# 🧪 Flaky Test Detection with Historical Test Analysis

## Overview

**Flaky tests** are tests that fail non-deterministically — they fail sometimes and pass at other times without any change in the underlying code. These are major productivity killers in CI/CD pipelines and can erode developer trust in test results. A robust flaky test detection strategy involves **historical test analysis**, statistical modeling, and automated quarantine systems.

## 🧠 Historical Analysis for Flake Detection

### 1. Test Result Logging

Track the result of each test over time:

- Test name
- Commit hash or build ID
- Pass/Fail status
- Execution time
- Platform/Environment details

Store in a structured format (e.g., Postgres, Elasticsearch, or BigQuery).

### 2. Flake Scoring

Define metrics such as:

- **Flake rate**: `Failures / Total runs`
- **Bounce rate**: Failures followed by pass in next retry
- **Intermittency score**: Normalized standard deviation of results

  🔁 Example:

`test_login_flow: 15 runs → [✓ ✓ ✗ ✓ ✓ ✗ ✓ ✓ ✓ ✓ ✓ ✓ ✗ ✓ ✓] Flake Rate = 3/15 = 20%`

## ? Tools for Flake Detection

| Tool | Features |
|---|---|
| **FlakyBot** (Google) | GitHub Action that detects and quarantines flaky tests |
| **BuildPulse** | SaaS that collects CI data, ranks flaky tests |
| **pytest-rerunfailures** | Useful for retry logic and detection support |
| **TestAnalytics** (CircleCI) | Test insights with flake analysis |
| **Custom ELK Stack** | Aggregates test logs and applies heuristics |

## 🧪 Automated Flaky Test Quarantine

1. Label flake candidates via thresholds (`flake_rate > 10%`)
2. Auto-quarantine in CI (e.g., skip unless manually invoked)
3. Notify developers with links to analysis
4. Periodically reintroduce and retest quarantined tests

   ✅ GitHub Actions Example:

```
if: steps.detect-flake.outputs.flaky == 'true'
run: echo "Quarantining test ${{ matrix.test }}"
```

## Conclusion

Historical test analysis is essential for proactively identifying and managing flaky tests. By collecting long-term data, computing flake scores, and integrating detection into CI, teams can prevent unreliable tests from blocking deploys and eroding confidence. The key is **detection, isolation, and continuous cleanup**.