uni.lu UNIVERSITÉ DU LUXEMBOURG

Serval securityandtrust.lu

PayPal

SnT securityandtrust.lu

What Made This Test Flake? Pinpointing Classes Responsible for Test Flakiness

International Conference on Software Maintenance and Evolution, 3-7 October, 2022


Sarra Habchi


Guillaume Haben


Jeongju Sohn


Adriano Franci


Mike Papadakis


Maxime Cordy


Yves Le Traon

Outline

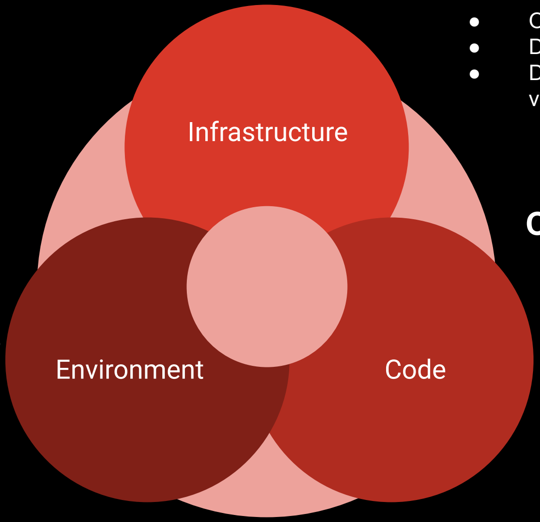
- Background on flakiness research
- Motivation
- Data collection
- Research Questions
- Conclusion

What is a flaky test?

> *“ A test that **passes** and **fails** for the same version of a program ”*

Environment

- Hardware
- External services
- Time of the day



Infrastructure

- Overloaded CI
- Differences in Testing/Production
- Different Operating Systems, versions...

Code

- Concurrency issues
- Usage of Date / Time
- Test order dependencies
- I/O (File, database)
- Network calls
- Asynchronous waits

Flakiness research focus

ML based
Vocabulary, test smells, FlakeFlagger, code metrics, runtime metrics, heart beat, Flakify…

Detection techniques

Tools
Deflaker, iDFlakies, Shaker…

Causes / Prevalence

Open source / Industrial context

Empirical studies

Java, Python

Fixing techniques

Flakiness research focus

Limits of detection techniques

ML-based accuracy ranges from ~70 to ~90%

Flakiness research focus

Limits of detection techniques

When a test is said to be flaky, what's next?


```python
def test_idle(self, updater, caplog):
    updater.start_polling(0.01)
    Thread(target=partial(self.signal_sender, updater=updater)).start()

    with caplog.at_level(logging.INFO):
        updater.idle()



    for idx, log in enumerate(caplog.records):
        if log.getMessage().startswith('Error while getting Updates: Conflict'):
            caplog.records.pop(idx)  # For stability
            assert len(caplog.records) == 2, caplog.records

    rec = caplog.records[-2]
    assert rec.getMessage().startswith(f'Received signal {signal.SIGTERM}')
    assert rec.levelname == 'INFO'

    rec = caplog.records[-1]
    assert rec.getMessage().startswith('Scheduler has been shut down')
    assert rec.levelname == 'INFO'

    # If we get this far, idle() ran through
    sleep(0.5)
    assert updater.running is False
```

Flakiness research focus

Limits of fixing techniques

- Often target only one category of flakiness

  e.g. order dependencies (ODRepair, iFixFlakies)

  randomness (Flex)

- Many prevalent categories (Async waits, concurrency) are not addressed

Motivation

Help developers find components responsible for flakiness in production code

- ~20% flakiness originates from CUT, nonetheless important to fix
- Non-specific to a category of flakiness
- Retarget Fault Localisation techniques to detect “flaky” components

A word on Spectrum-Based Fault Localization

SBFL is used to **find buggy elements** (statements, lines, methods…)

SBFL calculates the **suspiciousness score** based on **coverage matrix** and PASS / FAIL results of **tests**

Several formulas have been introduced like Tarantula, Ochiai, DStar…

SBFL gives you a **ranked list** of statements

		t1	t2	susp
1	`float max(float a, float b){`			
2	`if (Float.isNaN(a))`	●	●	0.5
3	`return b;`		●	0.0
4	`else if (Float.isNaN(b))`	●		1.0
5	`return b; // return a;`	●		1.0
6	`else`	●		1.0
7	`return Math.max(a,b);`	●		1.0
8	`}`	✗	✓	

Table I: SFL formulas [23]. For DStar we set the exponent * to 2, as recommended by Wong et al. [24].

Tarantula [25]:	$S(s) = \frac{failed(s)/totalfailed}{failed(s)/totalfailed + passed(s)/totalpassed}$
Ochiai [26]:	$S(s) = \frac{failed(s)}{\sqrt{totalfailed \cdot (failed(s) + passed(s))}}$
DStar [24]:	$S(s) = \frac{failed(s)^{*}}{passed(s) + (totalfailed - failed(s))}$
Op2 [27]:	$S(s) = failed(s) - \frac{passed(s)}{totalpassed + 1}$
Barinel [28]:	$S(s) = 1 - \frac{passed(s)}{passed(s) + failed(s)}$

Data collection

Looking for flakiness-fixing commits: flaky tests with corresponding "flaky" class

1. **Search** Look for commits in large Java projects containing *flaky* keyword
2. **Inspect** Limit to atomic commits fixing CUT (*fix, repair, patch* keywords)
3. **Coverage** Build, run the test suite and get the coverage matrix for all tests
4. **Extract** Flaky test, "flaky" class, coverages information, cause of flakiness

Data collection

TABLE I: Collected Data. *ffc:* number of flakiness-fixing commits. *all:* number of commits in the project.

Proj.	#Commits		#Tests		#Classes	
	ffc	all	min - max	avg	min – max	avg
Hbase	8	18,990	138 - 2,089	1,113	734 – 1366	1053.4
Ignite	14	27,903	15 - 1,018	174	72 – 1767	1262.3
Pulsar	10	8,516	194 - 1,326	626	171 – 422	259.7
Alluxio	3	32,560	315 - 694	473	131 – 817	360.3
Neo4j	3	71,824	21 - 5,782	2,139	40 – 1663	581.3
Total	38		15 - 5,782	905	40 – 1767	820.2

Research questions

- **RQ1** Are SBFL-based approaches effective in identifying flaky classes?
- **RQ2** How do code and change metrics contribute to the identification of flaky classes?
- **RQ3** How can ensemble learning improve the identification of flaky classes?
- **RQ4** How does an SBFL-based approach perform for different flakiness categories?

RQ1 Are SBFL-based approaches effective in identifying flaky classes?

SBFL gives a ranked list of classes to inspect.
1st most suspicious, last least suspicious.

Evaluation metrics

Accuracy *acc@n*: # flaky classes ranked in the top n

Wasted effort *wef*: # classes inspected before reaching the flaky class

Baseline *Rwef*: Relative effort wrt # covered classes. Between 0 and 100.

RQ1 Are SBFL-based approaches effective in identifying flaky classes?

We adapt Spectrum-Based Fault Localization for flakiness

For each class, we compute:

e_f : # flaky test executing the class
e_s : # stable test executing the class
n_f : # flaky test not executing the class
n_s : # stable test not executing the class

TABLE II: SBFL formulae adapted to flakiness.

Name	Formula
Ochiai [42]	$\frac{e_f}{\sqrt{(e_f+n_f)(e_f+e_s)}}$
Barinel [43]	$1 - \frac{e_s}{e_s+e_f}$
Tarantula [44], [45]	$\frac{\frac{e_f}{e_f+n_f}}{\frac{e_f}{e_f+n_f}+\frac{e_s}{e_s+n_s}}$
DStar [34]	$\frac{e_f^*}{e_s * n_f}$

We use Genetic Programming to evolve a new formula combining the existing ones

RQ1 Are SBFL-based approaches effective in identifying flaky classes?

TABLE V: RQ1: The effectiveness of GP evolved formulæ using Ochiai, Barinel, Tarantula, and DStar.

Project	**Total**	**acc**				**wef (R_{wef})**	
		@1	@3	@5	@10	mean	med
Hbase	8	1	4	5	5	13.12 (16)	2.5 (5)
Ignite	14	0	3	3	5	214.93 (21)	20.0 (4)
Pulsar	10	3	5	6	9	9.20 (23)	3.0 (9)
Alluxio	3	0	0	0	1	101.67 (65)	86.0 (83)
Neo4j	3	1	2	2	2	23.33 (43)	1.0 (18)
Total	38	5	14	16	22	94.24 (26)	6.5 (8)
Percentage (%)	100	**13**	**37**	42	**58**	-	-

RQ2 How do code and change metrics contribute to the identification of flaky classes?

TABLE III: Code and change metrics used to augment SBFL.

	Metric	Definition
Flakiness	#TOPS	Number of time operations performed by the class.
	#ROPS	Number of calls to the `random()` method in the class.
	#IOPS	Number of input/output operations performed by the class.
	#UOPS	Number of operations performed on unordered collections by the class.
	#AOPS	Number of asynchronous waits in the class.
	#COPS	Number of concurrent calls in the class.
	#NOPS	Number of network calls in the class.
Change	Changes	Number of unique changes made on the class.
	Age	Time interval to the last changes made on the class.
	Developers	Number of developers contributing to the class.
Size	LOC	The number of lines of code.
	CC	Cyclomatic complexity.
	DOI	Depth of inheritance.

RQ2 How do code and change metrics contribute to the identification of flaky classes?

TABLE VI: RQ2: The contribution of flakiness, change, and size metrics to the identification of flaky classes.

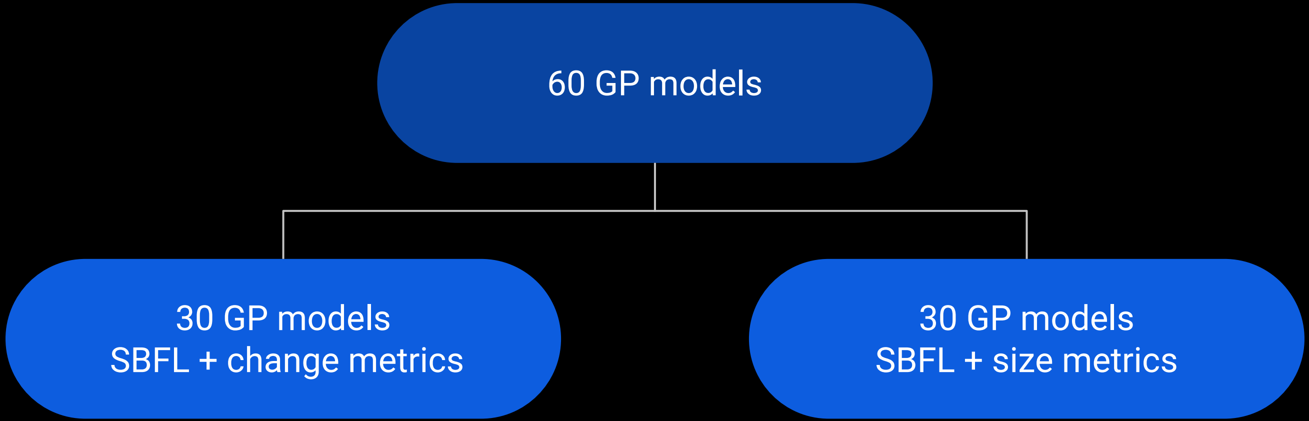
Proj. (#)	**SBFL & flakiness**						**SBFL & change**						**SBFL & size**					
	acc				wef (R_{wef})		acc				wef (R_{wef})		acc				wef (R_{wef})	
	@1	@3	@5	@10	mean	med	@1	@3	@5	@10	mean	med	@1	@3	@5	@10	mean	med
Hbase (8)	1	4	5	5	11.9 (12)	3 (4)	2	4	4	5	16.9 (13)	4 (4)	2	4	5	5	11.4 (12)	3 (3)
Ignite (14)	0	2	2	4	230.9 (26)	63 (4)	2	4	4	4	222.3 (24)	18 (4)	1	3	3	5	220.1 (24)	43 (4)
Pulsar (10)	2	5	6	8	10.2 (15)	3 (8)	3	5	7	9	8.0 (12)	2 (5)	2	5	7	9	6.9 (13)	2 (6)
Alluxio (3)	0	0	1	1	97.7 (51)	73 (65)	0	0	1	1	75.7 (49)	94 (39)	0	0	1	1	90.7 (49)	77 (58)
Neo4j (3)	1	2	2	2	19.3 (42)	1 (18)	2	2	2	2	6.7 (37)	0 (9)	2	2	2	2	23.0 (40)	0 (10)
Total (38)	4	13	16	20	99.5 (24)	8 (8)	9	15	18	21	94.1 (21)	5 (6)	7	14	18	22	94.3 (22)	5 (7)
Percentage (%)	**11**	34	42	**53**	-	-	**24**	39	47	55	-	-	**18**	37	47	58	-	-

Adding flakiness metrics to SBFL does not improve results

On the contrary, we see some improvements with change and size metrics

RQ3 How can ensemble learning improve the identification of flaky classes?

Ensemble learning via voting



RQ3 How can ensemble learning improve the identification of flaky classes?

TABLE VIII: RQ3: The effectiveness of the voting between 60 different GP-evolved models, 30 from SBFL with change metrics, and 30 from using SBFL with size metrics. 'Perc' denotes Percentage

Project	**Total**	**acc**				**wef ($\mathbf{R}_{wef}$)**	
		@1	@3	@5	@10	mean	med
Hbase	8	3	5	6	6	9.62 (12)	1.5 (2)
Ignite	14	2	4	4	4	228.61 (24)	17.5 (4)
Pulsar	10	3	6	7	9	7.30 (12)	2.0 (5)
Alluxio	3	1	1	1	2	61.83 (22)	9.0 (10)
Neo4j	3	1	2	2	2	19.67 (42)	1.0 (18)
Total	38	10	18	20	23	94.61 (19)	3.5 (5)
Perc (%)	100	26	**47**	53	61	-	-

RQ4 How does an SBFL-based approach perform for different flakiness categories?

TABLE IX: RQ4: The effectiveness per flakiness category

Flakiness Category	**acc** @1	@3	@5	@10	**wef** ($\mathbf{R}_{wef}$) mean	med
Concurrency (16)	6 (**38**)	7 (44)	7(44)	8 (**50**)	147.53 (27)	9.5 (9)
Async wait (10)	3 (**30**)	6 (60)	8 (80)	8 (**80**)	21.05 (8)	1.5 (3)
Ambiguous (4)	1 (25)	2 (50)	2 (50)	3 (75)	18.88 (5)	3.5 (5)
Time (3)	0 (0)	0 (0)	0 (0)	1 (**33**)	88.33 (16)	14.0 (10)
Network (2)	0 (0)	2 (100)	2 (100)	2 (100)	1.00 (10)	1.0 (10)
Unordered collections (2)	0 (0)	1 (50)	1(50)	1 (50)	331.5 (33)	331.5 (33)
I/O (1)	0 (0)	0 (0)	0(0)	0 (**0**)	12.50 (3)	12.5 (3)
Random (1)	0 (0)	1 (100)	1 (100)	1 (100)	2.00 (75)	2.0 (75)
Total (39[4])	10	18	20	23	94.47 (19)	3.5 (5)
Perc (%)	26	47	53	61	-	-

Conclusion

- We need to further help developers deal with flakiness
- We propose an approach using SBFL to find components in the code causing flakiness
- Ensemble learning gives the best results with ~50% flaky classes identified in the top 3