A Case Against Coverage-Based Program Spectra

Péter Attila Soha Tamás Gergely Ferenc Horváth
Béla Vancsics Árpád Beszédes

University of Szeged, Hungary

ICST'23, Dublin, Ireland

Where is the Bug?

(given computation impact and test case outcomes)

```java
1  public class Circle {
2    private double area;
3    private double perimeter;
4    public Circle(double radius) {
5       area = radius * radius * Math.PI;
6       perimeter = radius * Math.PI;
7    }
8    double getArea() {
9       return area;
10   }
11   double getPerimeter() {
12      return perimeter;
13   }
14 }
```

impact?	5	6	9	12	pass/fail
t1	yes	**no**	yes	no	✓
t2	no	**yes**	no	yes	✗
t3	no	**no**	no	yes	✓

```java
public class CircleTest extends TestCase {
  static Circle circle = new Circle(0);
  public void t1() {
    assertEquals(Math.PI, new Circle(1).getArea(), 1e-10);
  }
  public void t2() {
    assertEquals(2.0*Math.PI, new Circle(1).getPerimeter(), 1e-10);
  }
  public void t3() {
    assertEquals(0, circle.getPerimeter(), 1e-10);
  }
}
```

"Program Slicing" + "Spectrum Based Fault Localization"

Spectrum-Based Fault Localization
(SBFL)

- P : **program** under investigation
- T : set of **test cases** for testing P
- E : set of **code elements** in P (e.g., methods, statements, branches)
- M : (binary or integer) **spectrum matrix** of size $|T| \times |E|$
  - $m_{i,j}$: dynamic relationship between test t_i and element e_j
- R : (binary) **results vector** of size $|T|$
  - $r_i = 0$ iff t_i completed without failure
- F : (binary) **faults vector** of size $|E|$ (used when evaluating SBFL effectiveness)
  - $f_j = 1$ iff e_j element contains a fault

M	e_1	…	$e_{\|E\|}$
t_1	0/1	0/1	0/1
t_2	0/1	0/1	0/1
…	0/1	0/1	0/1
$t_{\|T\|}$	0/1	0/1	0/1

R
0/1
0/1
0/1
0/1

F	0/1	0/1	0/1

The Spectrum Matrix
(dynamic relationship between test cases and code elements)

In the example above

- $m'_{i,j} = 1$: when executing test t_i element e_j **influences** the test case outcome r_i
  - *Backward Dynamic Program Slice (with test output as the slicing criterion)*
  - Denoted by M' in the following

"slice-based spectrum"

Traditional approach

- $m_{i,j} = 1$: when executing test t_i element e_j is **covered** (exercised)
  - *Hit-Based Spectrum (the traditional coverage-based spectrum)*
  - Denoted by M in the following

"coverage-based spectrum"

(Other definitions exist, e.g. count-based spectrum)

SBFL Approach
Spectrum metrics

Statistical counts for each $e \in E$

- ef (ef') : number of failing tests that cover (are influenced by) e in M and M', respectively
- nf (nf') : failing but not covering (not influenced)
- ep (ep') : passing and covering (influenced)
- np (np') : passing but not covering (not influenced)

SBFL Approach
Formulas

SBFL formulas

- Calculate a **suspiciousness score** for each program element, which is then used to **rank the elements** to aid automated debugging
- Examples (same for both M and M' matrix types):

$$\frac{ef}{ef + ep}$$

Barinel

$$\frac{ef^2}{ep + nf}$$

DStar

$$\frac{ef}{\sqrt{(ef + nf) \cdot (ef + ep)}}$$

Ochiai

SBFL Approach

On the example

$$\frac{ef}{ef + ep}$$

M′	5	6	9	12	R
t1	1	0	1	0	0
t2	0	1	0	1	1
t3	0	0	0	1	0
ef′	0	1	0	1	
ep′	1	0	1	1	
nf′	1	0	1	0	
np′	1	2	1	1	
Barinel′	0	**1**	0	0.5	
F	0	**1**	0	0	

Where is the Bug? (reprise)

(this time using the coverage-based matrix)

M	5	6	9	12	R
t1	1	1	1	0	0
t2	1	1	0	1	1
t3	0	0	0	1	0
ef	1	1	0	1	
ep	1	1	1	1	
nf	0	0	1	0	
np	1	1	1	1	
Barinel	**0.5**	**0.5**	0	**0.5**	?
F	0	1	0	0	

$$\frac{ef}{ef + ep}$$

Comparing Slice and Coverage

```java
1  public class Circle {
2    private double area;
3    private double perimeter;
4    public Circle(double radius) {
5       area = radius * radius * Math.PI;
6       perimeter = radius * Math.PI;
7    }
8    double getArea() {
9       return area;
10   }
11   double getPerimeter() {
12      return perimeter;
13   }
14 }
```

M′	5	6	9	12	R
t1	1	**0**	1	0	0
t2	**0**	1	0	1	1
t3	0	0	0	1	0

M	5	6	9	12	R
t1	1	**1**	1	0	0
t2	**1**	1	0	1	1
t3	0	0	0	1	0

```java
1  public class CircleTest extends TestCase {
2    static Circle circle = new Circle(0);
3    public void t1() {
4      assertEquals(Math.PI, new Circle(1).getArea(), 1e-10);
5    }
6    public void t2() {
7      assertEquals(2.0*Math.PI, new Circle(1).getPerimeter(), 1e-10);
8    }
9    public void t3() {
10     assertEquals(0, circle.getPerimeter(), 1e-10);
11   }
12 }
```

So, why is everybody still using coverage-based spectrum?

- Spoiler: **coverage is trivial** to compute, and slicers are ~~complex~~ ~~imprecise~~ ~~costly~~ *non-existent*
- Also, some concepts are more **complex with slicing** (e.g. what is the “test output” used for the slicing criterion?)
- **Are people ignorant** too?

Goal

Short and longer term research objectives

How **big is the handicap** of coverage-based SBFL compared to the (theoretically precise) slice-based one?

1. **Theoretical analysis**

2. **Empirical investigation**

Theoretical analysis

Assuming a perfect slicer, how coverage and slice-based ranks will compare?

Additional notations

- $C(t) \subseteq E$: a row of M, i.e. set of covered elements by test t
- $DS(t) \subseteq E$: a row of M', i.e. backward dynamic slice for test t

Assumptions

- $|F| = 1$, and we denote the faulty element by f and all other elements by n
- $\forall t \in T$ are associated with exactly one slicing criterion (rows of M and M' are compatible)
- $\forall t \in T : |C(t)| > 0$ and $|DS(t)| > 0$
- $\forall t \in T : R(t) = 1 \Rightarrow M(t, f) = 1$ (faulty element is covered by all failing tests)
- $\forall t \in T : R(t) = 1 \Rightarrow M'(t, f) = 1$ (faulty element contributes to the slicing criterion in all failing tests)
- $\forall t \in T : DS(t) \subseteq C(t) \subseteq E$ (slice is meaningful)

Theoretical analysis

Average slice size

- How much are slices more precise than the coverage?
- And, what is the impact of this on the SBFL performance?
- We use the **average slice size** as a proxy to the probability that a covered code element e is also in the slice:

$$p = \frac{\sum_{t \in T} \frac{|DS(t)|}{|C(t)|}}{|T|}$$

Theoretical analysis
Spectrum metrics

Observations

- For f, $ef' = ef$ and $nf' = nf$ (because each failing test's slice includes f)
- But $ep' = p \cdot ep$ and $np' = (1 - p) \cdot ep + np$
- For all other non-faulty elements n, all four metrics are scaled with p

The effect on SBFL formulas

- Since ef is often in the numerator, the **score** in the slice-based spectrum will be typically **higher** for the **faulty element**, and **lower** for the **non-faulty**

Theoretical analysis

Suspiciousness formulas

- We analyzed several formulas (including *Barinel*, *Tarantula*, *Ochiai*, *Dstar*, etc.) and we showed that:

$$S'(f) \geq S(f) \text{ and } S'(n) \leq S(n)$$

for any formula S score

- In other words, coverage-based SBFL **will necessarily produce worse ranking** than slice-based one
- Also, the difference is (inversely) **determined by p**

$$Bar'(f) = \frac{ef}{ef + p \cdot ep} \geq Bar(f)$$

$$Bar'(n) = \frac{p \cdot ef}{p \cdot ef + p \cdot ep} = Bar(n)$$

Empirical investigation

Case study

- Stock tools for coverage-based spectra
- *Slicer4J* dynamic slicer tool*
- Slice-based spectrum: one row **for each assert** in each test case which were then merged (slicing criteria: asserted value)
- Subject program: *Time* from *Defects4J*
- 26 bugs with about 14k statements and 4k tests
- Analyzed 9 bugs in details (after excluding others due to various reasons)

* K. Ahmed, M. Lis, and J. Rubin, “Slicer4j: A dynamic slicer for java,” in Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ser. ESEC/FSE 2021. ACM, 2021, p. 1570–1574.

Empirical investigation

Results

- Many problems with slicing results
  - E.g. non-covered elements in the slice (we exclude them)
- **RQ1:** Average slice size (p) is
  - around **45%** (with respect to coverage)
- **RQ2:** Avg. **ranks** of faulty elements in slice-based SBFL **notably better**
  - Barinel (43 → 10), DStar (21 → 9), Ochiai (20 → 10), etc.

Empirical investigation

Qualitative results

- **RQ3:** Manual examination of each bug to find out the reasons for the differences
  - Many **superfluous elements** in the coverage
  - In some cases, the **results were the same**
  - In some cases, **coverage-based results were better** because the slicer did not include some statements in the slice (errors with passing tests)
  - Due to the slicer's imperfections some of the assumptions were not met (e.g. slice **did not reach** the buggy element)

Take away messages

1. **Coverage-based SBFL is a big over-approximation of slice-based SBFL**
   - Coverage-based SBFL necessarily produces worse ranking than slice-based SBFL
   - The rate of imprecision (slice size over coverage size) severely influences the formula performance
2. **Theory vs. practice**
   - Assumptions not always met, but case study supported the theory even with an imperfect slicer
   - Coverage is a more simpler concept, e.g. no need for a slicing criterion
3. **The area needs more research!**
   - The SBFL community may not be aware of why coverage is a bad proxy for slice???
   - E.g. experiment with hybrid slicing algorithms

https://slicefl.github.io/

M′	5	6	9	12	R
t1	1	**0**	1	0	0
t2	**0**	1	0	1	1
t3	0	0	0	1	0

M	5	6	9	12	R
t1	1	**1**	1	0	0
t2	**1**	1	0	1	1
t3	0	0	0	1	0

$S'(f) \geq S(f)$ **and** $S'(n) \leq S(n)$

for any formula S score

- **RQ1:** Average slice size (p) is around 45%
- **RQ2:** Barinel (43 → 10), DStar (21 → 9)
- **RQ3:** Manual examination of each bug
- **Research encouraged in the topic!**