The 24th Belgium-Netherlands Software Evolution Workshop. 17 - 18 November 2025, Enschede, The Netherlands.

University of Antwerp
Faculty of Science

# FlaDaGe

## A Framework for Generation of Synthetic Data to Compare Flakiness Scores

Mert Ege Can, Joanna Kisaakye, Mutlu Beyazıt, Serge Demeyer

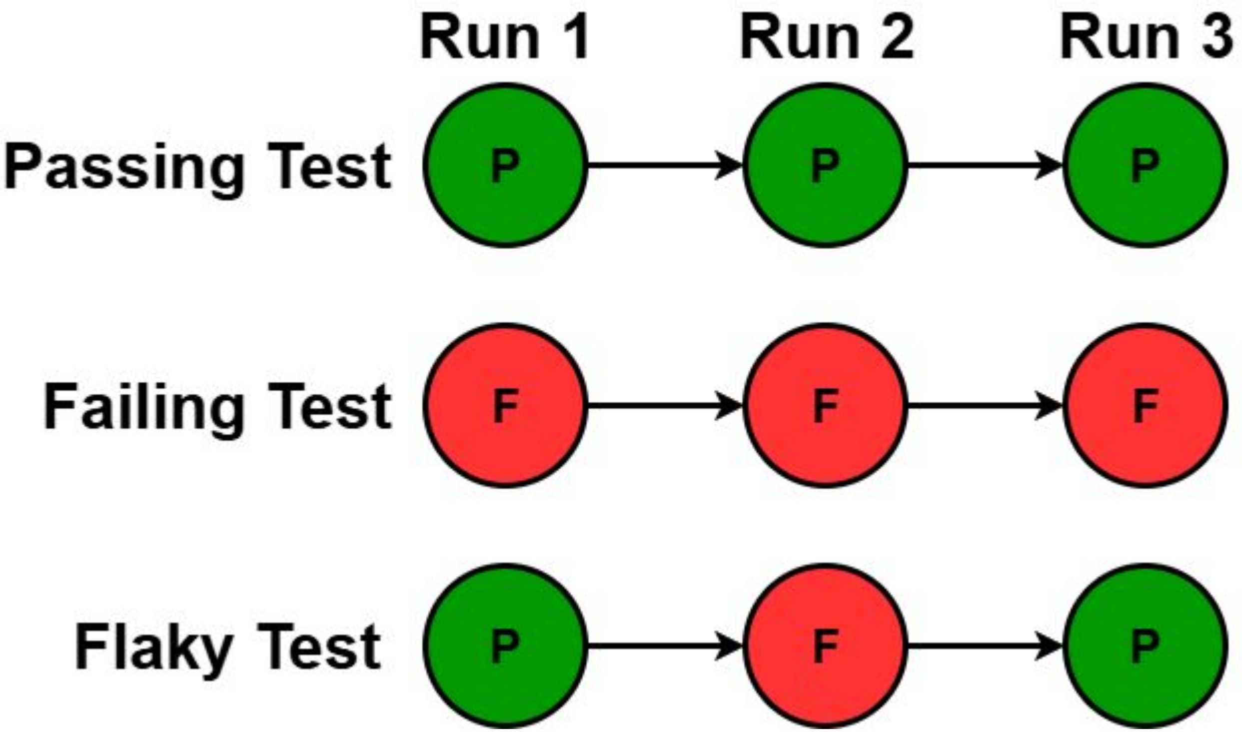
# Overview

1. Introduction
2. Research Question
3. FlaDaGe
4. Results
5. Next Steps

# Introduction

# What is a flaky test?

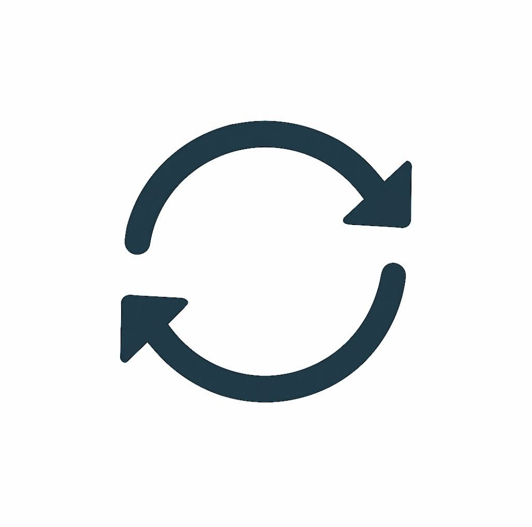
A test that alternates between different outcomes under unchanged conditions.



# Mitigation Strategies for Flaky Tests

## Re-run

Execute tests multiple times.



## Monitor

Record a long-term history of test results.

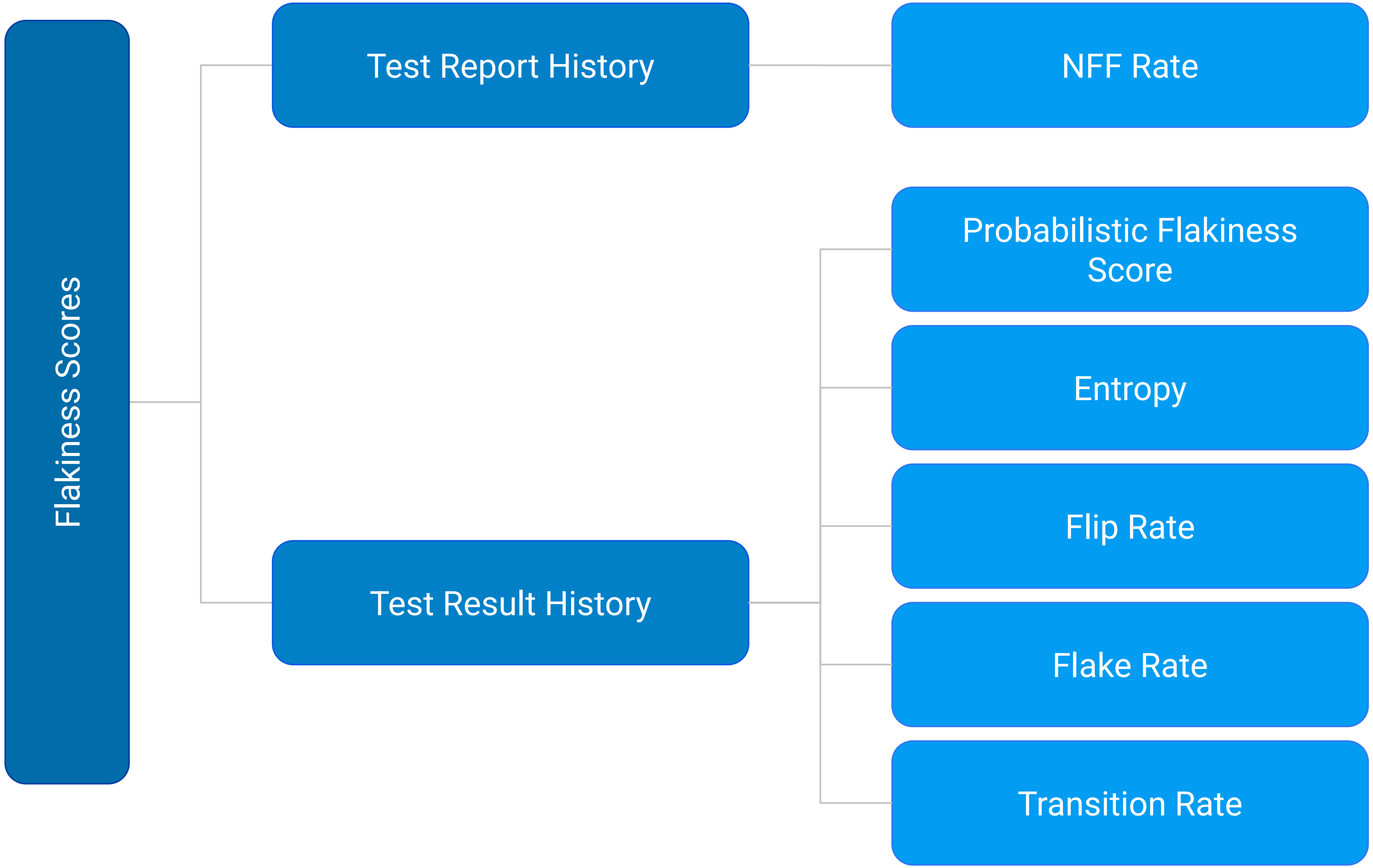
## Fix

Identify and resolve the root cause of flakiness.

The images are generated via Chat GPT.

# The Flakiness Score

The flakiness score of a test is derived from analysing its execution history over a defined period in the CI pipeline.



# Research Question

**How can we design a unified and statistically controlled dataset that enables a fair and algorithm-neutral comparison of different flakiness scoring algorithms?**

## How can we design a unified and statistically controlled dataset that enables a fair and algorithm-neutral comparison of different flakiness scoring algorithms?

$$\mathsf{NFFRate}_t(f, r) = \frac{f}{r}$$

The number of test failures that did not lead to a product fault, f, for a set of runs, r.

[3] Rehman, 2021

$$T(R_{v,*,\{P,F\}}) = \frac{numTransitions(R_{v,*,\{P,F\}})}{numTotalTransitions(R_{v,*})}$$

The number of times we observe transition, P $\longrightarrow$ F, divided by the number of possible transitions or flips.

[2] Kisaakye, 2024

## How can we design a unified and statistically controlled dataset that enables a fair and algorithm-neutral comparison of different flakiness scoring algorithms?

$$\mathsf{NFFRate}_t(f, r) = \frac{f}{r}$$

- The flakiness source is failed tests without reports.
- Likelihood is used to decide rerun order by Binomial Stability Order (BSO).

[3] Rehman et al, 2021

$$T(R_{v,*,\{P,F\}}) = \frac{numTransitions(R_{v,*,\{P,F\}})}{numTotalTransitions(R_{v,*})}$$

- The flakiness source is the transitions between different test outcomes.
- A flakiness score is calculated for each test per version.

[2] Kisaakye et al, 2024

## How can we design a unified and statistically controlled dataset that enables a fair and algorithm-neutral comparison of different flakiness scoring algorithms?

Comparing different scoring models requires a shared neutral evaluation ground

1. Different result states/outcomes
2. Report Associations
3. Varied flakiness trends
4. Version and run structure

# FlaDaGe

## Artificial Dataset Generation Framework

# Evolution

2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)

## Modeling and Ranking Flaky Tests at Apple

Emily Kowalczyk, Apple Inc., Cupertino, USA, ekowalczyk@apple.com
Karan Nair, Apple Inc., Cupertino, USA, karan_nair@apple
Zebao Gao, Apple Inc., Cupertino, USA, zebao_gao@apple.com
Leo Silberstein, Apple Inc., Cupertino, USA, lsilberstein@apple.com
Teng Long, Apple Inc., Cupertino, USA, teng_long@apple.com
Atif Memon, Apple Inc., Cupertino, USA, atif_memon@apple.com

### ABSTRACT

Test flakiness—inability to reliably repeat a test's Pass/Fail outcome—continues to be a significant problem in Industry, adversely impacting continuous integration and test pipelines. Completely eliminating flaky tests is not a realistic option as a significant fraction of system tests (typically non-hermetic) for services-based implementations exhibit some level of flakiness. In this paper, we view the flakiness of a test as a rankable value, which we quantify, track and assign a confidence. We develop two ways to model flakiness, capturing the randomness of test results via entropy, and the temporal variation via flipRate, and aggregating these over time. We have implemented our flakiness scoring service and discuss how its adoption has impacted test suites of two large services at Apple. We show how flakiness is distributed across the tests in these services, including typical score ranges and outliers. The flakiness scores are used to monitor and detect changes in flakiness trends. Evaluation results demonstrate near perfect accuracy in ranking, identification and alignment with human interpretation. The scores were used to identify 2 causes of flakiness in the dataset evaluated, which have been confirmed, and where fixes have been implemented or are underway. Our models reduced flakiness by 44% with less than 1% loss in fault detection.

**ACM Reference Format:**
Emily Kowalczyk, Karan Nair, Zebao Gao, Leo Silberstein, Teng Long, and Atif Memon. 2020. Modeling and Ranking Flaky Tests at Apple. In *Software Engineering in Practice (ICSE-SEIP '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3377813.3381370

### 1 INTRODUCTION

"Continuous integration (CI) and testing" is the cornerstone of quality assurance in today's large companies [4, 11, 13, 14]. Developers integrate code into a shared repository several times a day. Each check-in is verified by an automated build-and-test process, fully integrated into the CI server, allowing developers to detect problems early [2, 10, 15, 17]. CI processes are severely hindered due to the presence of *flaky tests* [14, 15]. A flaky test is one that may fail or pass non-deterministically. Intuitively, a flaky test's outcome defies the developer's expectation. For example, if a developer runs a test case multiple times, keeping all things constant (the test code, source code, the environment, etc.), then the developer expects the test should constantly pass or fail as no apparent changes have occurred.

Consider 4 tests in Table 1. Each test is run 15 times, with passes (green) and fails (red) shown in order of execution from left to right. After 4 runs, we see *epoch*[1] $e_1$, which is a change to the code under test; $e_2$ is a modification to the test code that impacts all 4 tests; and $e_3$ is a change to the underlying data used by the software under test. Assuming that everything else remains constant between epochs, our expectation is that test results will not change between epochs (they may certainly change *across* epochs). This is not the case for $tc_2$ and $tc_4$. Both tests passed and failed non-deterministically before $e_1$; test $tc_2$ between $e_2$ and $e_3$, and after $e_3$.

Even though one can claim to control "everything" when re-running tests, flakiness exists for a number of valid—and prevalent—reasons. Network traffic and latency, asynchronous waits and concurrency are some of the common causes of test flakiness [3, 8, 12]. Google reported 16% of their 4.2 million tests were flaky, causing 1.5% of their test runs to flake [15]. Similarly, Microsoft analyzed 5 projects and found 4.6% of their tests were flaky [10], and Mozilla maintains a database of flaky tests, which they have estimated grows by more than 100 new flaky tests each week [3].

We recognize that flaky tests may need to exist in a test repository because they sometimes help to uncover real bugs – indeed the underlying causes of flakiness may point to bugs in the test infrastructure or the code under test. We however, focus on the CI usecase, where flakiness causes tests to provide undesirable signals that cause unnecessary delays. By and large, developers agree that

**Table 1: Example test result history.**



[1] A significant event such as a code/data/config/flags change that may cause a test to change its behavior.



[1]Kowalczyk et al, 2020

- Modelling flakiness as static distributions with pre-assigned flakiness probabilities.

# Evolution

2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)

## Modeling and Ranking Flaky Tests at Apple

Emily Kowalczyk, Apple Inc., Cupertino, USA, ekowalczyk@apple.com
Karan Nair, Apple Inc., Cupertino, USA, karan_nair@apple
Zebao Gao, Apple Inc., Cupertino, USA, zebao_gao@apple.com
Leo Silberstein, Apple Inc., Cupertino, USA, lsilberstein@apple.com
Teng Long, Apple Inc., Cupertino, USA, teng_long@apple.com
Atif Memon, Apple Inc., Cupertino, USA, atif_memon@apple.com

### ABSTRACT

Test flakiness—inability to reliably repeat a test's Pass/Fail outcome—continues to be a significant problem in Industry, adversely impacting continuous integration and test pipelines. Completely eliminating flaky tests is not a realistic option as a significant fraction of system tests (typically non-hermetic) for services-based implementations exhibit some level of flakiness. In this paper, we view the flakiness of a test as a rankable value, which we quantify, track and assign a confidence. We develop two ways to model flakiness, capturing the randomness of test results via entropy, and the temporal variation via flipRate, and aggregating these over time. We have implemented our flakiness scoring service and discuss how its adoption has impacted test suites of two large services at Apple. We show how flakiness is distributed across the tests in these services, including typical score ranges and outliers. The flakiness scores are used to monitor and detect changes in flakiness trends. Evaluation results demonstrate near perfect accuracy in ranking, identification and alignment with human interpretation. The scores were used to identify 2 causes of flakiness in the dataset evaluated, which have been confirmed, and where fixes have been implemented or are underway. Our models reduced flakiness by 44% with less than 1% loss in fault detection.

**ACM Reference Format:**
Emily Kowalczyk, Karan Nair, Zebao Gao, Leo Silberstein, Teng Long, and Atif Memon. 2020. Modeling and Ranking Flaky Tests at Apple. In *Software Engineering in Practice (ICSE-SEIP '20), May 23–29, 2020, Seoul, Republic of Korea*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3377813.3381370

### 1 INTRODUCTION

"Continuous integration (CI) and testing" is the cornerstone of quality assurance in today's large companies [4, 11, 13, 14]. Developers integrate code into a shared repository several times a day. Each check-in is verified by an automated build-and-test process, fully integrated into the CI server, allowing developers to detect problems early [2, 10, 15, 17]. CI processes are severely hindered due to the presence of *flaky tests* [14, 15]. A flaky test is one that may fail or pass non-deterministically. Intuitively, a flaky test's outcome defies the developer's expectation. For example, if a developer runs a test case multiple times, keeping all things constant (the test code, source code, the environment, etc.), then the developer expects the test should constantly pass or fail as no apparent changes have occurred.

Consider 4 tests in Table 1. Each test is run 15 times, with passes (green) and fails (red) shown in order of execution from left to right. After 4 runs, we see *epoch*[1] $e_1$, which is a change to the code under test; $e_2$ is a modification to the test code that impacts all 4 tests; and $e_3$ is a change to the underlying data used by the software under test. Assuming that everything else remains constant between epochs, our expectation is that test results will not change between epochs (they may certainly change *across* epochs). This is not the case for $tc_2$ and $tc_4$. Both tests passed and failed non-deterministically before $e_1$; test $tc_2$ between $e_2$ and $e_3$, and after $e_3$.

Even though one can claim to control "everything" when re-running tests, flakiness exists for a number of valid—and prevalent—reasons. Network traffic and latency, asynchronous waits and concurrency are some of the common causes of test flakiness [3, 8, 12]. Google reported 16% of their 4.2 million tests were flaky, causing 1.5% of their test runs to flake [15]. Similarly, Microsoft analyzed 5 projects and found 4.6% of their tests were flaky [10], and Mozilla maintains a database of flaky tests, which they have estimated grows by more than 100 new flaky tests each week [3].

We recognize that flaky tests may need to exist in a test repository because they sometimes help to uncover real bugs – indeed the underlying causes of flakiness may point to bugs in the test infrastructure or the code under test. We however, focus on the CI usecase, where flakiness causes tests to provide undesirable signals that cause unnecessary delays. By and large, developers agree that

**Table 1: Example test result history.**



[1] A significant event such as a code/data/config/flags change that may cause a test to change its behavior.


ACM ISBN 978-1-4503-7123-0/20/05…$15.00
https://doi.org/10.1145/3377813.3381370

## Extending a Flakiness Score for System-Level Tests

Joanna Kisaakye✉[1,2][0000–0001–7081–5385], Mutlu Beyazıt[1,2][0000–0003–2714–8155], and Serge Demeyer[1,2][0000–0002–4463–2945]

[1] Universiteit Antwerpen, Antwerp, Belgium
joanna.kisaakye@uantwerpen.be
mutlu.beyazit@uantwerpen.be
serge.demeyer@uantwerpen.be
[2] Flanders Make vzw, Kortrijk, Belgium

**Abstract.** Flaky tests (i.e. automated tests with a non-deterministic test outcome) undermine the trustworthiness of today's DevOps build-pipelines, and recent research has investigated ways to detect or even remove flaky tests. In contrast, others proclaim that test engineers should "Assume all Tests Are Flaky" because, in today's build-pipelines, one can never fully control all components of the system under test. Test engineers then capture the randomness of test results via what is called a *flakiness score*. In this paper, we extend an existing flakiness score to deal with system-level tests. We illustrate, via simulated test outcomes, how this refined score can support three different strategies for dealing with flaky tests — (i) Rerun, (ii) Fix and (iii) Monitor.

**Keywords:** DevOps · Flaky Tests · Flakiness Score

### 1 Introduction

DevOps is defined by Bass et al. as "*a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality*" [2]. The combination of these practices is embedded in a fully automated build-pipeline. Such a build-pipeline is driven by a series of automated tests that scrutinise every code change.

Flaky tests (i.e. automated tests with a nondeterministic test outcome) undermine the trustworthiness of such a build-pipeline. Studies have shown that flakiness, when neglected, can lead to developer stress, and waste of time and resources, ultimately compromising product quality [12,22,26]. Consequently, various existing studies offer different solutions —both automated and manual— to detect or even remove flaky tests [4,7,8,19,30,32].

In contrast, others proclaim that test engineers should "Assume all Tests Are Flaky" [3,13]. Indeed, many data centric systems have evolved from monolithic architectures to micro-service architectures [5]. Build-pipelines therefore rely on a distributed test execution environment where some aspects of the system configuration are inherently out of the test engineers control. For embedded systems, the build-pipeline distinguishes between model-in-the-loop, software-in-the-loop and hardware-in-the-loop [28]. There as well, the various system configurations induce a certain degree of uncertainty with respect to the real-time behaviour.

Adopting an "Assume all Tests Are Flaky" perspective, test engineers consider a test as having a probabilistic outcome (the range $[0 \ldots 1]$ in favour of a particular test outcome) instead of deterministic one (only one of $\{pass, fail\}$) and capture the randomness of test outcomes via what is

[1]Kowalczyk et al, 2020

[2] Kisaakye et al, 2024

- Added dynamic flakiness distributions.

# Evolution

2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)

## Modeling and Ranking Flaky Tests at Apple

Emily Kowalczyk, Apple Inc., Cupertino, USA, ekowalczyk@apple.com
Karan Nair, Apple Inc., Cupertino, USA, karan_nair@apple
Zebao Gao, Apple Inc., Cupertino, USA, zebao_gao@apple.com
Leo Silberstein, Apple Inc., Cupertino, USA, lsilberstein@apple.com
Teng Long, Apple Inc., Cupertino, USA, teng_long@apple.com
Atif Memon, Apple Inc., Cupertino, USA, atif_memon@apple.com

[1]Kowalczyk et al, 2020

## Extending a Flakiness Score for System-Level Tests

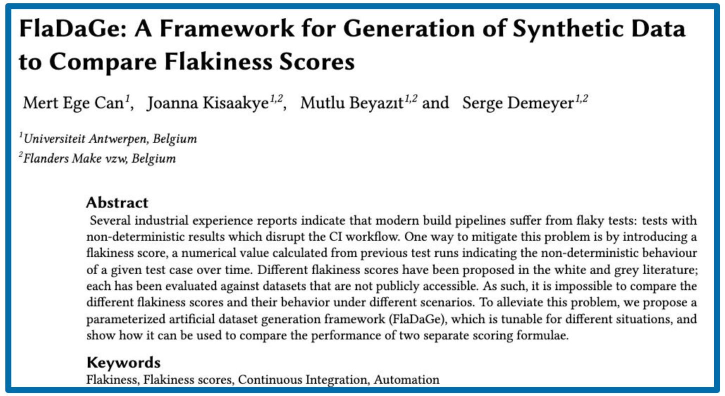
Joanna Kisaakye[1,2][0000−0001−7081−5385], Mutlu Beyazıt[1,2][0000−0003−2714−8155], and Serge Demeyer[1,2][0000−0002−4463−2945]

[1] Universiteit Antwerpen, Antwerp, Belgium
joanna.kisaakye@uantwerpen.be
mutlu.beyazit@uantwerpen.be
serge.demeyer@uantwerpen.be
[2] Flanders Make vzw, Kortrijk, Belgium

**Abstract.** Flaky tests (i.e. automated tests with a non-deterministic test outcome) undermine the trustworthiness of today's DevOps build-pipelines, and recent research has investigated ways to detect or even remove flaky tests. In contrast, others proclaim that test engineers should "Assume all Tests Are Flaky" because, in today's build-pipelines, one can never fully control all components of the system under test. Test engineers then capture the randomness of test results via what is called a *flakiness score*. In this paper, we extend an existing flakiness score to deal with system-level tests. We illustrate, via simulated test outcomes, how this refined score can support three different strategies for dealing with flaky tests — (i) Rerun, (ii) Fix and (iii) Monitor.

**Keywords:** DevOps · Flaky Tests · Flakiness Score

[2] Kisaakye et al, 2024



## FlaDaGe: A Framework for Generation of Synthetic Data to Compare Flakiness Scores

Mert Ege Can[1], Joanna Kisaakye[1,2], Mutlu Beyazıt[1,2] and Serge Demeyer[1,2]

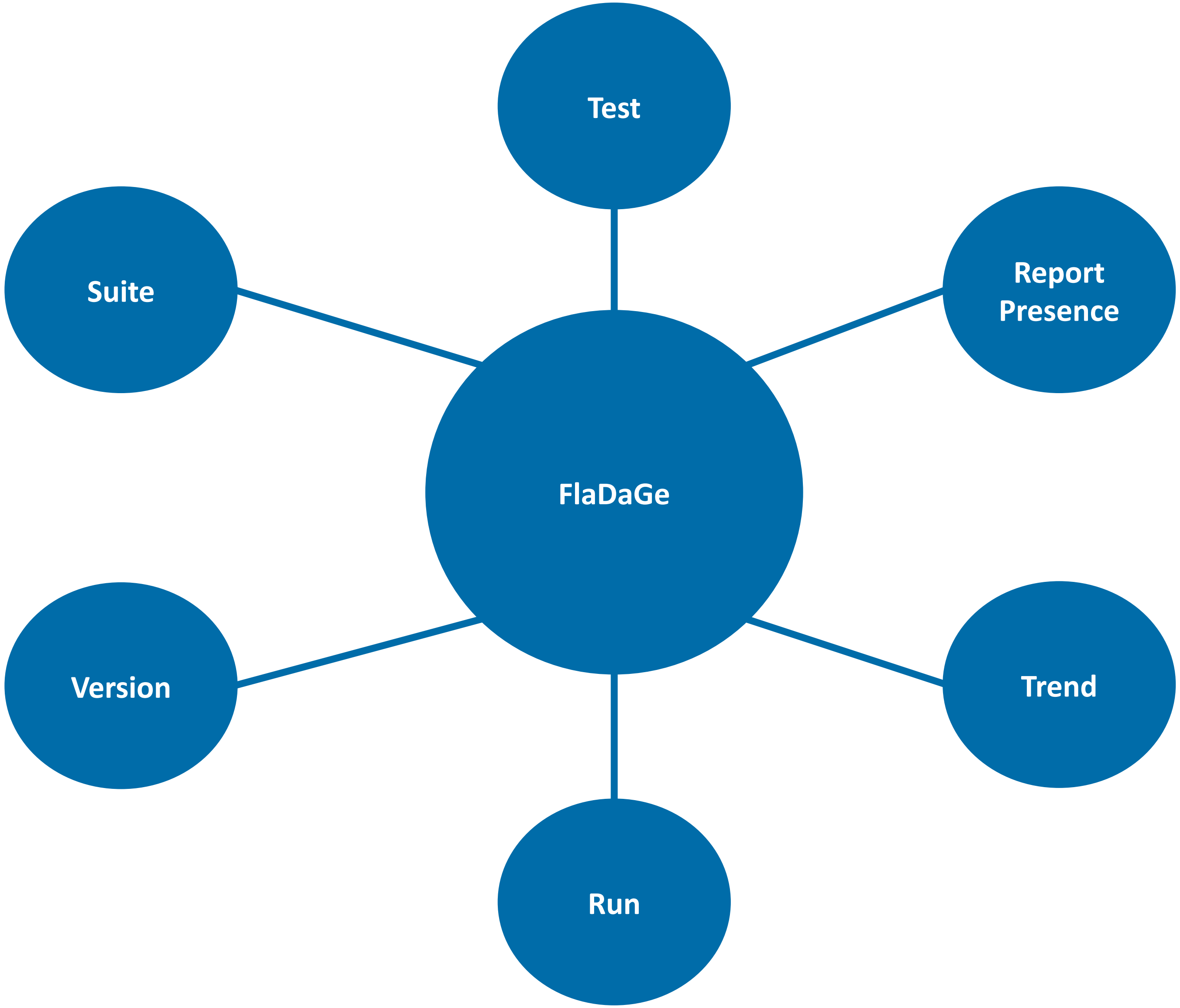
[1]*Universiteit Antwerpen, Belgium*
[2]*Flanders Make vzw, Belgium*

**Abstract**
Several industrial experience reports indicate that modern build pipelines suffer from flaky tests: tests with non-deterministic results which disrupt the CI workflow. One way to mitigate this problem is by introducing a flakiness score, a numerical value calculated from previous test runs indicating the non-deterministic behaviour of a given test case over time. Different flakiness scores have been proposed in the white and grey literature; each has been evaluated against datasets that are not publicly accessible. As such, it is impossible to compare the different flakiness scores and their behavior under different scenarios. To alleviate this problem, we propose a parameterized artificial dataset generation framework (FlaDaGe), which is tunable for different situations, and show how it can be used to compare the performance of two separate scoring formulae.

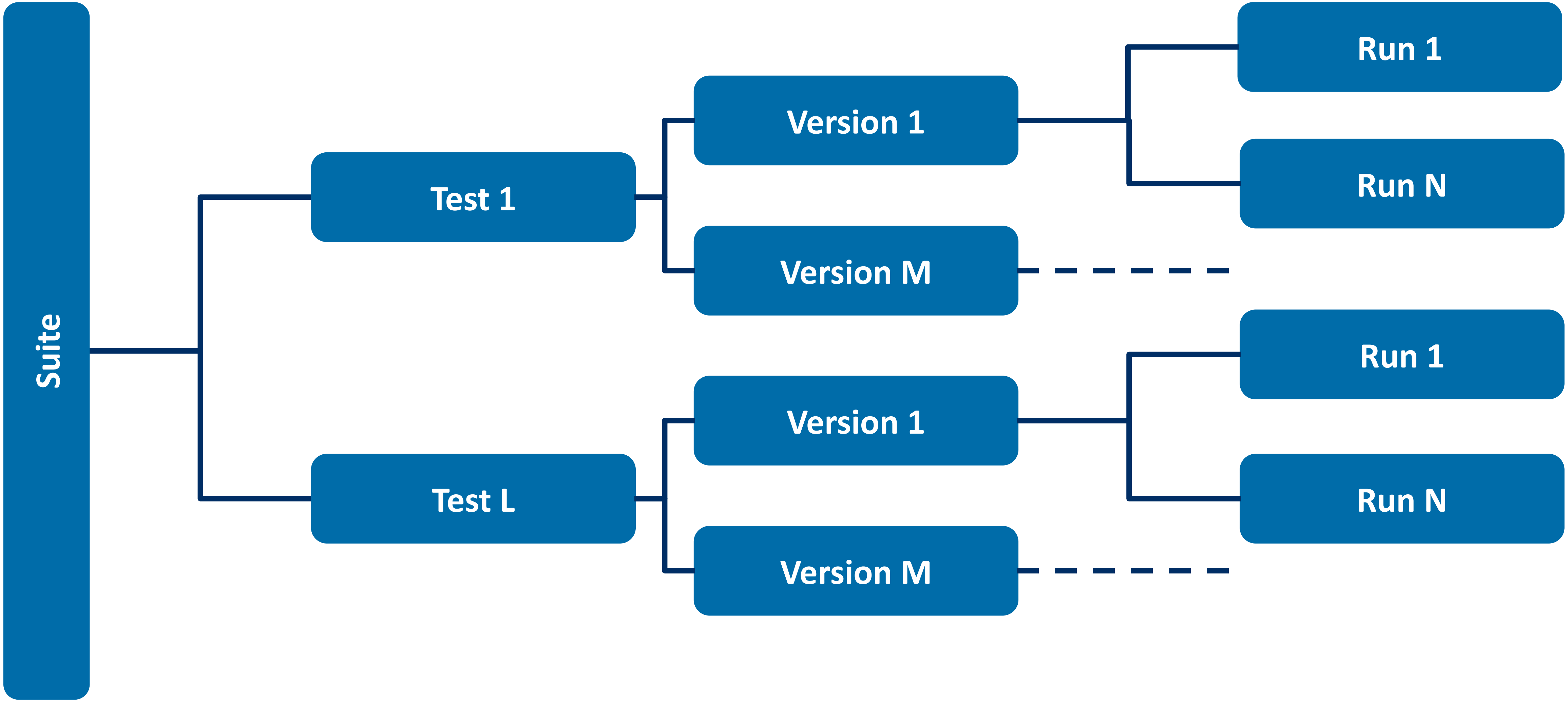
**Keywords**
Flakiness, Flakiness scores, Continuous Integration, Automation

- Dynamic flakiness distributions at multiple levels
- Support for different scoring models

# Concepts

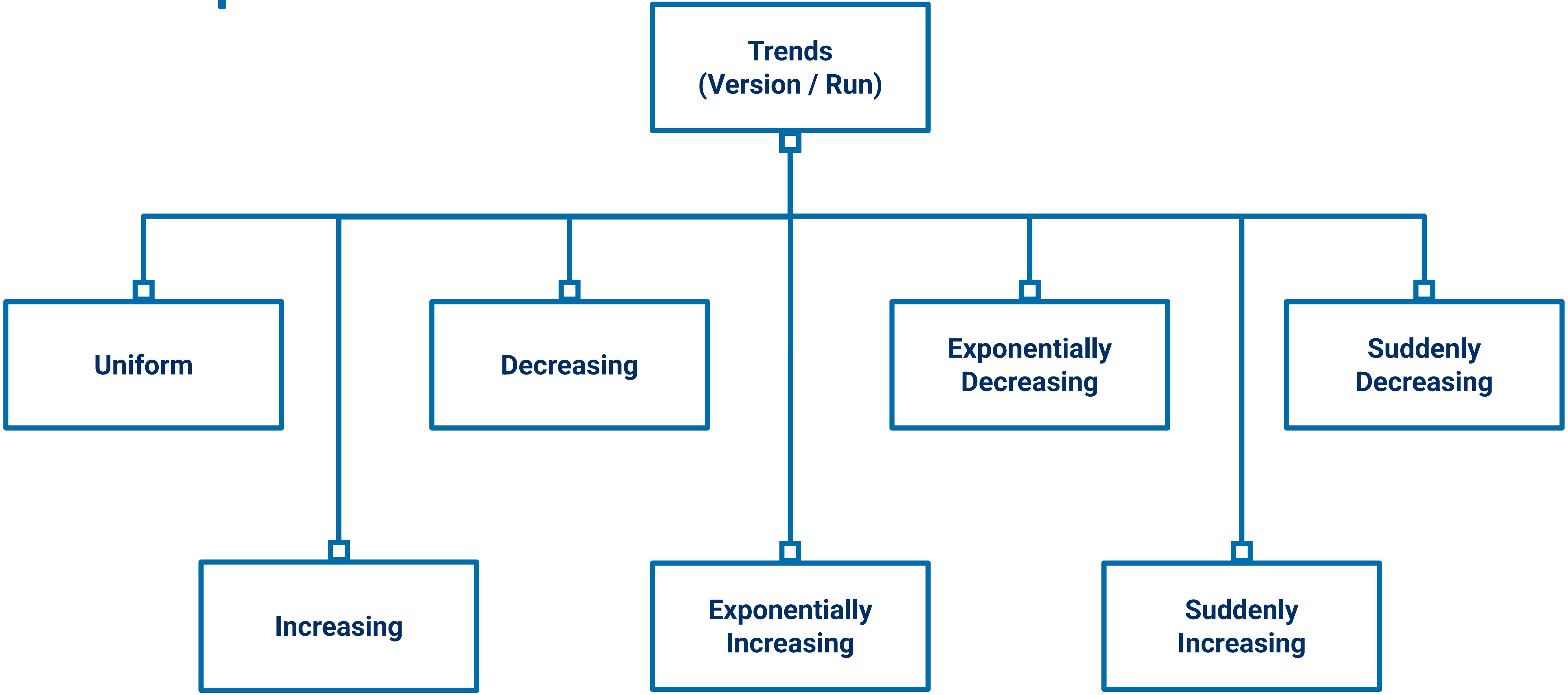


# Concepts



Suite
Test 1
Test L
Version 1
Version M
Version 1
Version M
Run 1
Run N
Run 1
Run N

# Concepts


Trends
(Version / Run)
Uniform
Increasing
Decreasing
Exponentially Increasing
Exponentially Decreasing
Suddenly Increasing
Suddenly Decreasing

# Run Attributes

1. Test Id
2. Release Id
3. Run Id
4. Report Flag
5. Verdict
6. Execution Timestamp

# Results

# Dataset Overview

100 Tests

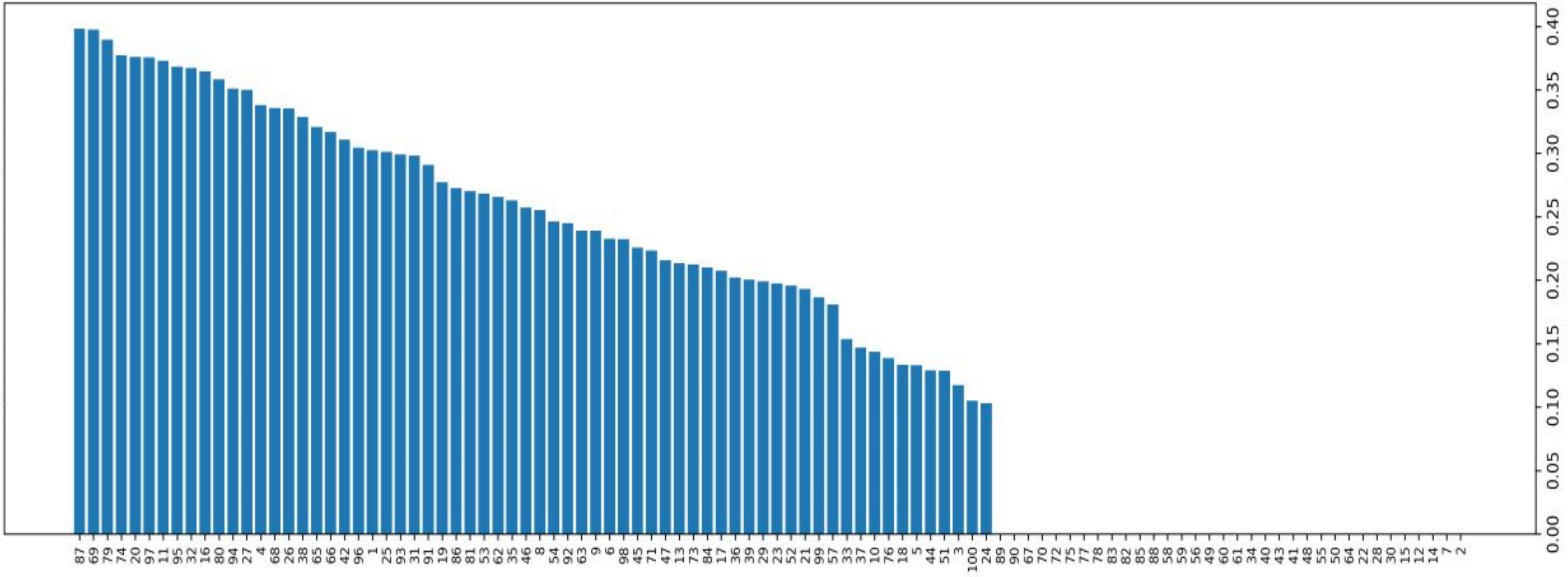
250 Runs

4 Versions

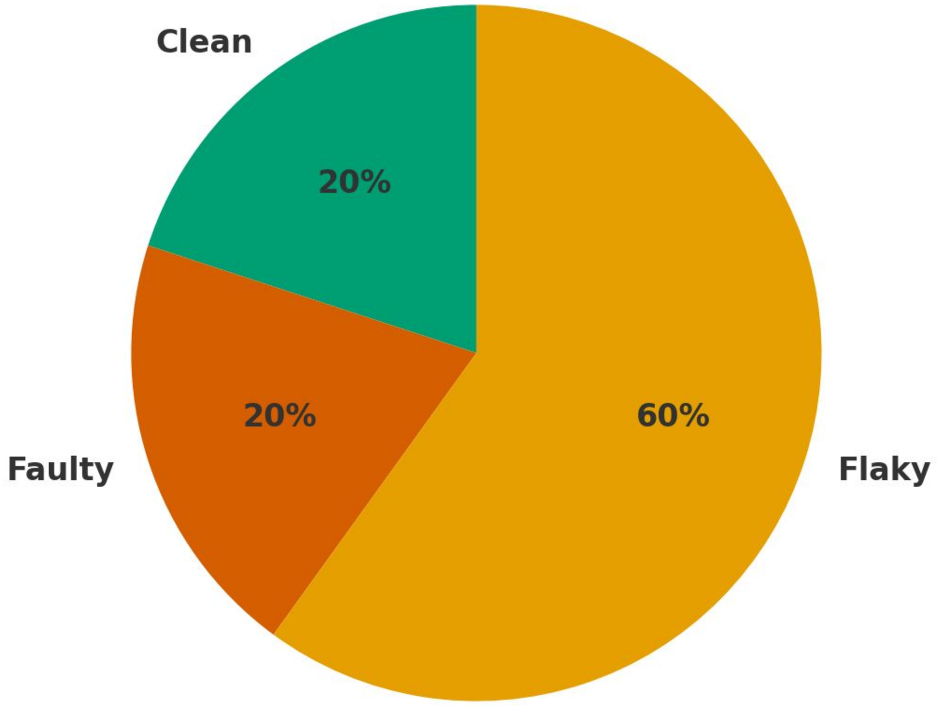
5000000 entries

49 Trend combinations

# Test Level Flakiness Probability Distribution



# Flaky Test Distribution

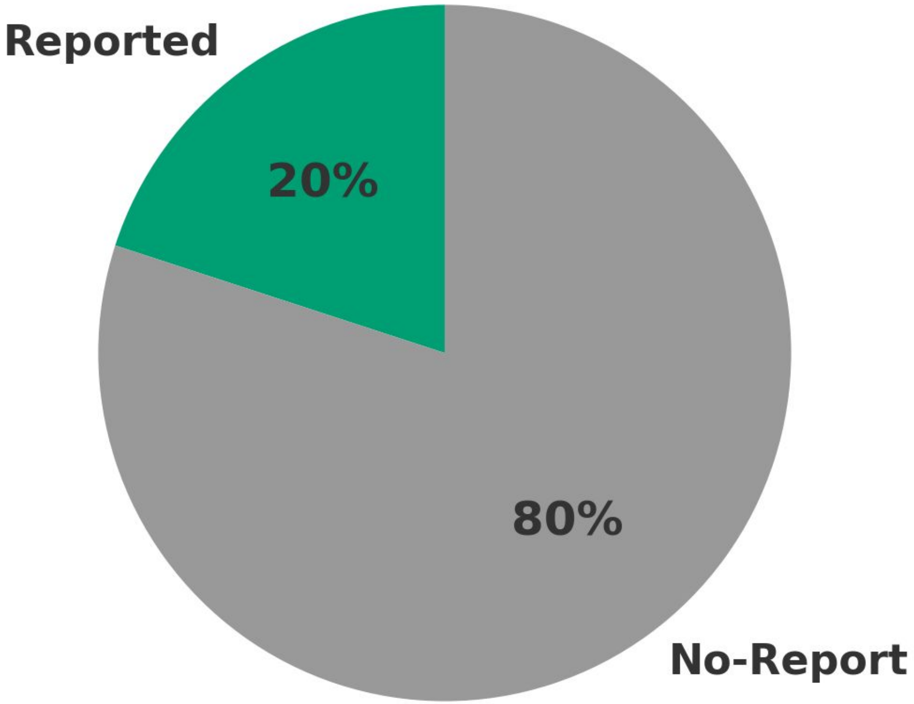



## Flakiness Setting

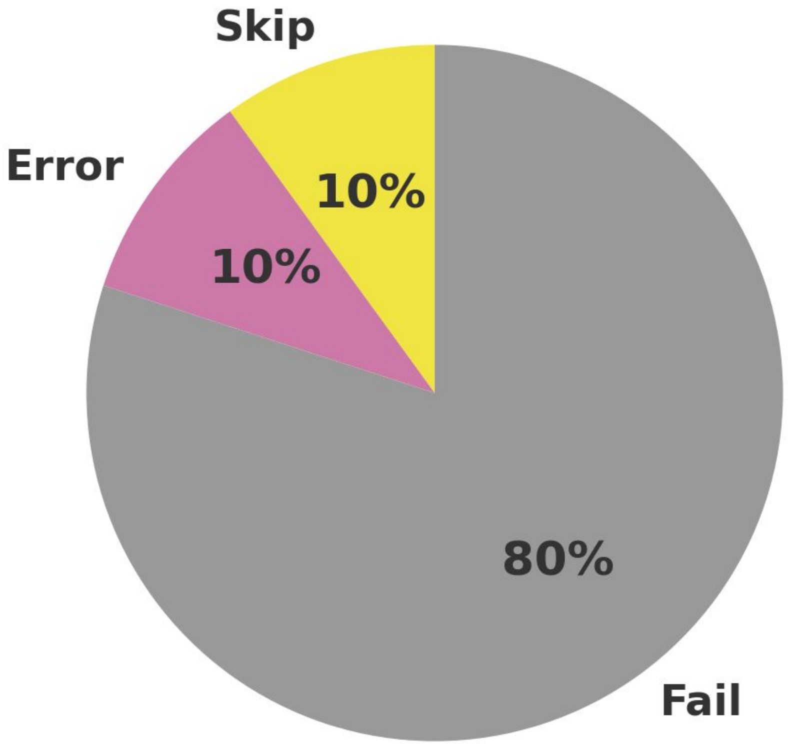
- Clean tests never fail.
- Faulty tests always fail.
- Flaky tests randomly assign their result in each run.

# Flaky Outcome Distributions

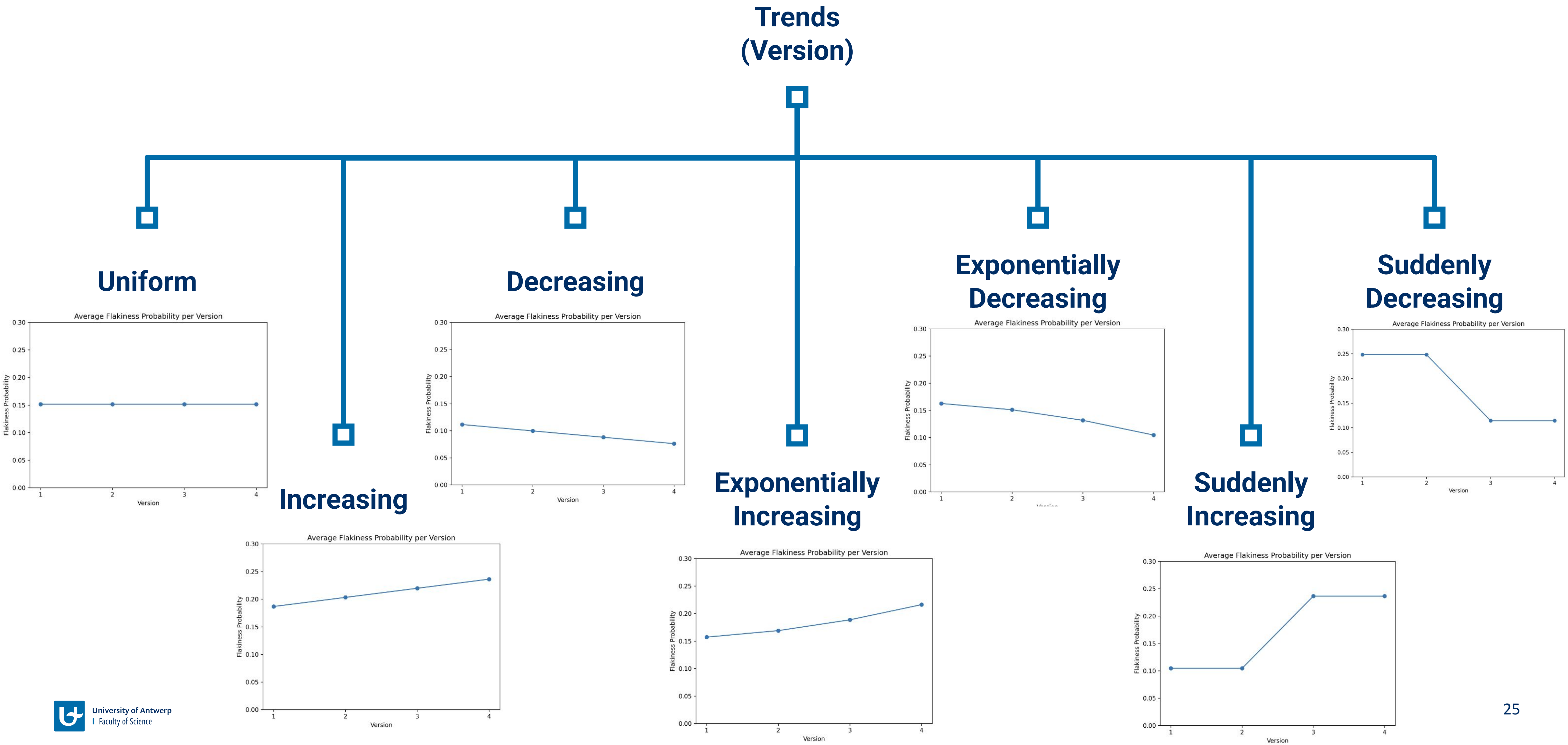
## NFF Flaky Outcomes Distribution




## EFS Flaky Outcomes Distribution

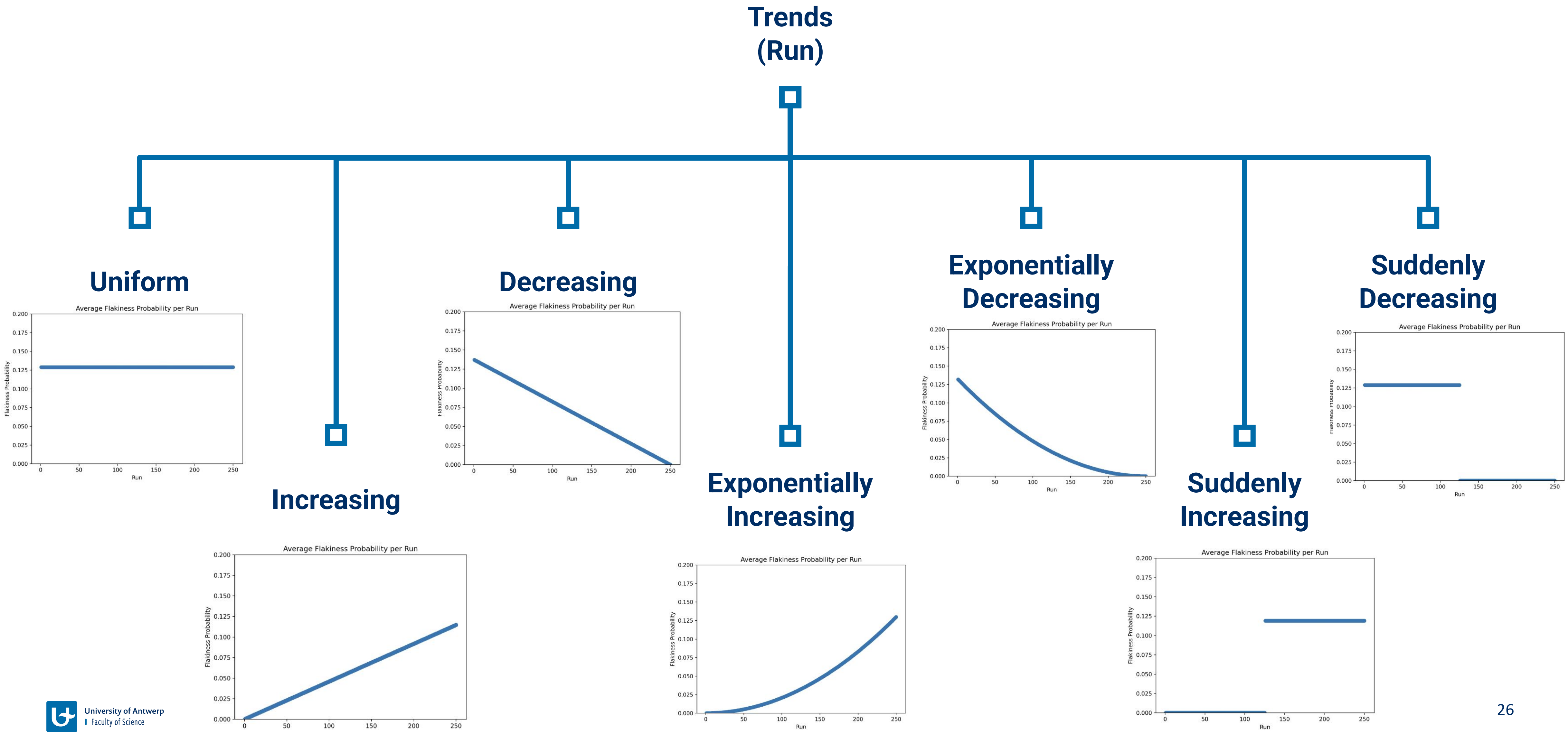


# Trends



Trends
(Version)
Uniform
Increasing
Decreasing
Exponentially
Increasing
Exponentially
Decreasing
Suddenly
Increasing
Suddenly
Decreasing
Average Flakiness Probability per Version
Flakiness Probability
Version

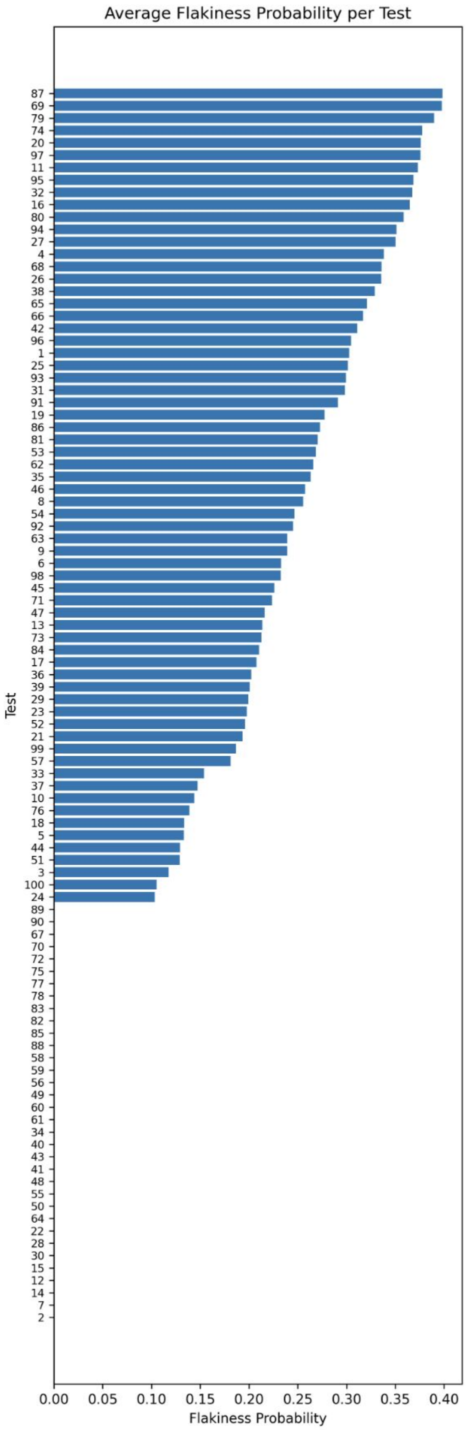
# Trends

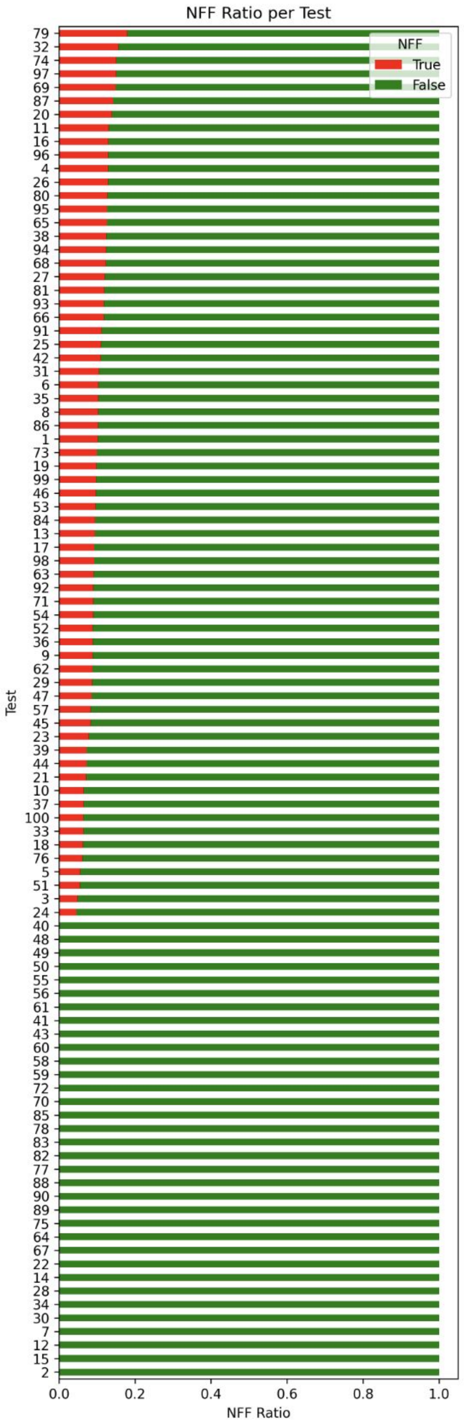


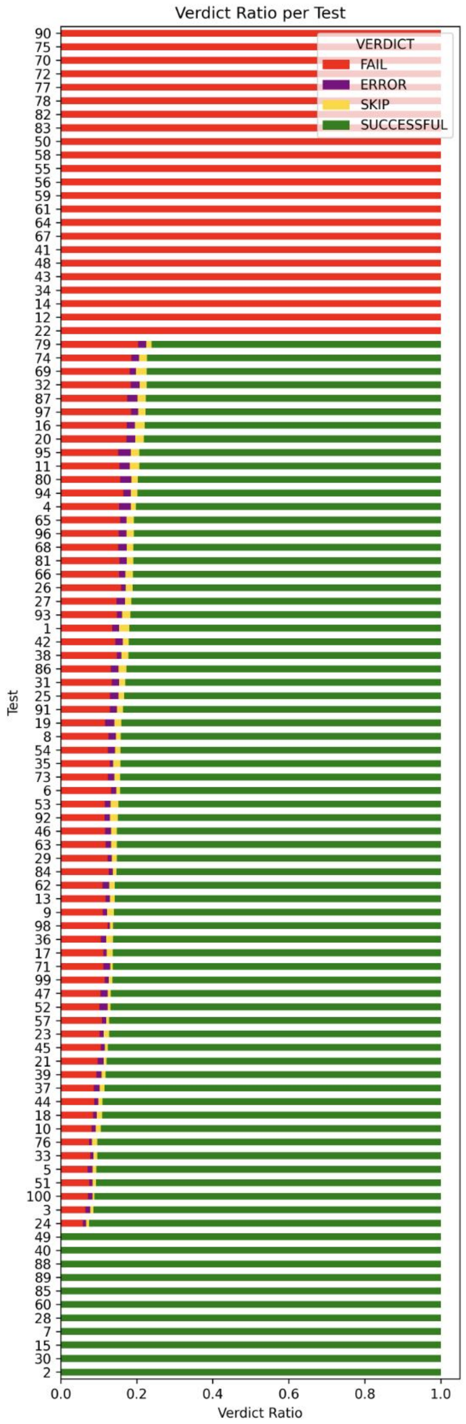
Trends (Run)
Uniform
Increasing
Decreasing
Exponentially Increasing
Exponentially Decreasing
Suddenly Increasing
Suddenly Decreasing
Average Flakiness Probability per Run
Flakiness Probability
Run

# Flakiness

As observed by NFF and EFS for the **increase -decrease** trend combination.

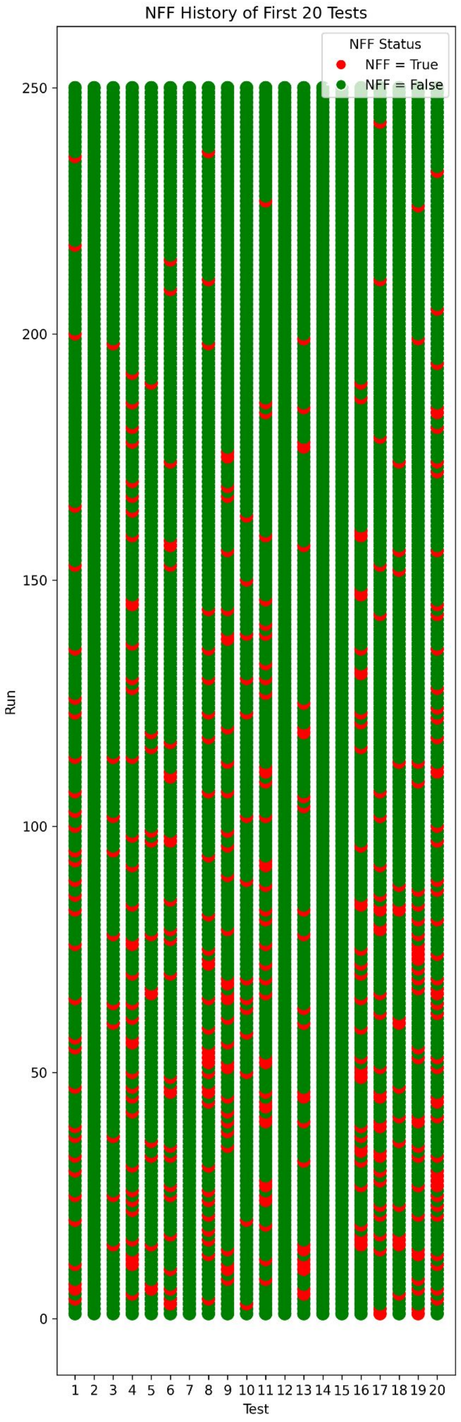


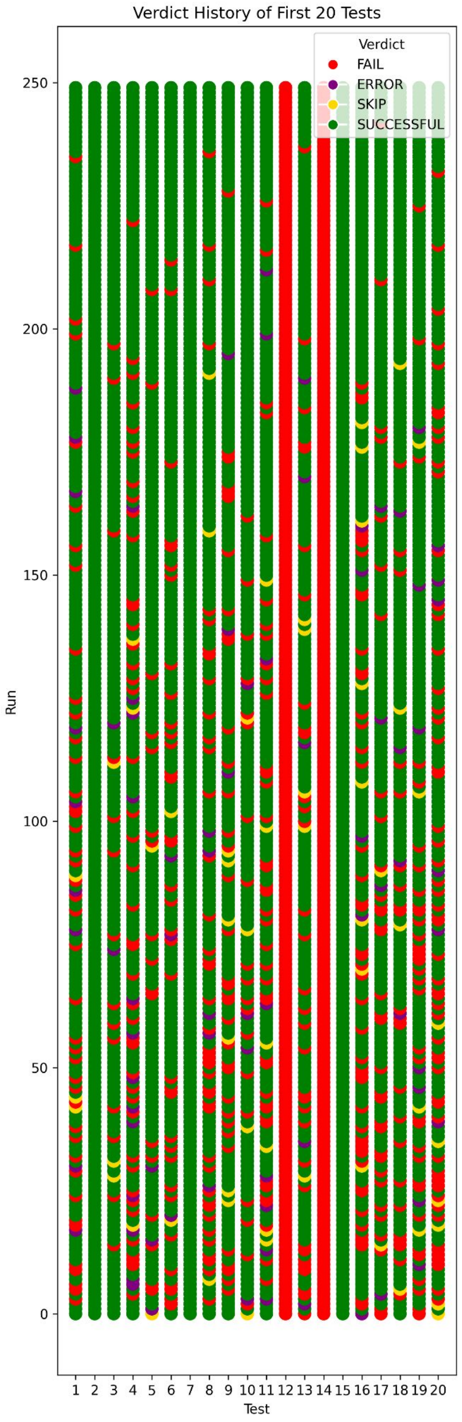



# History Correlation

How each algorithm views the same data for the first 20 tests.



## How can we design a unified and statistically controlled dataset that enables a fair and algorithm-neutral comparison of different flakiness scoring algorithms?

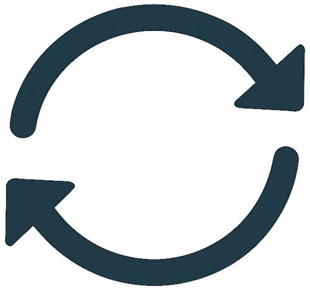
1. **Algorithm-neutral datasets** are generated by random distribution.

2. **49** unique suites were created for every version/run trend combination.

3. Randomly assigned **report presence** flag and **Pass, Fail, Skip and Error** states.

# Next Steps

# Which algorithm is best suited for each task?



- Run efficiency
- Correlation to Ground Truth ordering

*Kendall’s Tau*

- Proximity to Underlying Trend

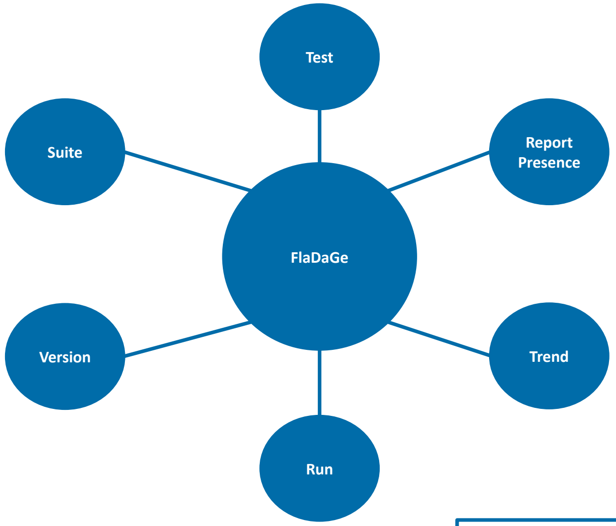
*Frechet Distance*

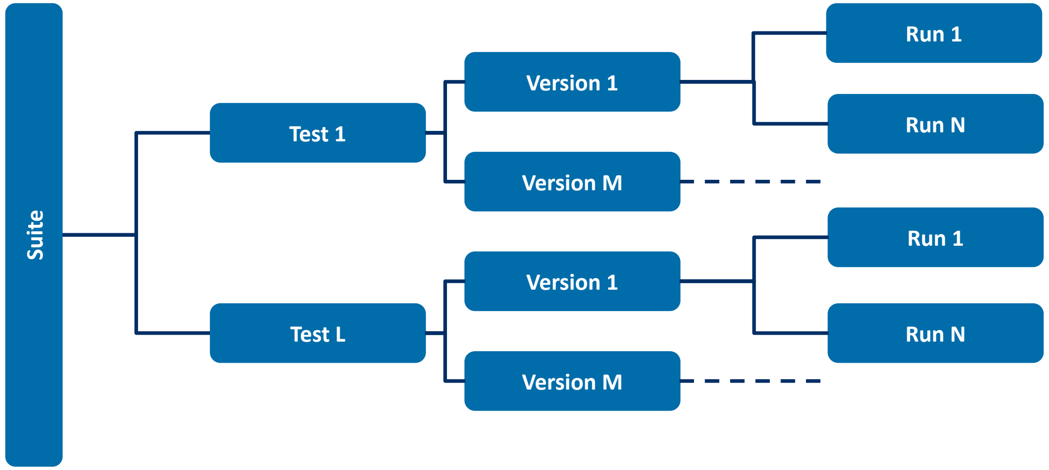
- Batch prioritisation

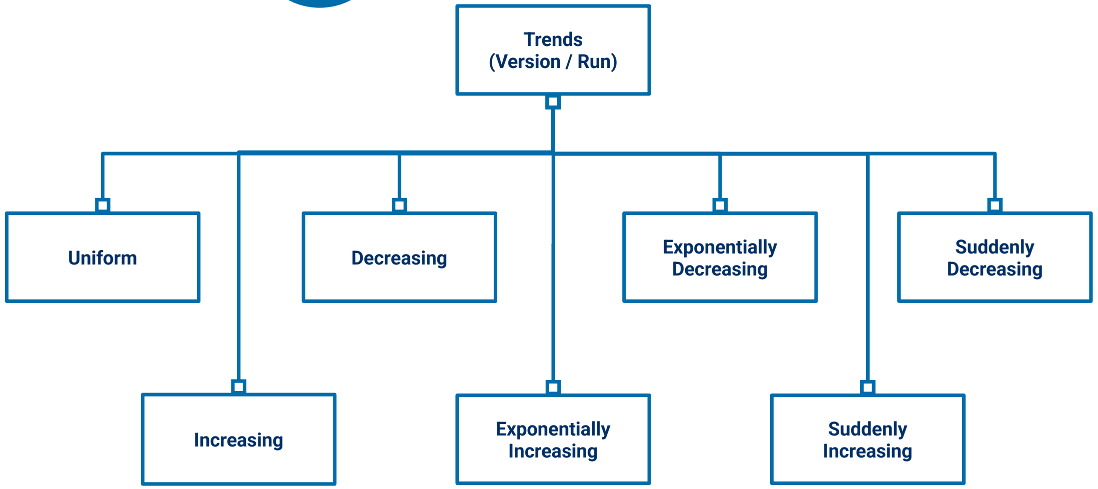
*Top-k Overlap*

The images are generated via Chat GPT.

## FlaDaGe | Parameterised Dataset | Reproducible Comparison




# References


- Emily Kowalczyk, Karan Nair, Zebao Gao, Leo Silberstein, Teng Long, and Atif Memon. “Modeling and ranking flaky tests at Apple.” In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '20), 2020.

- Joanna Kisaakye, Mutlu Beyazıt, and Serge Demeyer. “Extending a flakiness score for system-level tests.” IFIP International Conference on Testing Software and Systems. Cham: Springer Nature Switzerland, 2024.

- Maaz Hafeez Ur Rehman,  and Peter C. Rigby. “Quantifying No-Fault-Found test failures to prioritize inspection of flaky tests at Ericsson.” Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2021.