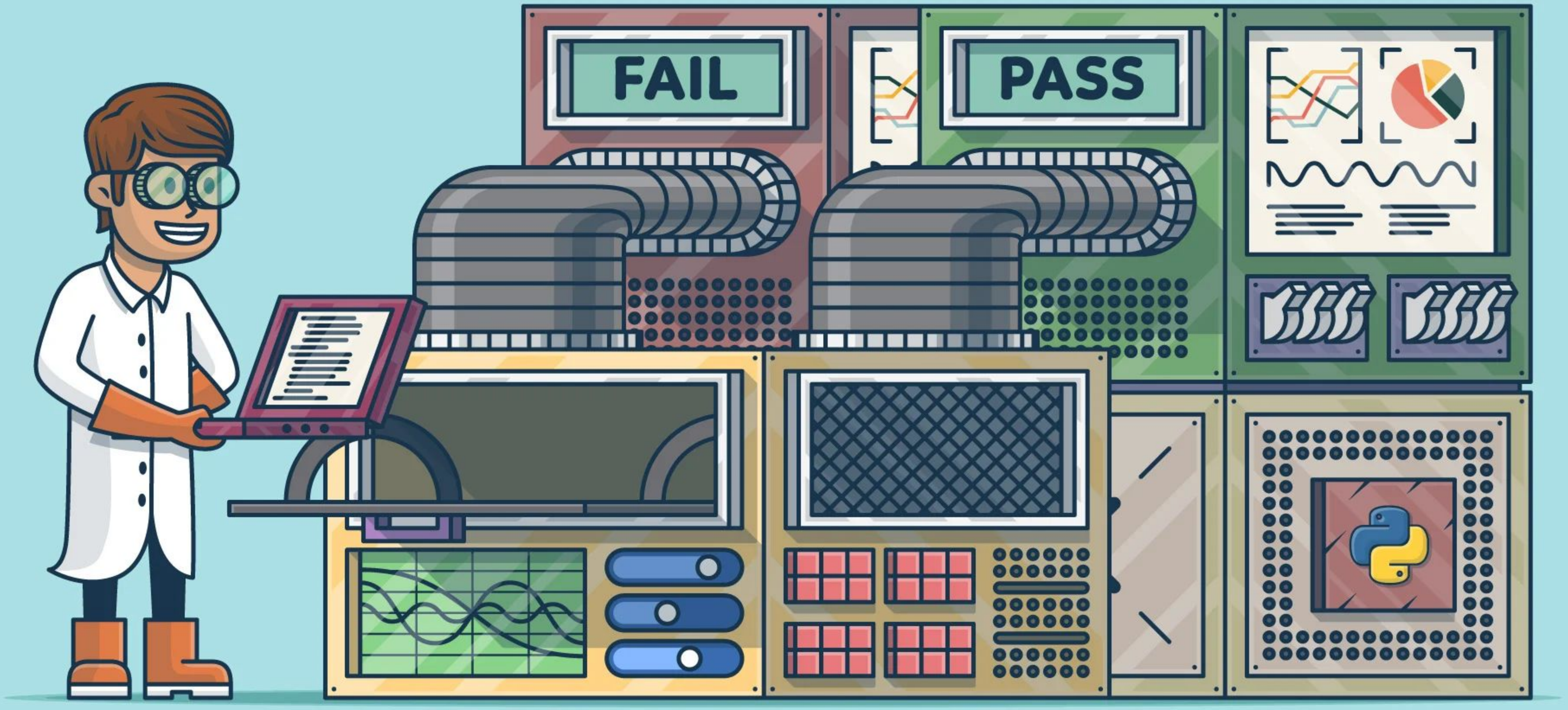
A Qualitative Study on the Sources, Impacts, and Mitigation Strategies of Flaky Tests

Sarra Habchi, <u>Guillaume Haben</u>, Mike Papadakis, Maxime Cordy, Yves Le Traon

ICST 2022



Real Python

```
Assert.AreEqual(
    GetTimeOfDay(),
    "Morning" );
```

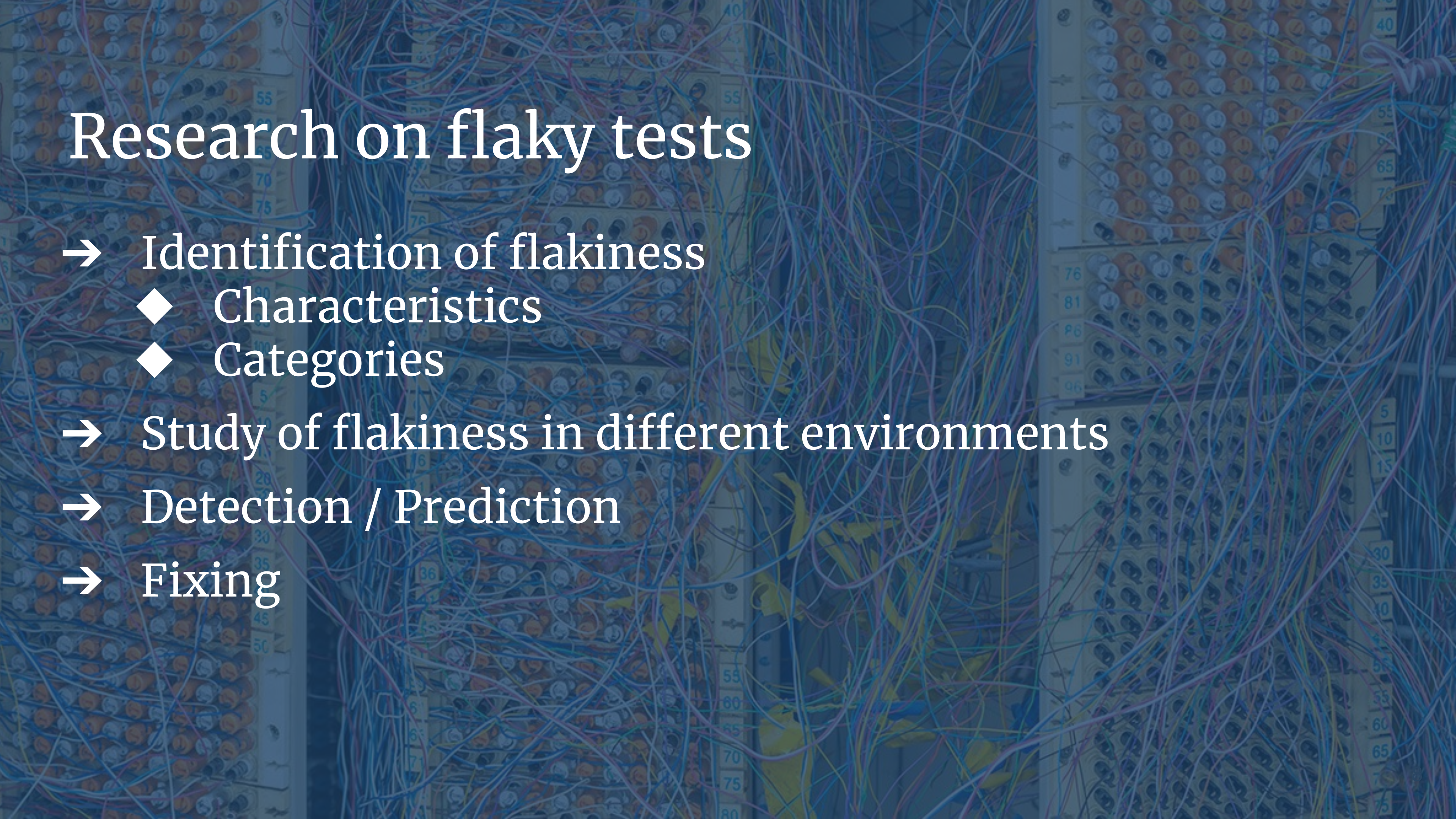
Flakiness

Flaky tests exhibit both a passing and a failing result with the same code.

- → Crippled CI
- → Wasted resources
- → Unreliable test suite

Google

- 16% of tests exhibit some flakiness.
- 2-16% of the testing budget is dedicated to rerunning flaky tests.



Research on flaky tests

- ➔ Identification of flakiness
  - ◆ Characteristics
  - ◆ Categories
- ➔ Study of flakiness in different environments
- ➔ Detection / Prediction
- ➔ Fixing

How do practitioners perceive and mitigate flakiness?

Grey Literature Review

- Mitigation strategies
- Identify gaps

Practitioner interviews

- Semi-structured
- 14 participants
- Diverse roles & companies

Analytical categories

- Sources
- Impacts
- Strategies
- Challenges encountered by practitioners

⇒ Automation opportunities

Sources

→ Test

→ CUT

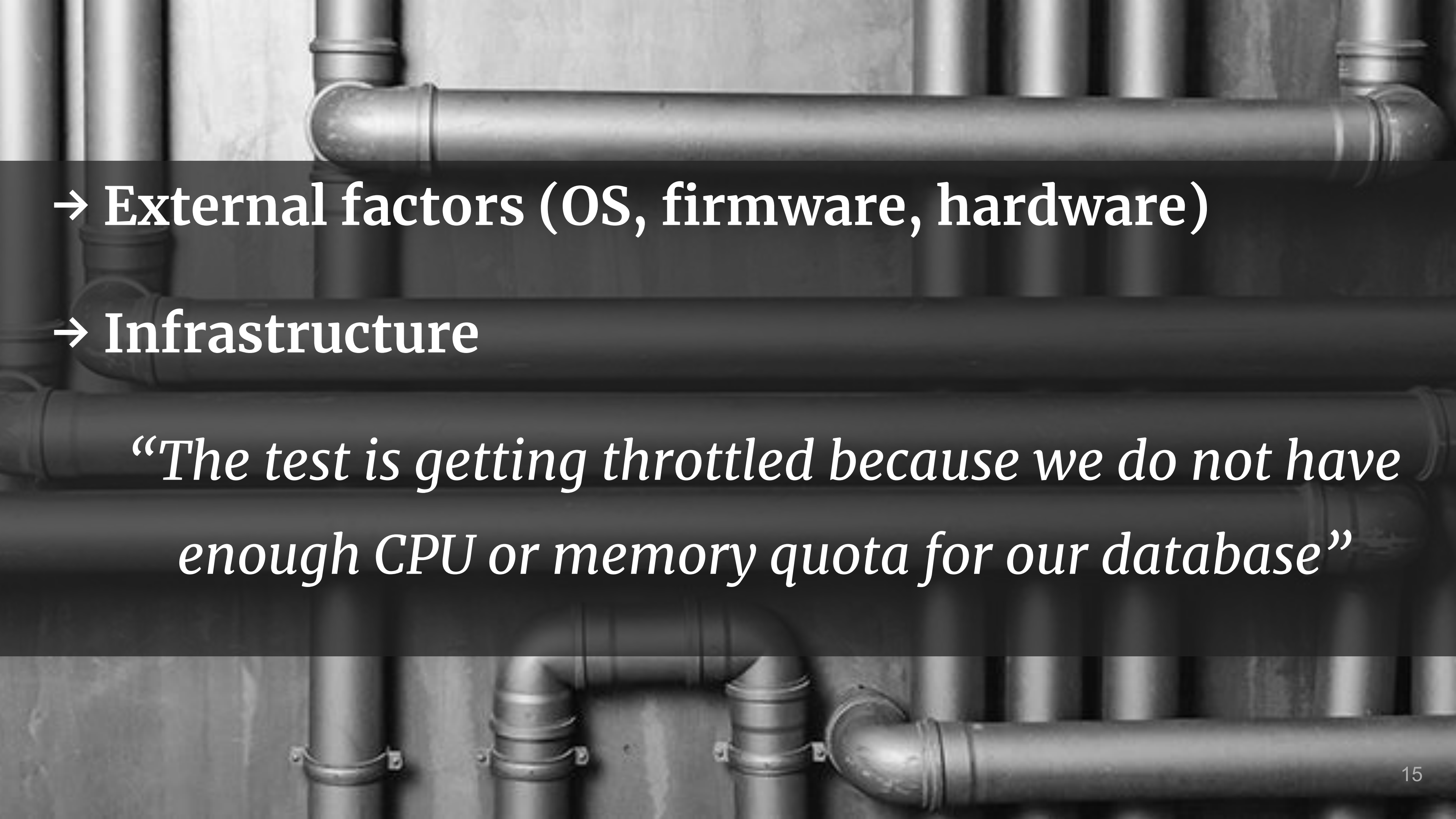
Usually leveraged in flakiness studies

→ Testing frameworks

→ Manual testers

➔ Orchestration between system components

“It only takes one timeout in the communication between two services or other middleware to make a test fail randomly”



- **External factors (OS, firmware, hardware)**
- **Infrastructure**

“The test is getting throttled because we do not have enough CPU or memory quota for our database”

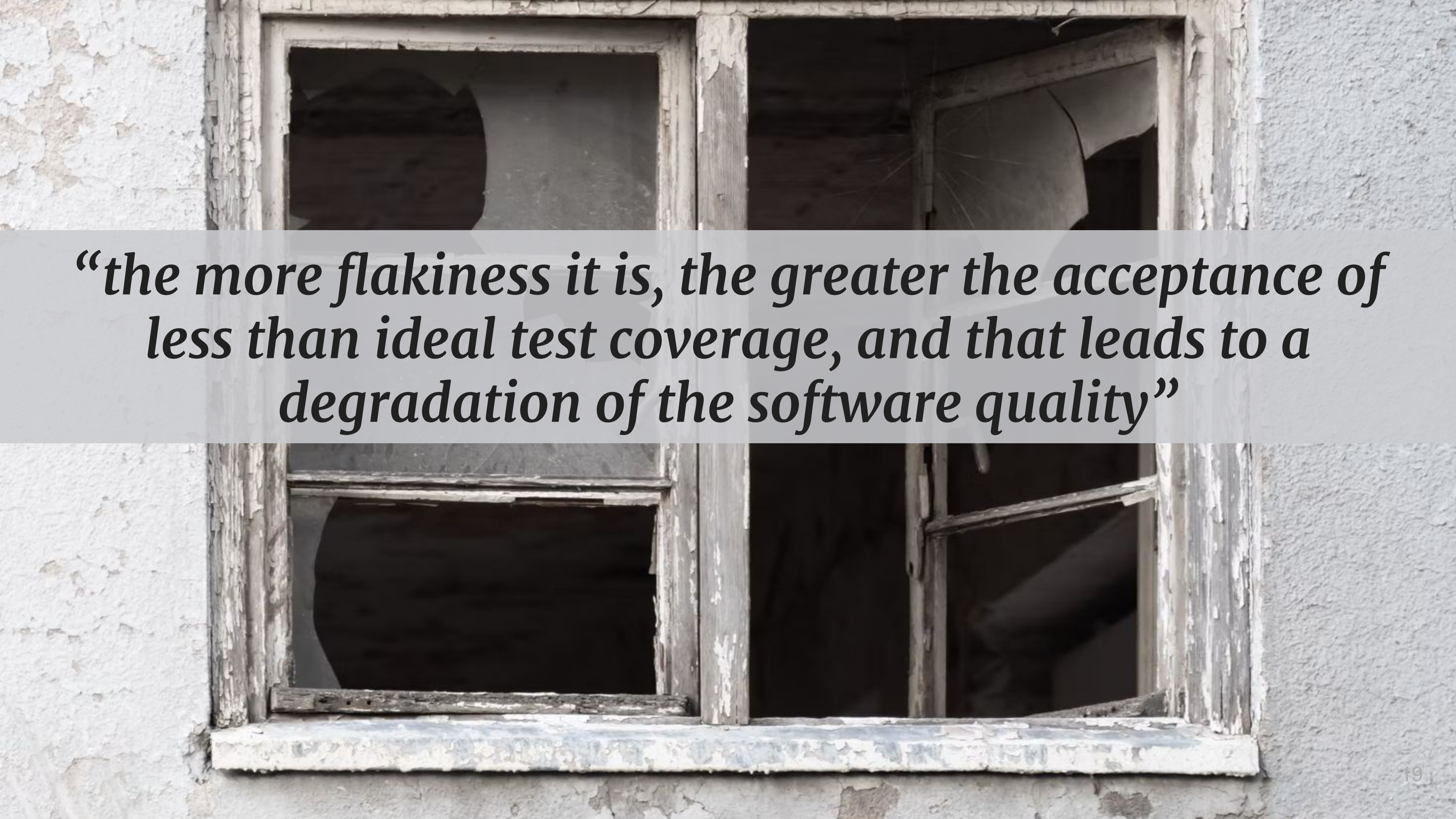
Impacts

→ Wasting development resources

→ Disrupting the CI and slowing down development

→ Undermining system reliability

→ Lenient testing policy
→ Writing less tests
→ Quality deterioration



“the more flakiness it is, the greater the acceptance of less than ideal test coverage, and that leads to a degradation of the software quality”

- ➔ Disguise non-deterministic features
- ➔ Deliver buggy product
- ➔ Debug in production

Mitigation

Strategies

1. Prevention
2. Detection
3. Treatment
4. Support

1. Prevention

→ Establish testing guidelines

- Avoid testing anti-patterns (*e.g.*, cupcake & ice cream cone)
- Enforce guidelines with static analysis

1. Prevention

→ Establish testing guidelines

→ Setup a reliable infrastructure

  → Limit external dependencies

  → Mock when possible

2. Detection

→ Reruns

→ Manual analysis (trace, screenshots)

→ Test history

3. Treatment

→ Fix

→ Ignore

→ Quarantine

→ Remove

→ Document

3. Treatment

→ Fix: rarely achieved

→ Ignore

→ Quarantine

→ Remove

→ Document

3. Treatment

→ Fix: rarely achieved

→ Ignore: “a very low flake rate is not worth inspecting”

→ Quarantine

→ Remove

→ Document

3. Treatment

→ Fix: rarely achieved

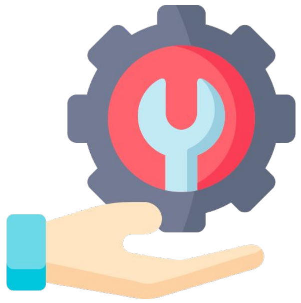
→ Ignore: “a very low flake rate is not worth inspecting”

→ Quarantine

→ Remove: “it’s better to remove the test due to its cost”

→ Document

4. Support



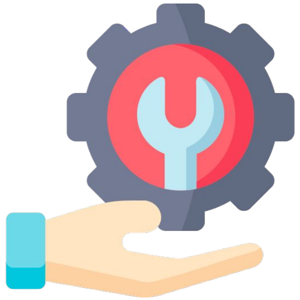
→ Monitor and log system and test outcomes

4. Support

→ Monitor and log system and test outcomes

- Facilitate reproduction and root cause analysis

4. Support



→ Monitor and log system and test outcomes

- Facilitate reproduction and root cause analysis

→ Establish testing workflows

- Test selection and prioritisation

Automation Opportunities

→ Reproduction and root cause identification (debug)

→ Reproduction and root cause identification

→ Prediction (using historical data, logs)

→ Reproduction and root cause identification

→ Prediction

→ Fine-grained analysis

$\Rightarrow$ flake rate, flakiness level estimation

→ Reproduction and root cause identification

→ Prediction

→ Fine-grained analysis

→ Test validation:

  ⇒ Static analysis (e.g. sleep)

→ Orchestration between system components

“It only takes one timeout in the communication between two services or other middleware to make a test fail randomly”

3. Treatment

→ Fix: rarely achieved

→ Ignore: “a very low flake rate is not worth inspecting”

→ Quarantine

→ Remove: “it’s better to remove the test due to its cost”

→ Document

→ Disguise non-deterministic features

→ Deliver buggy product

→ Debug in production

Automation Opportunities