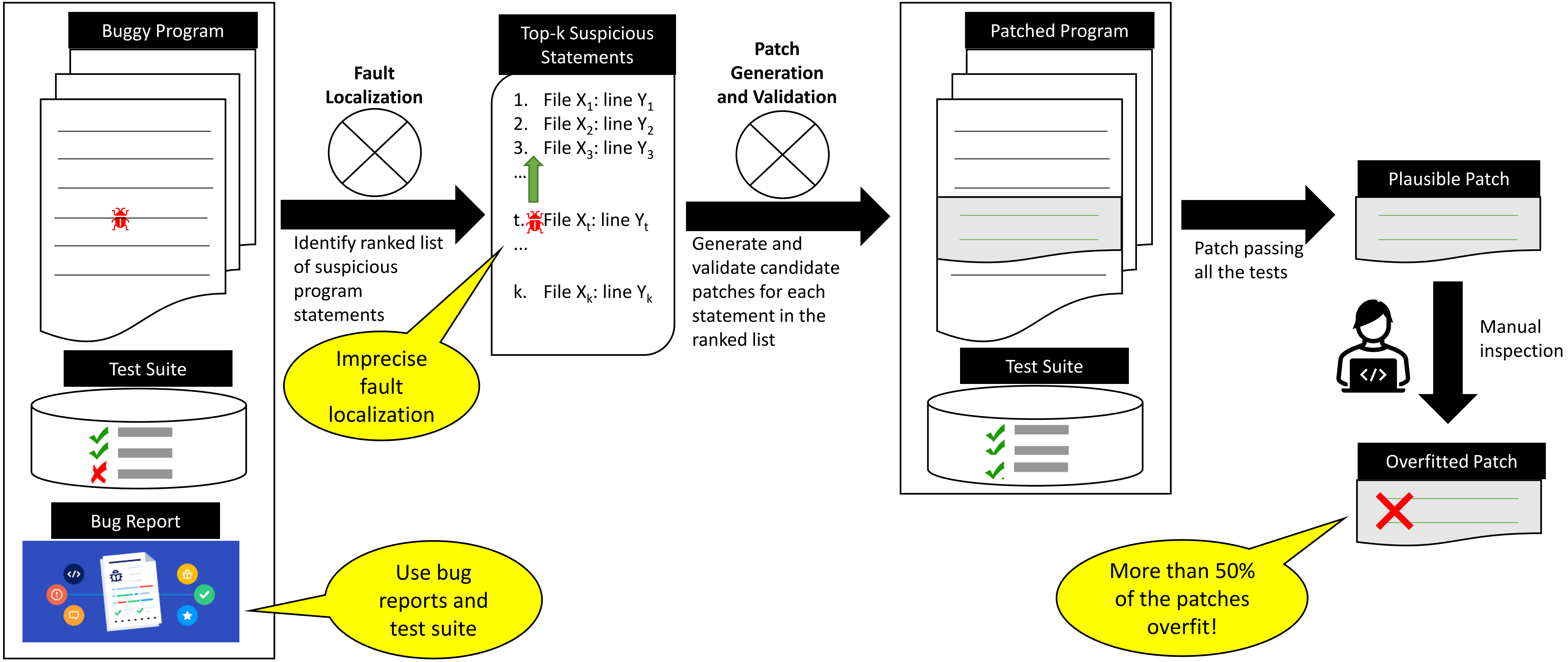
Smith et al., Is the cure worse than the disease? Overfitting in automated program repair, ESEC/FSE, 2015
Long et al., An analysis of patch plausibility and correctness for generate-and-validate patch generation systems, ISSTA, 2015
Long et al., An analysis of the search spaces for generate and validate 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



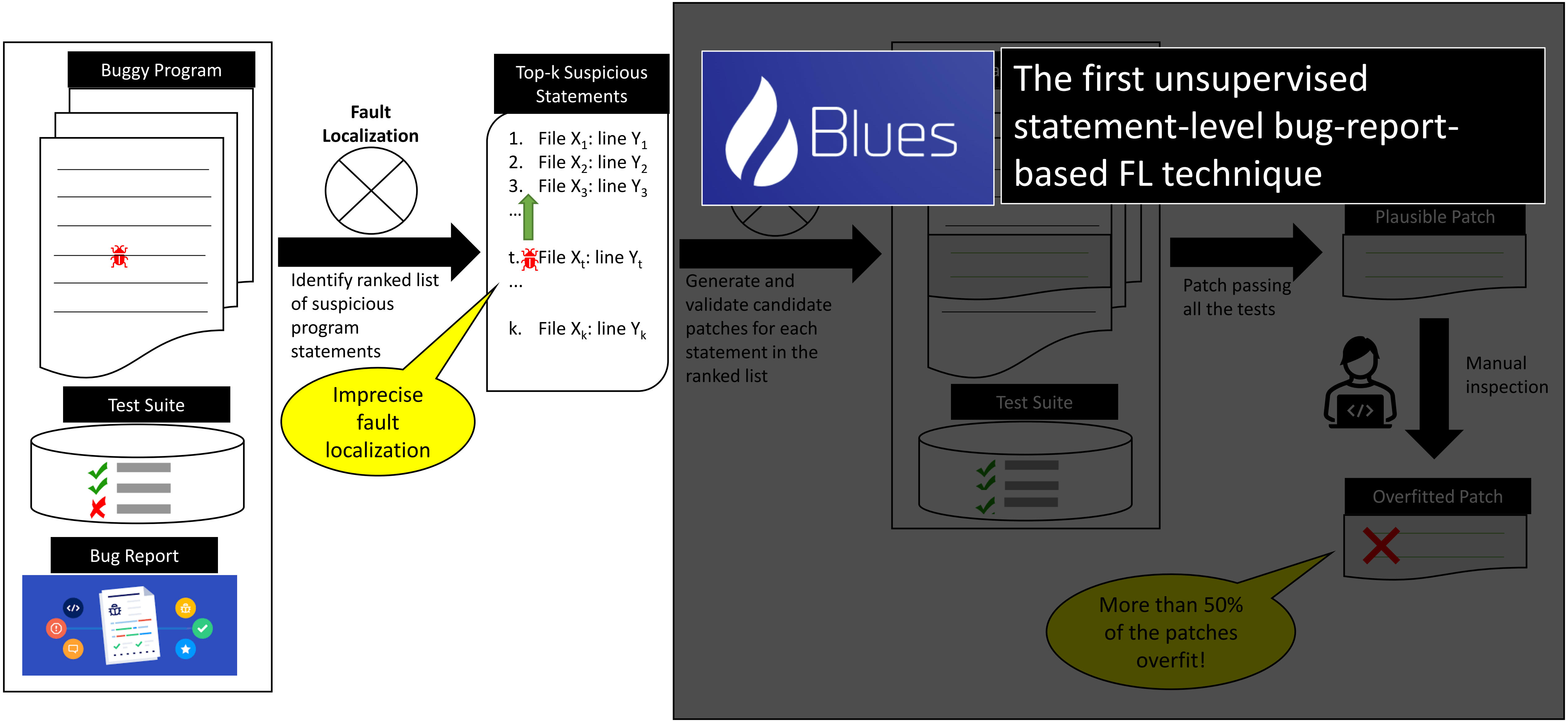
Afzal et al., SOSRepair: Expressive semantic search for real-world program repair, TSE, 2021
Yang et al., Evaluating the strategies of statement selection in automated program repair, ISSTA, 2018
Wen et al., An empirical analysis of the influence of fault space on search-based automated program repair, 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

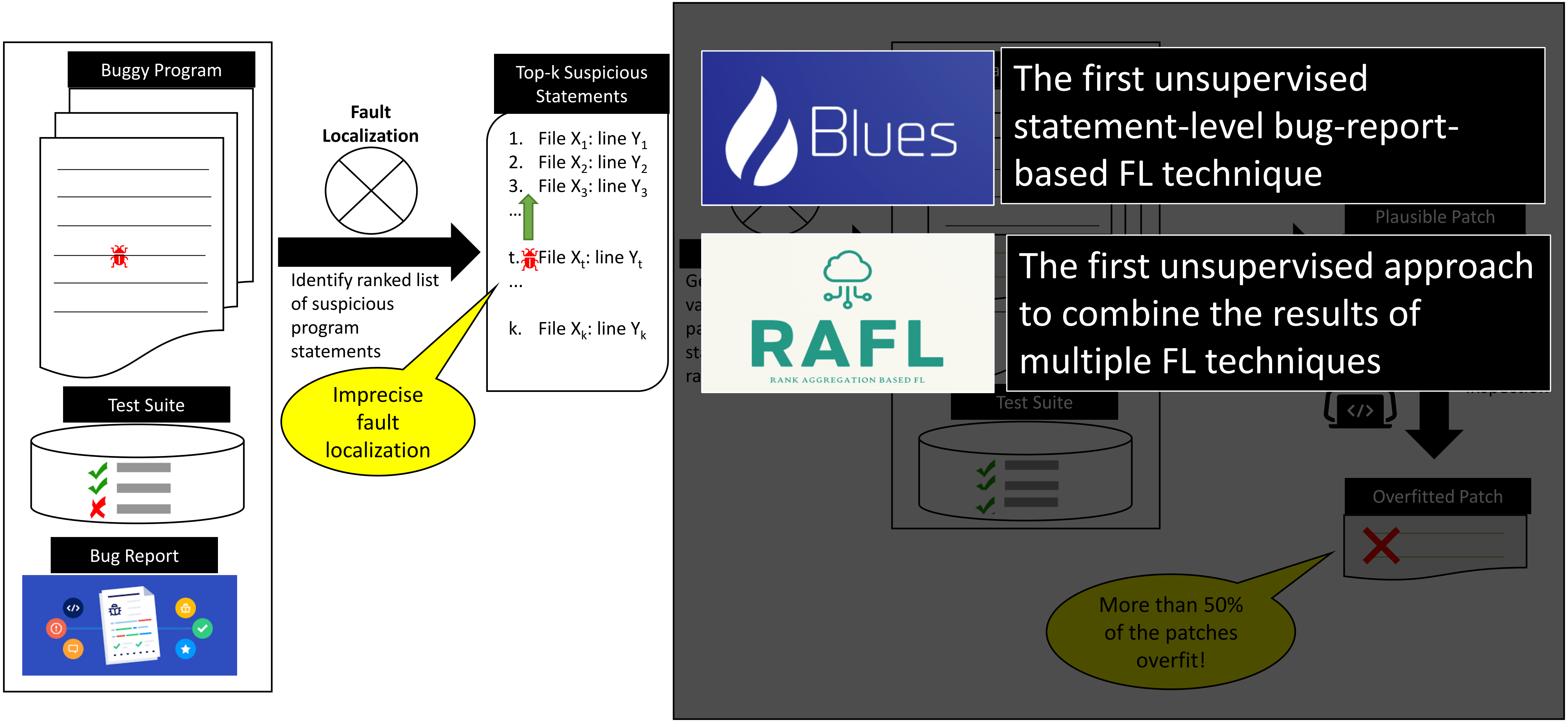


Buggy Program
Test Suite
Bug Report
Fault Localization
Identify ranked list of suspicious program statements
Imprecise fault localization
Use bug reports and test suite
Top-k Suspicious Statements
1. File X_1: line Y_1
2. File X_2: line Y_2
3. File X_3: line Y_3
...
t. File X_t: line Y_t
...
k. File X_k: line Y_k
Patch Generation and Validation
Generate and validate candidate patches for each statement in the ranked list
Patched Program
Test Suite
Patch passing all the tests
Plausible Patch
Manual inspection
Overfitted Patch
More than 50% of the patches overfit!

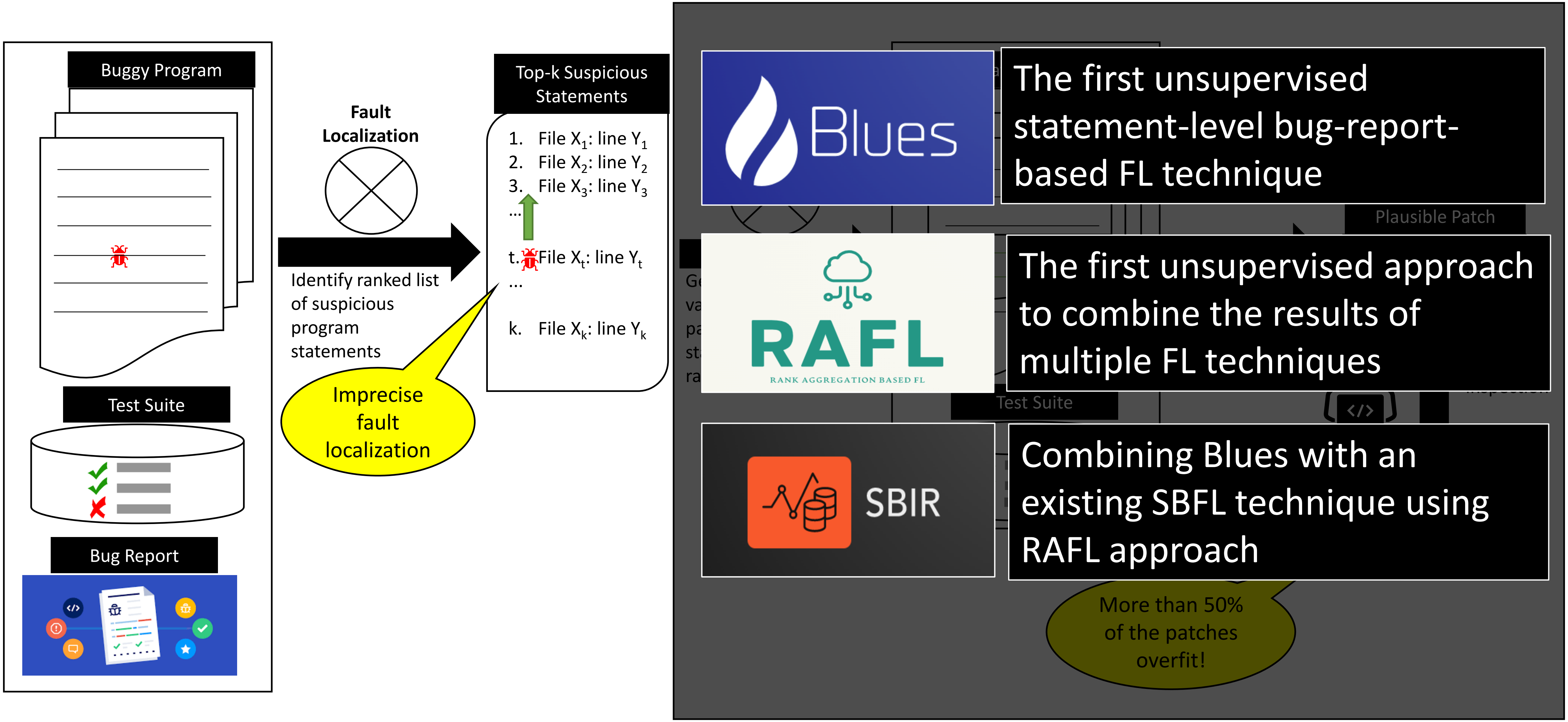
Our Solution



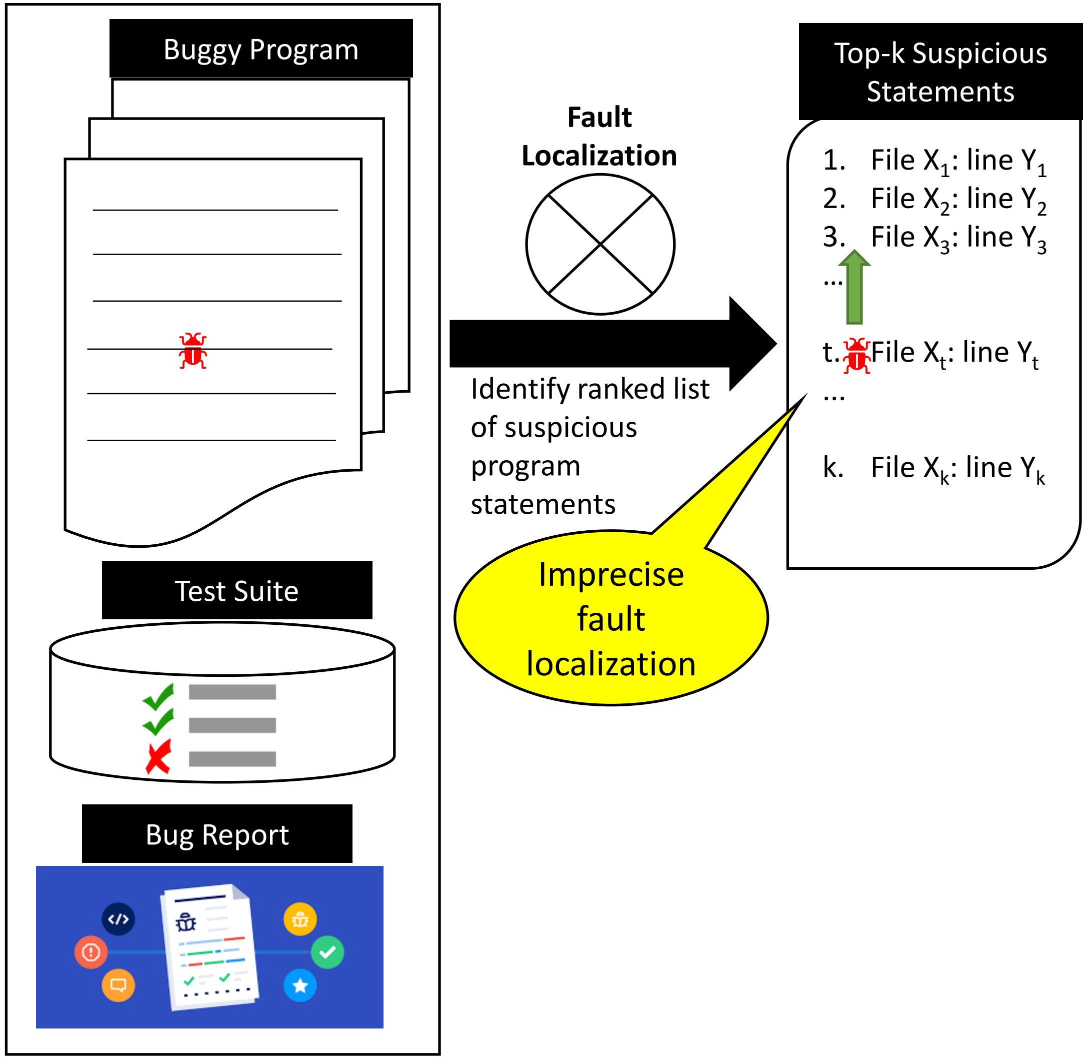
Our Solution



Our Solution



Our Solution



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

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

Heuristics-based repair technique/ High FL sensitivity

Learning-based repair technique/ Med FL sensitivity

Template-based repair technique/ Low FL-sensitivity

Arja

SequenceR

SimFix

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
- 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 techniques**[1] couldn't fix

1. Liu et al., You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems. ICST 2019.

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	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 **they** couldn't fix earlier	Lang-7 Lang-10 Lang-59 Math-35	(Uses Perfect FL)	Closure-68 Closure-92 Closure-126

1. Liu et al., You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems. ICST 2019.

Effect of Using SBIR on Repair Performance

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

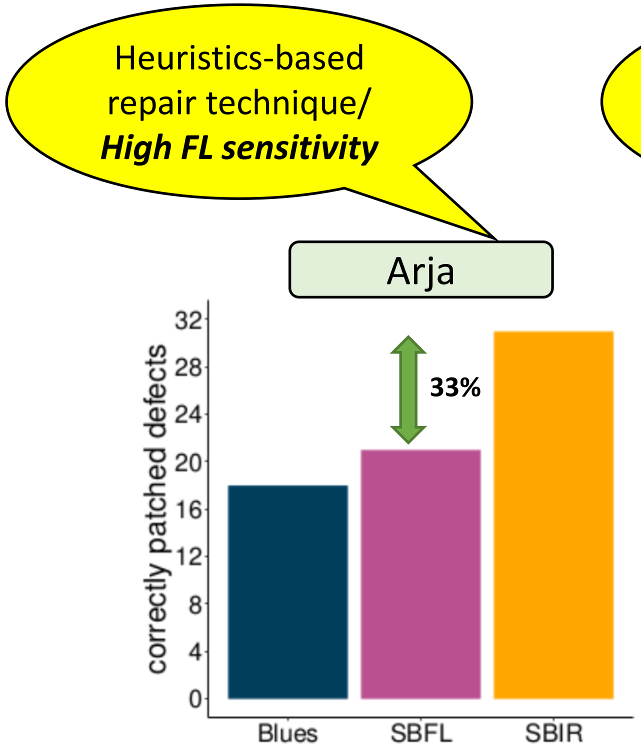
SBIR enables repair tools to correctly patch many new defects without modifying their patch generation algorithms.

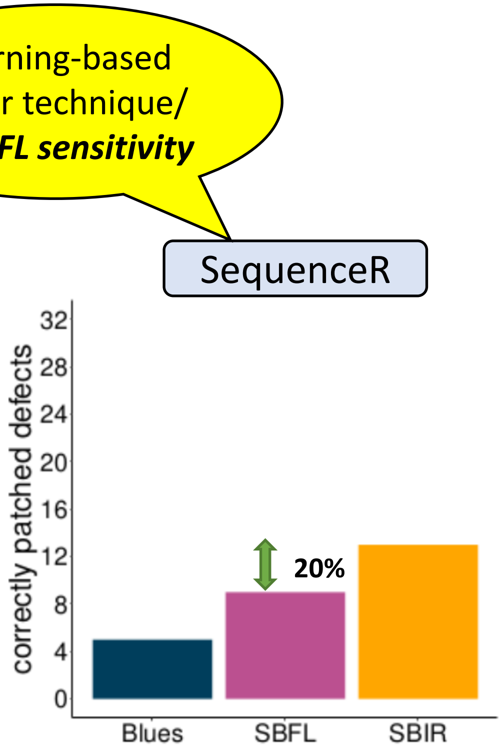
	Arja	SequenceR	SimFix
Using SBIR enables APR tools to **correctly repair** 7 new defects that **14 existing techniques[1]** couldn't fix	Chart-12 Closure-78 Closure-86 Lang-10 Lang-20	Closure-86	Closure-68 Closure-92
Using SBIR enables APR tools to **correctly repair** 7 new defects that **they** couldn't fix earlier.	Lang-7 Lang-10 Lang-59 Math-35	(Uses Perfect FL)	Closure-68 Closure-92 Closure-126

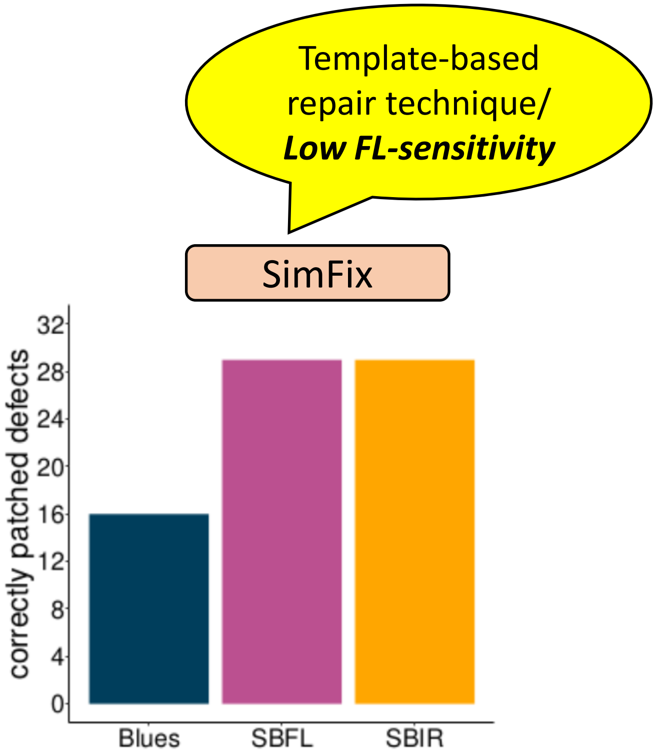
1. Liu et al., You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems. ICST 2019.

Effect of Using SBIR on Repair Quality

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





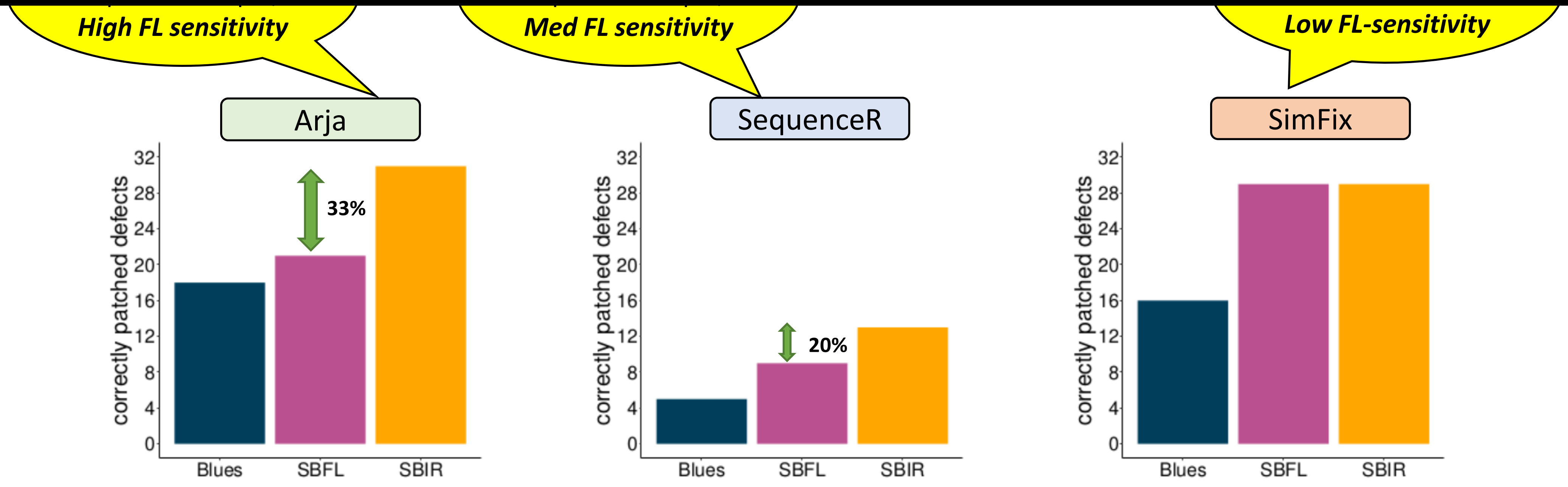


FL sensitivity computed using approach described in: "Liu et al., A critical review on the evaluation of automated program repair systems. Journal of Systems and Software, 171:110817, 2021"

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.



FL sensitivity computed using approach described in: "Liu et al., A critical review on the evaluation of automated program repair systems. Journal of Systems and Software, 171:110817, 2021"

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 ⬇			
SBFL	15	14	2
Blues	21	20	19
SBIR	**8**	**12**	**2**

Lower is better

Effect of Using SBIR on Localization Error

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

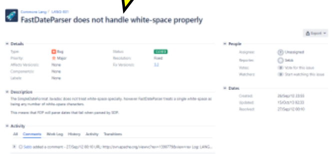
	Arja	SequenceR	SimFix
localization error assessment			
upper bound	36	24	32
⬇ # of defects not correctly patched due to localization error ⬇			
SBFL	15	14	2
Blues	21	20	19
SBIR	**8**	**12**	**2**

⬇ Lower is better

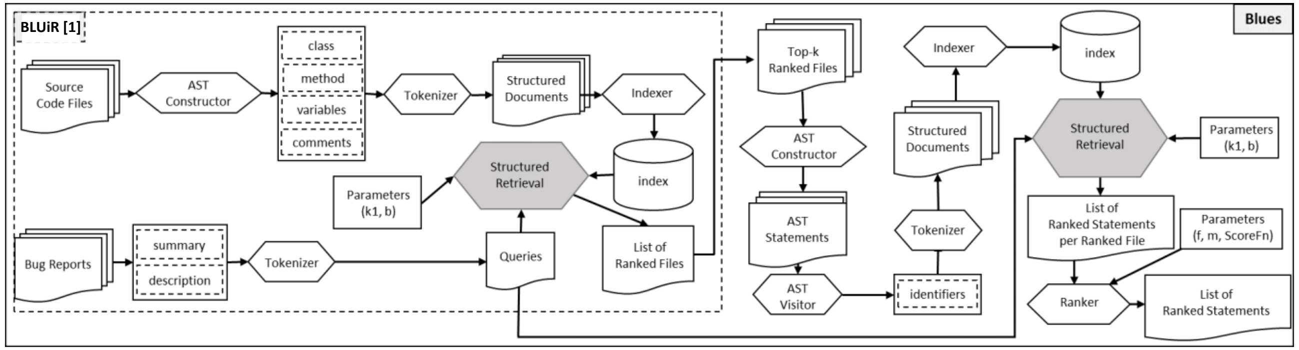
SBIR lowers the localization error by providing repair tools an earlier opportunity to patch actual buggy statements.

Blues: Unsupervised, Statement-Level FL Using Bug Reports

Bug report



Source files



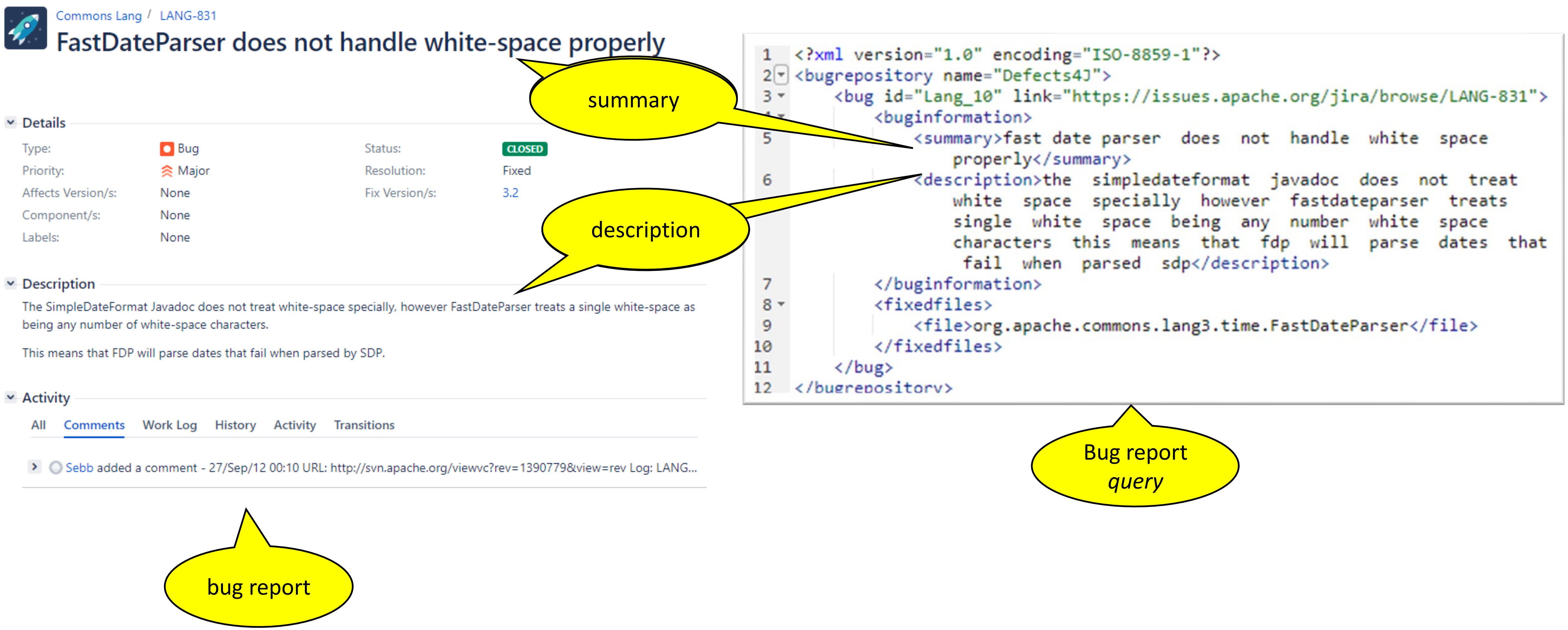
Blues

```
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
```

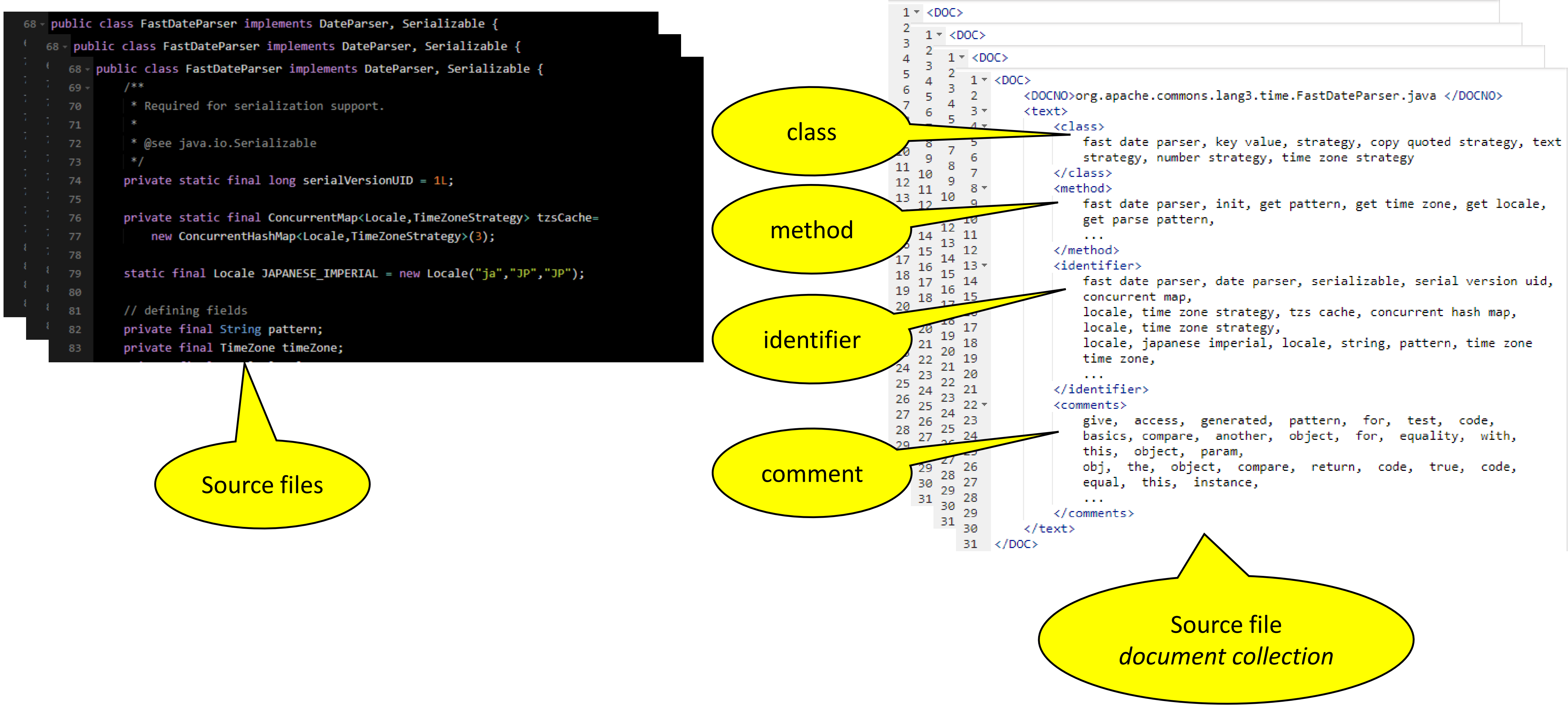
ranked list of suspicious program statements

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

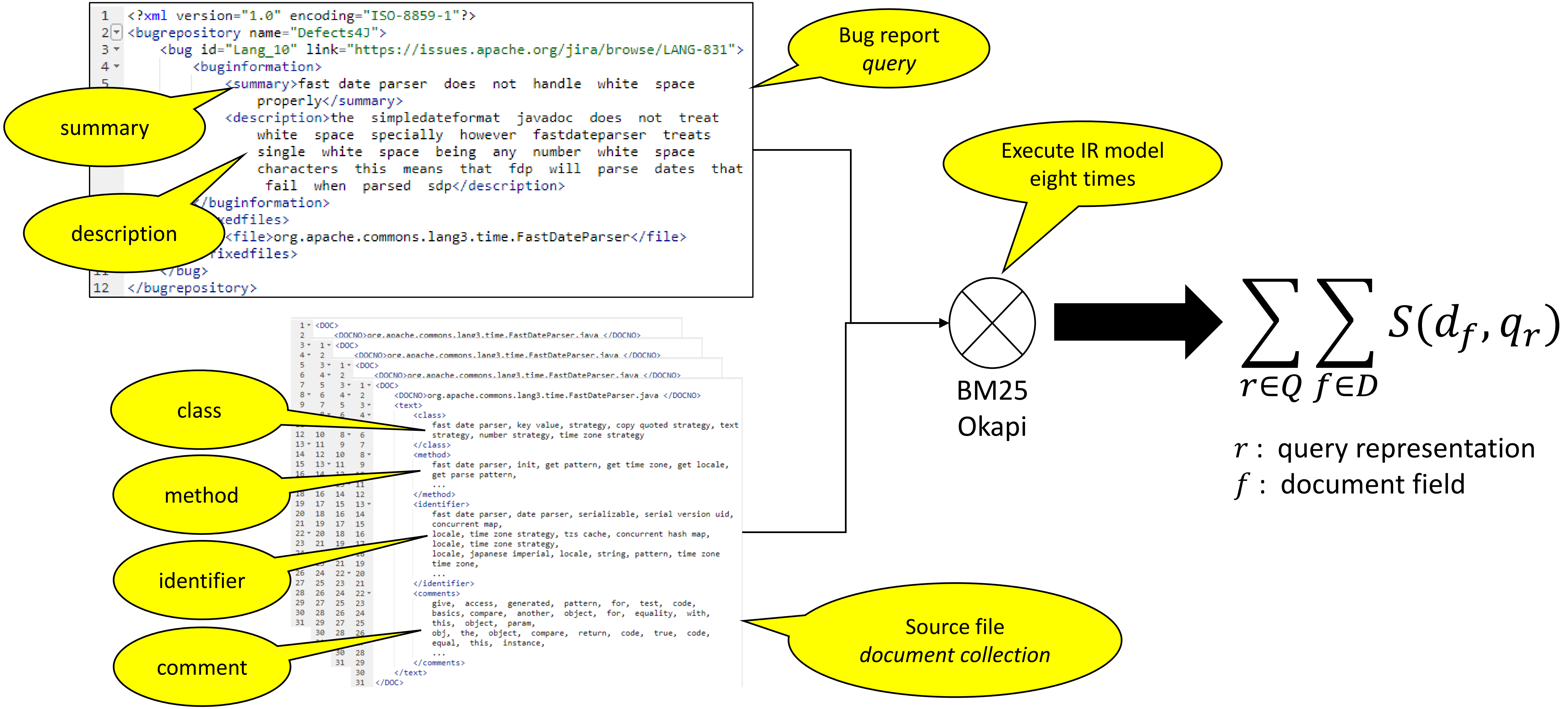
Step 1: Extract terms associated with *Summary* and *Description* from bug report



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



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



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

defect | source file | rank | suspiciousness score

```
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
```

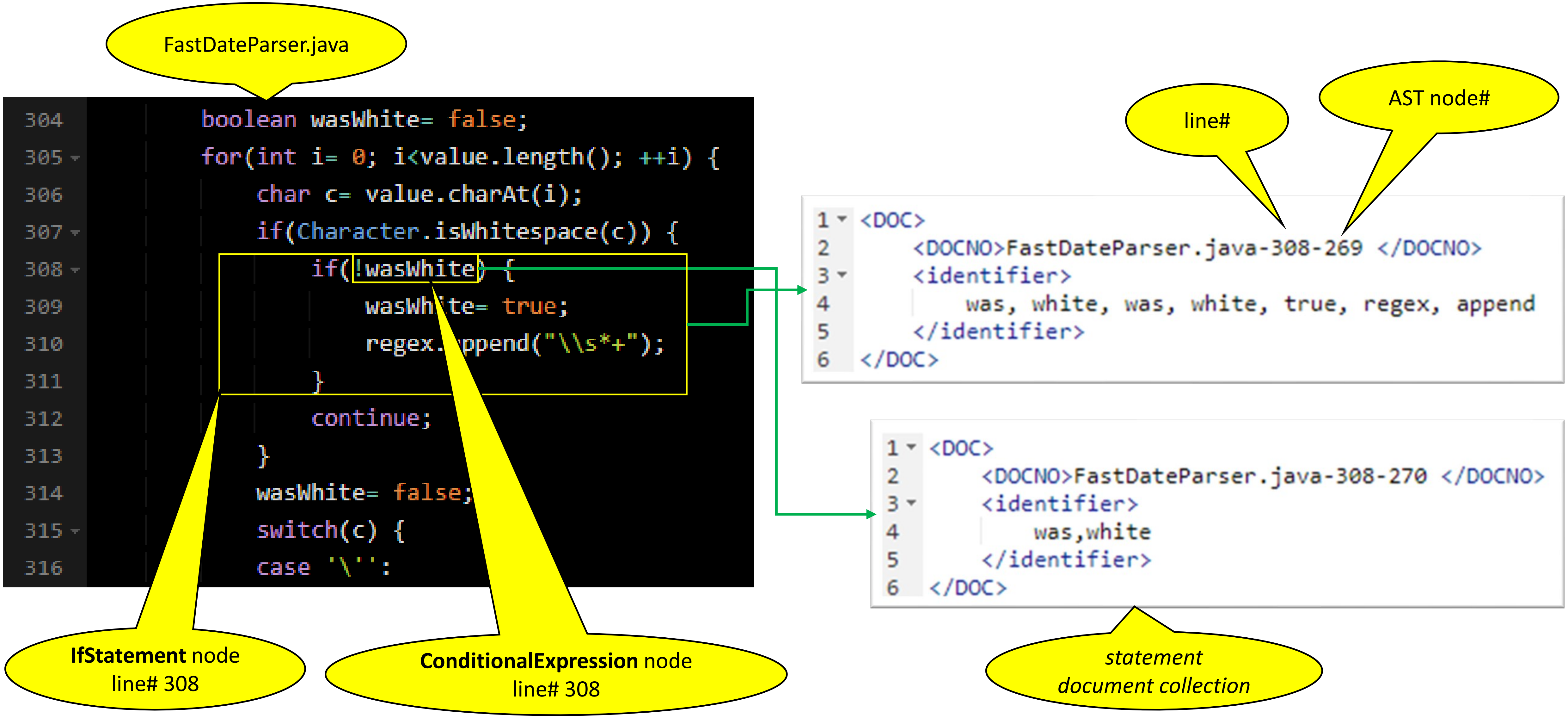
Ranked list of suspicious source files

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

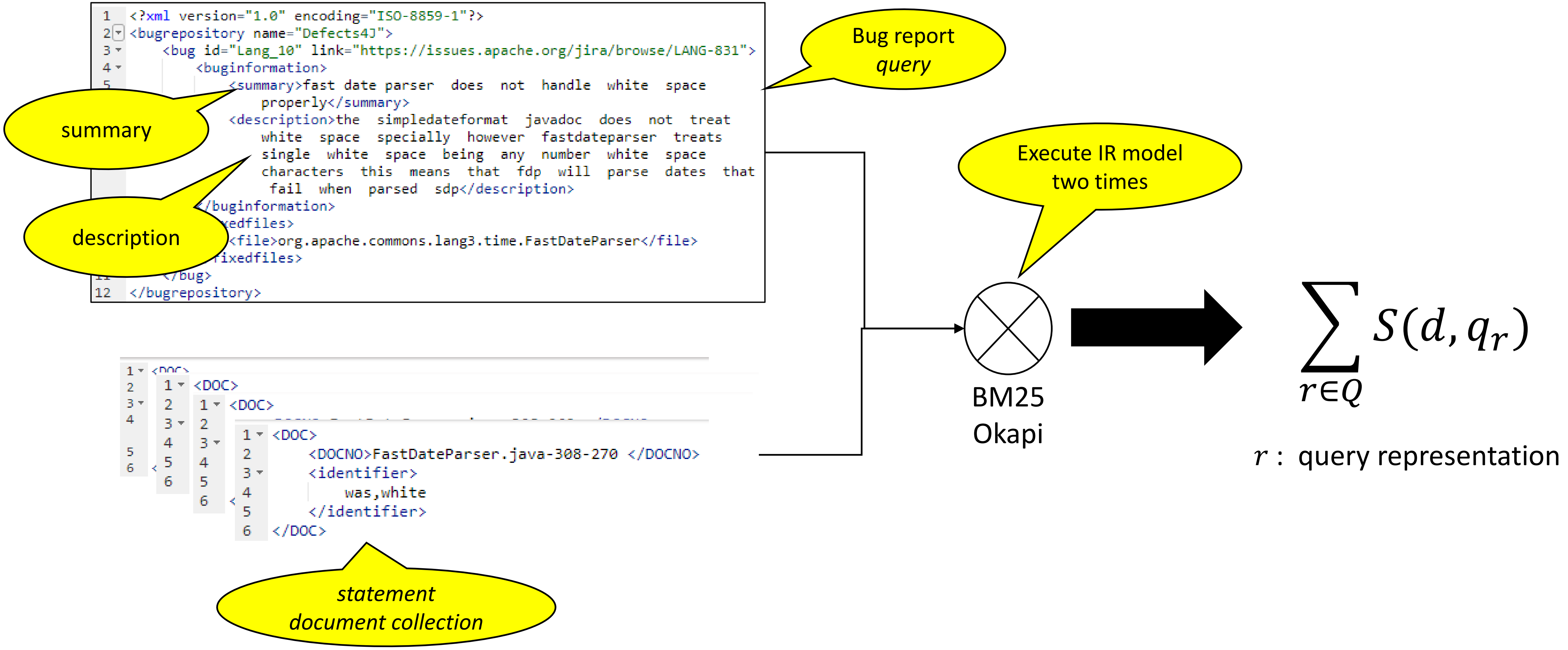
```
Lang_10 Q0 org.apache.commons.lang3.time.FastDateParser.java 1 0.403753 indri
Lang_10 Q0 org.apache.commons.lang3.time.FastDateFormat.java 2 0.225249 indri
Lang_10 Q0 org.apache.commons.lang3.time.FastDatePrinter.java 3 0.217153 indri
Lang_10 Q0 org.apache.commons.lang3.StringUtils.java 4 0.21257 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



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



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

source file | Line# | AST node# | rank | suspiciousness score

defect

```
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 statements

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
```

Ranked list of suspicious 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 statements per ranked file

Ranker — Parameters (*f, m, ScoreFn*)

```
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#353,0.861111111111111
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

statement-level IRFL technique used in iFixR repair tool

171 Lang and Math defects in Defects4J v1.0

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**

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

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

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.

statement-level IRFL technique used in iFixR repair tool

171 Lang and Math defects in Defects4J v1.0

Higher is better (hit@k) / Lower is better (EXAM)

(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**

815 defects from 17 projects in Defects4J v2.0

Higher is better (hit@k) / Lower is better (EXAM)

Does not consider suspicious file scores

(815 defects)	*hit@k*					EXAM
	k = 1	25	50	100	all	*k* = all
vanilla BLUiR	26	143	192	245	611	0.159
Blues	**27**	**184**	**241**	**306**	**611**	**0.111**

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

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.

statement-level IRFL technique used in

171 Lang and Math defects in Defects4J v1.0

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**

For scenarios relevant to APR (*k* >=25), Blues consistently outperforms the state-of-the-art and baseline.

815 defects from 17 projects in Defects4J v2.0

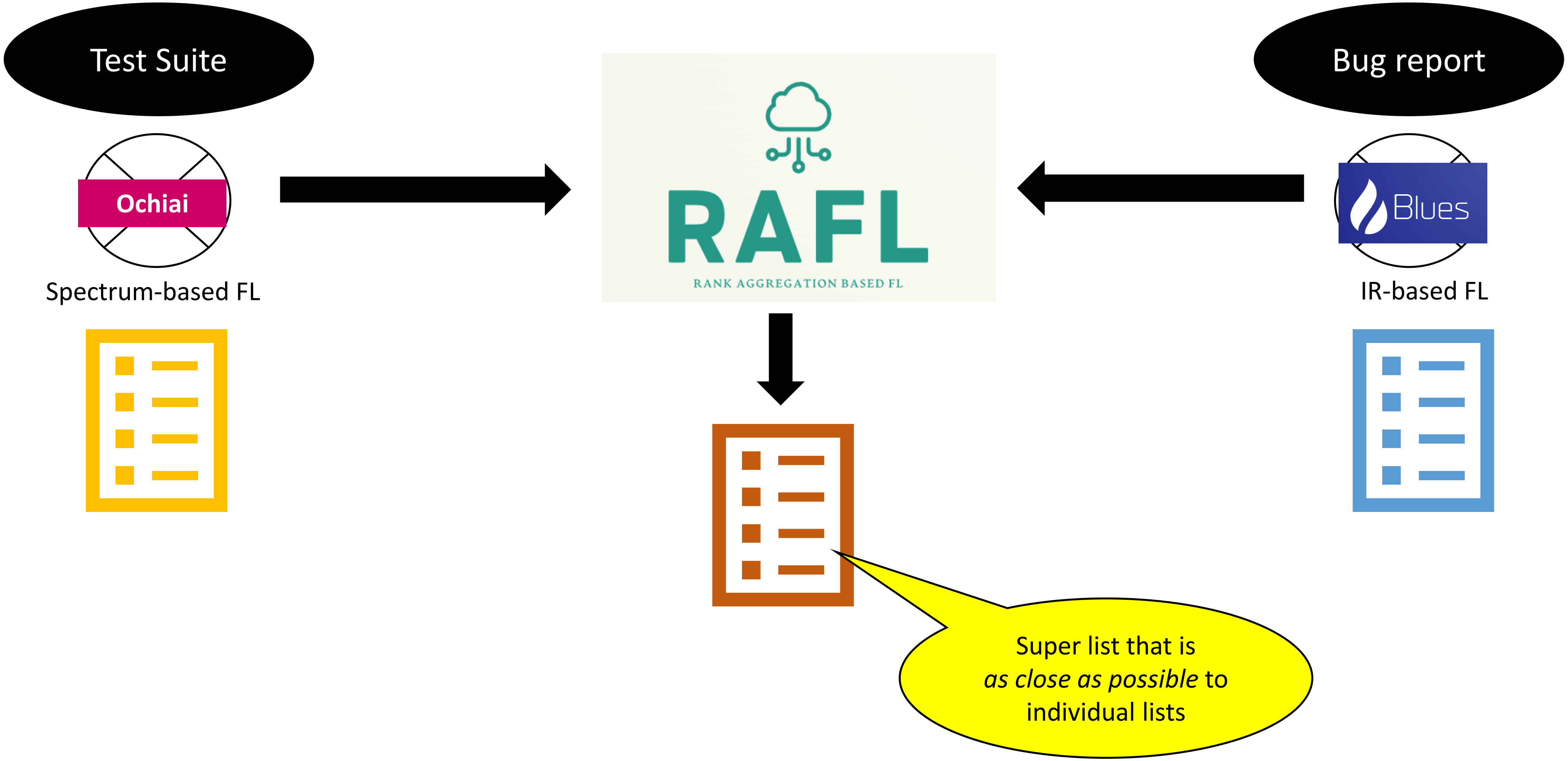
Higher is better | Lower is better

(815 defects)	*hit@k*					EXAM
	k = 1	25	50	100	all	*k* = all
vanilla BLUiR	26	143	192	245	611	0.159
Blues	**27**	**184**	**241**	**306**	**611**	**0.111**

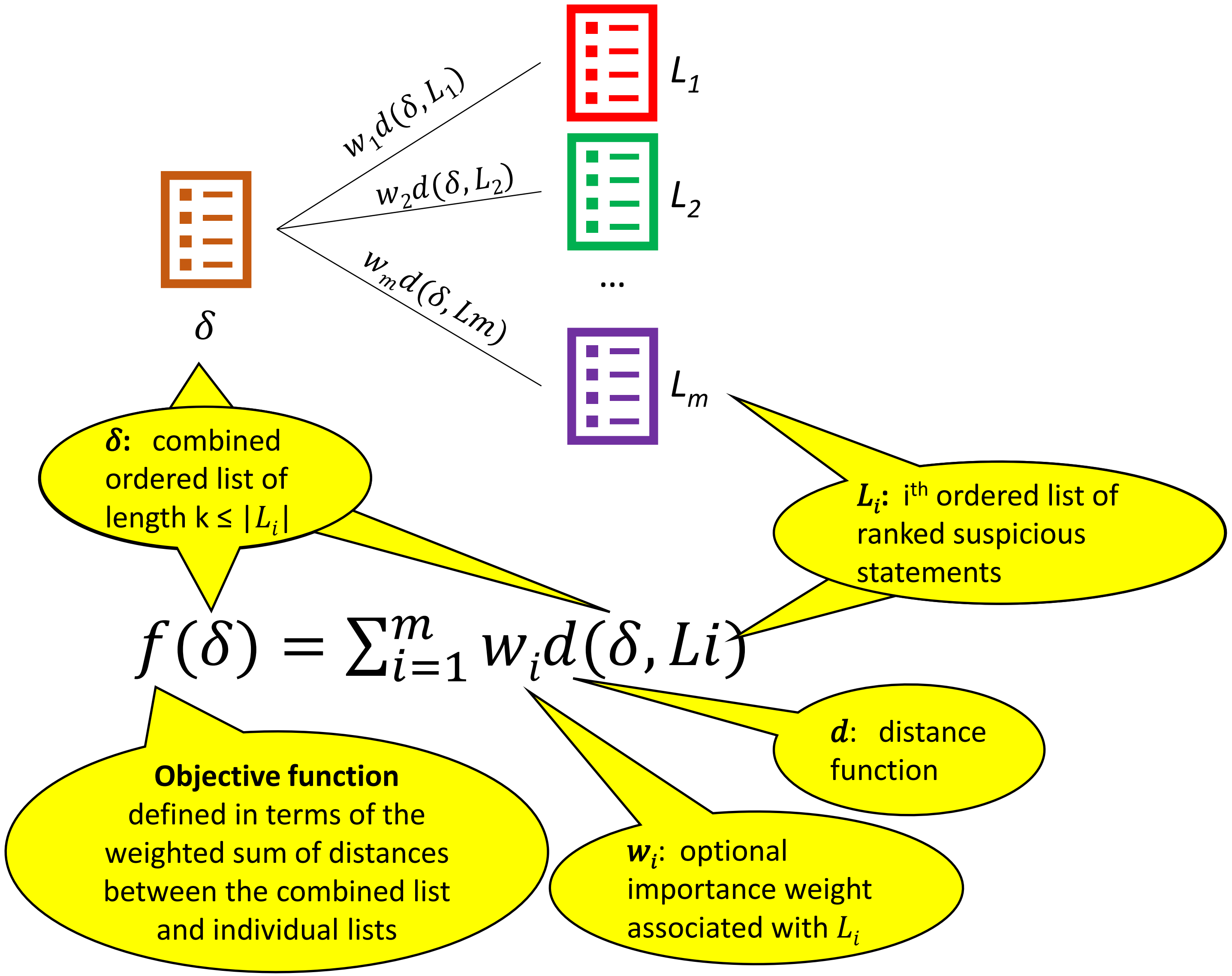
Does not consider suspicious file scores

How to Combine Multiple FL Techniques for APR?

RAFL: Rank-Aggregation-Based Fault Localization



Key Insight: Formulate Rank-Aggregation as an Optimization Problem

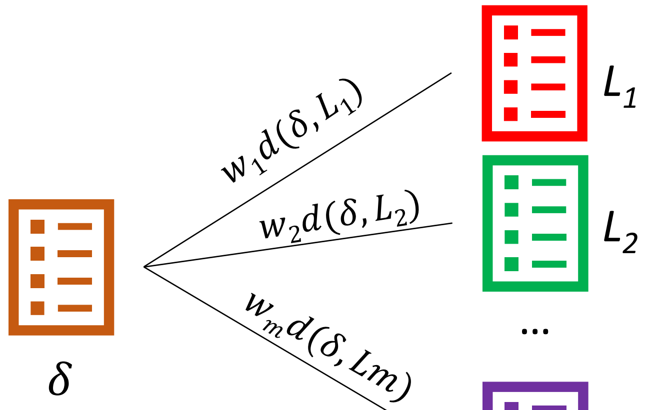


RAFL: Rank-Aggregation-Based Fault Localization

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

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

$\boldsymbol{\delta^*}$: ordered list of length k ≤ $|Li|$ that is *as close as possible* to all Li



$\boldsymbol{L_i}$: i[th] ordered list of ranked suspicious statements

$\boldsymbol{\delta}$: combined ordered list of length k ≤ $|L_i|$

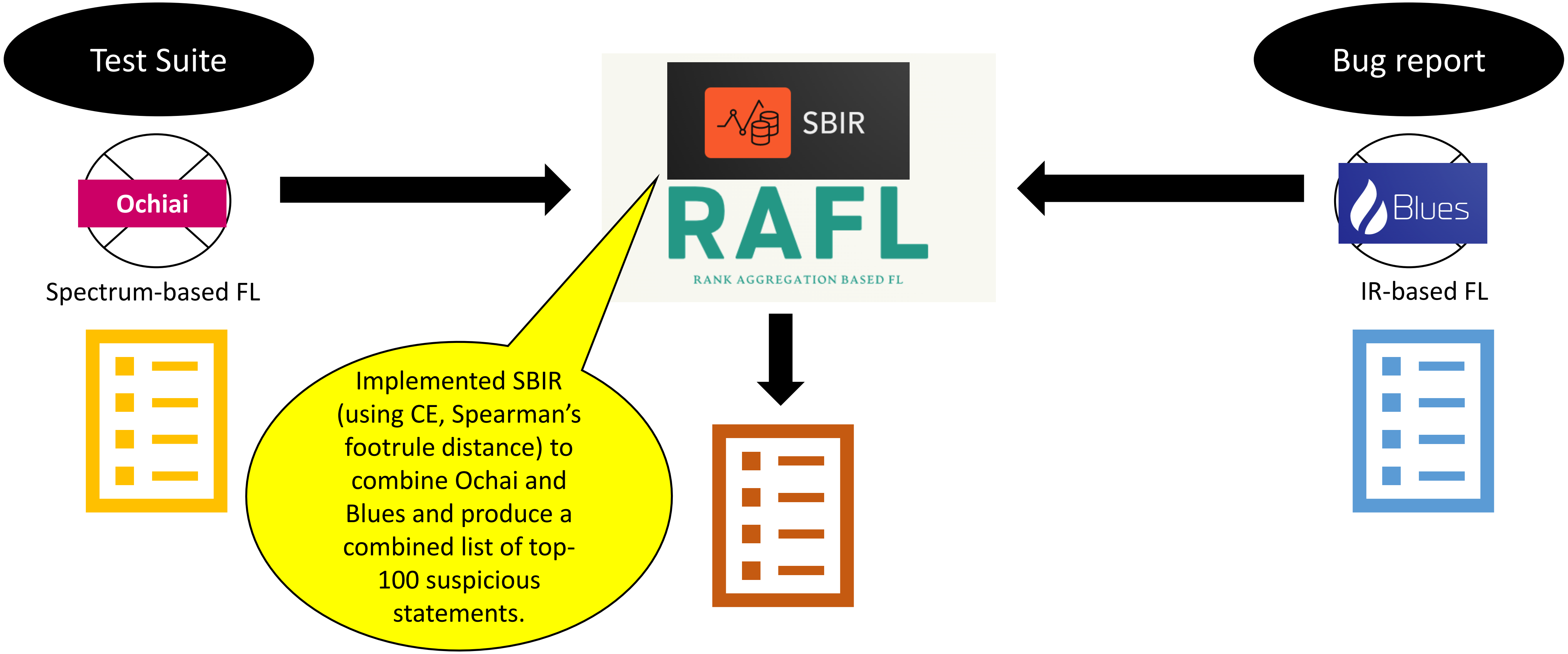
$$f(\delta) = \sum_{i=1}^{m} w_i d(\delta, Li)$$

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

$\boldsymbol{w_i}$: optional importance weight associated with L_i

$\boldsymbol{d}$: distance function

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.

Higher is better | Lower is better

(815 defects)		*hit@k*				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

$$cv = \frac{stdev}{mean}$$

< 0.1 means 10 seeds' results are tightly coupled

1. CombineFL (Zou et al., TSE, 2019); 2. NeuralBugLocator (Gupta et al., NeurIPS, 2019); 3. DeepFL (Li et al., ISSTA, 2019)

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.

Higher is better | Lower is better

(815 defects)		*hit@k*				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

$$cv = \frac{stdev}{mean}$$

< 0.1 means 10 seeds' results are tightly coupled

SBIR outperforms SBFL and Blues corroborating existing research[1,2,3] on combining FL techniques.

1. CombineFL (Zou et al., TSE, 2019); 2. NeuralBugLocator (Gupta et al., NeurIPS, 2019); 3. DeepFL (Li et al., ISSTA, 2019)

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

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

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

Higher is better (hit@k) — Lower is better (EXAM)

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

(334 defects)		*hit@k*				EXAM
family	technique	*k* = 1	25	50	100	*k* = 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

$$cv = \frac{stdev}{mean}$$

< 0.1 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)		*hit@k*				EXAM
family	technique	*k* = 1	25	50	100	*k* = 100
SBFL	Ochiai	30	168	196	221	0.254
	DStar	32	169	199	222	0.254
MBFL	Metallaxis	40	154	175	195	0.238
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	slicing-frequency	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
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

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

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.

⬆ Higher is better (for $E_{inspect}@k$) ⬇ Lower is better (for $EXAM_{inspect}$)

(815 defects)		$E_{inspect}@k$				$EXAM_{inspect}$
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

1. CombineFL (Zou et al., TSE, 2019); 2. Fluccs (Sohn et al., ISSTA, 2017); 3. Trapt (Li et al., OOPSLA, 2017); 4. Savant (Le et al., OOPSLA, 2016);

Performance of SBIR Compared to the Supervised Combining Method

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

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

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

Higher is better / Lower is better

(815 defects)		$E_{inspect}@k$				$EXAM_{inspect}$
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

$$cv = \frac{stdev}{mean}$$

< 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].

1. CombineFL (Zou et al., TSE, 2019); 2. Fluccs (Sohn et al., ISSTA, 2017); 3. Trapt (Li et al., OOPSLA, 2017); 4. Savant (Le et al., OOPSLA, 2016);

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	SequenceR	SimFix
Using SBIR enables APR tools to **correctly repair** 7 new defects that **14 existing techniques[1]** couldn't fix	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 **they** couldn't fix earlier	Lang-7 Lang-10 Lang-59 Math-35	(Uses Perfect FL)	Closure-68 Closure-92 Closure-126

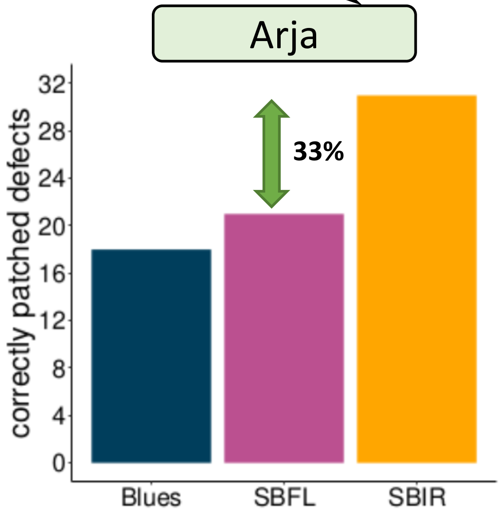
1. Liu et al., You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems. ICST 2019.

Effect of Using SBIR on Repair Quality

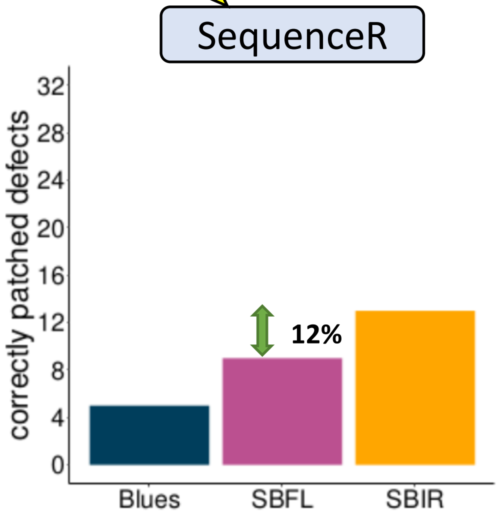
689 single file edit defects from 17 real-world Java projects available in Defects4J v2.0

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

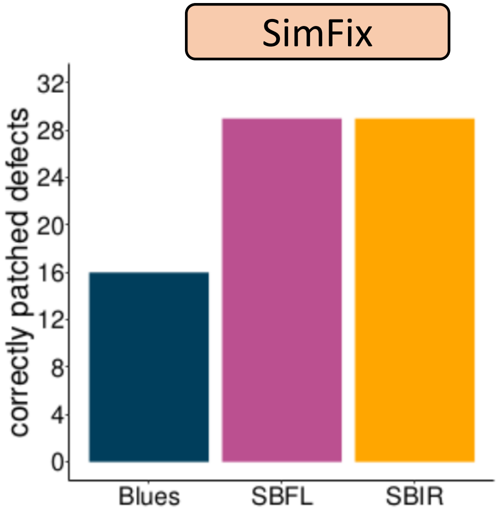
High FL sensitivity



Med FL sensitivity



Low FL-sensitivity



FL sensitivity computed using approach described in: "Liu et al., A critical review on the evaluation of automated program repair systems. Journal of Systems and Software, 171:110817, 2021"

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 ↓			
SBFL	15	14	2
Blues	21	20	19
SBIR	**8**	**12**	**2**

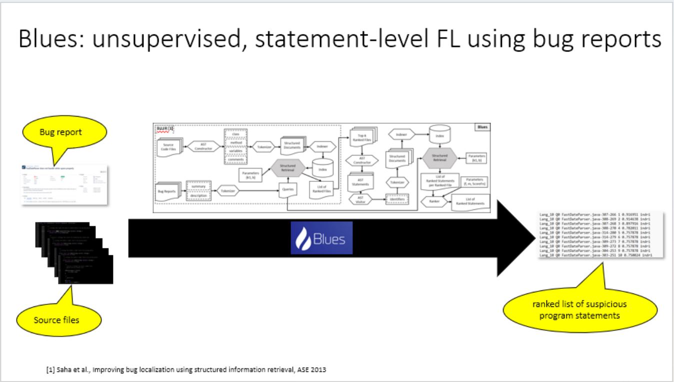
Lower is better

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



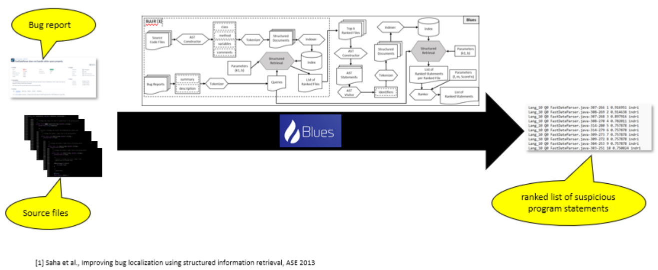
https://github.com/LASER-UMASS/Blues

scan me for paper

Contributions

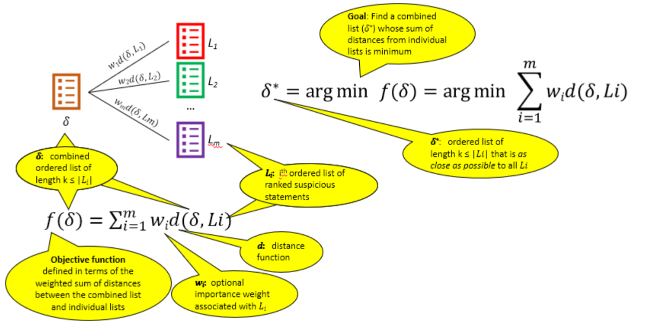
scan me for artifact

Blues: unsupervised, statement-level FL using bug reports



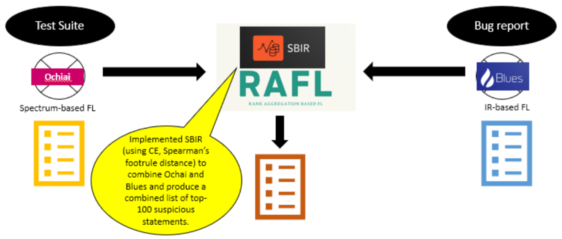
https://github.com/LASER-UMASS/Blues

RAFL: Rank-Aggregation-Based Fault Localization



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

SBIR: combining SBFL with Blues using RAFL

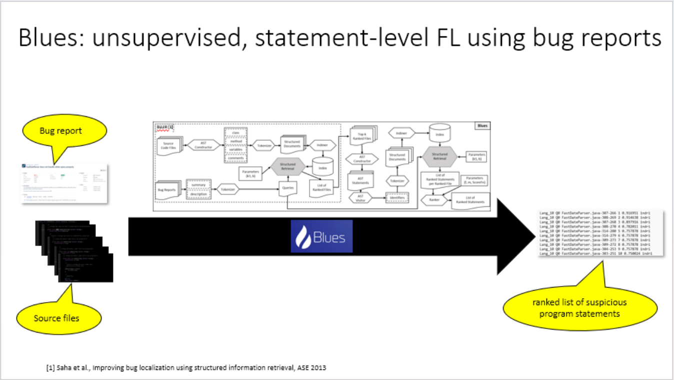


https://github.com/LASER-UMASS/SBIR-ReplicationPackage

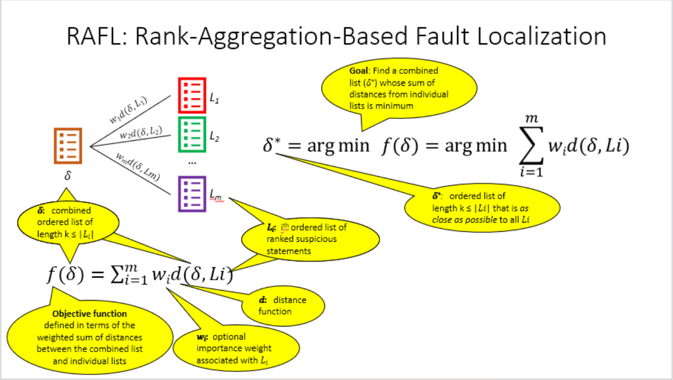
scan me for paper

Contributions

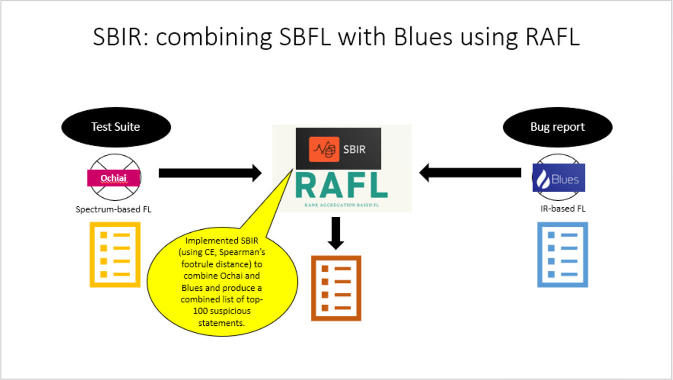
scan me for artifact



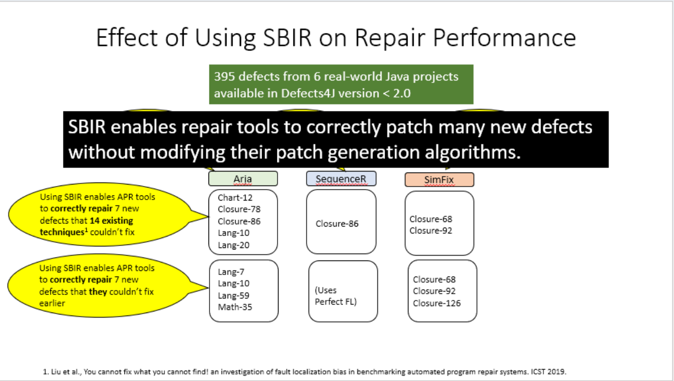
https://github.com/LASER-UMASS/Blues

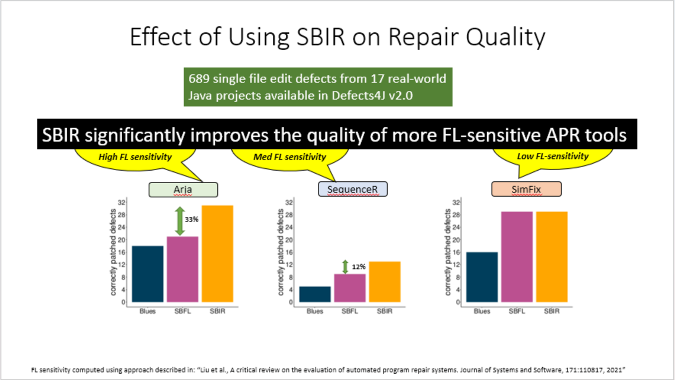


https://github.com/LASER-UMASS/RAFL



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





Effect of Using SBIR on Localization Error

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

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

Lower is better

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

https://manishmotwani3.github.io/