Keeping Master Green with Machine Learning

João Luís Lousada

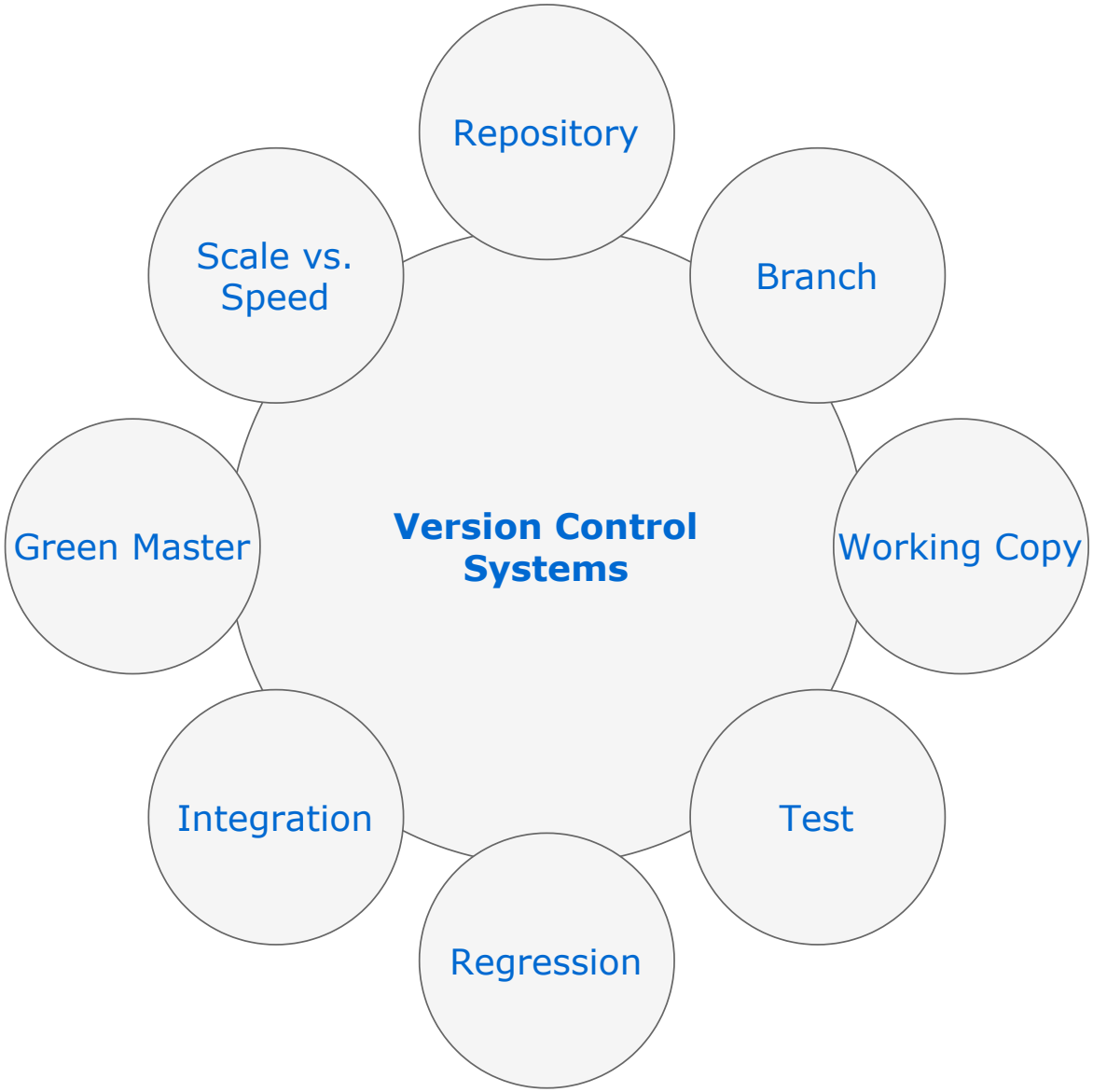
Supervisors: Prof. Doutor Rui Dilão
Eng. Miguel Ribeiro

Projeto MEFT

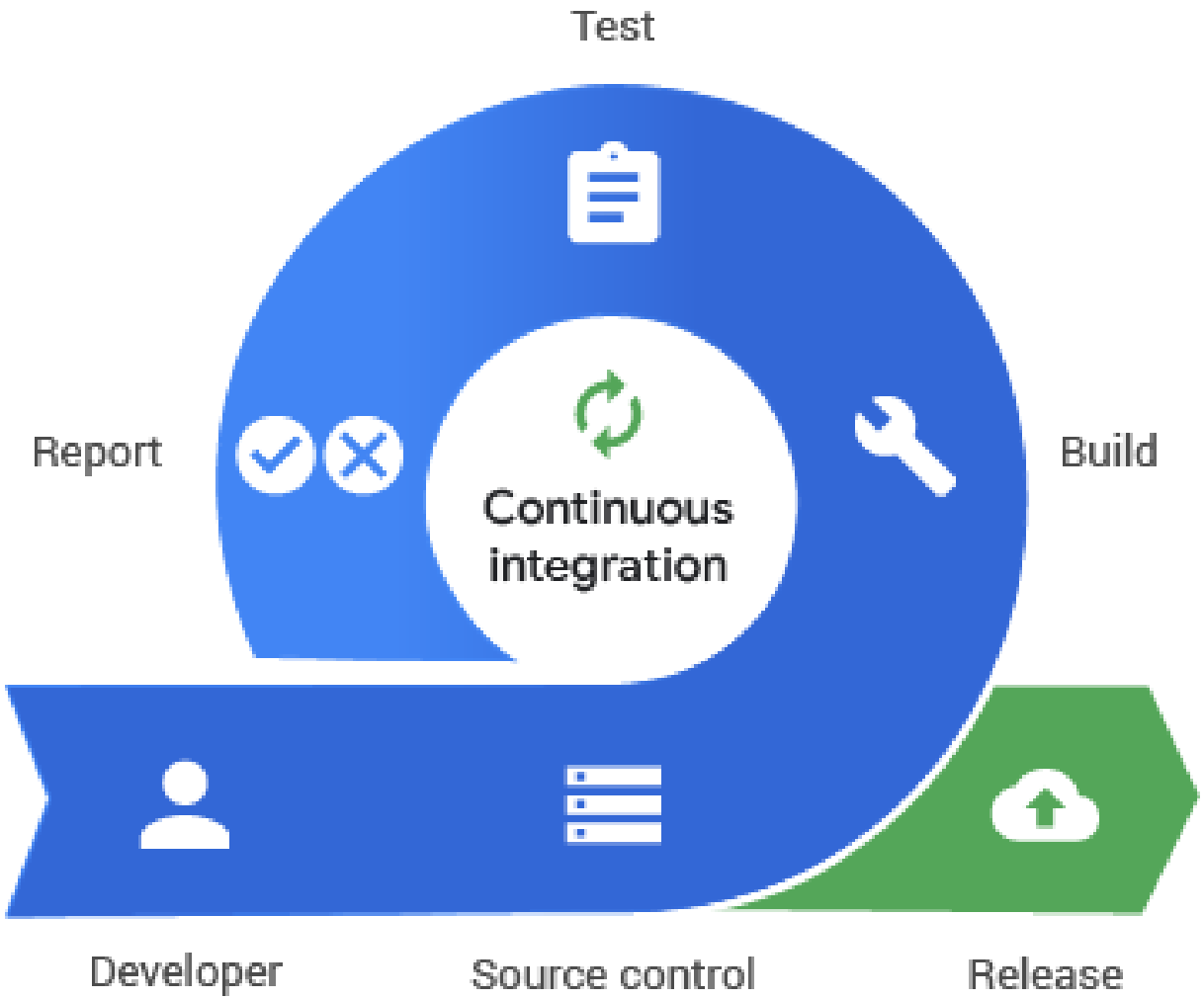
Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

12 de Janeiro de 2020

Motivation


Repository
Branch
Working Copy
Test
Regression
Integration
Green Master
Scale vs. Speed
Version Control Systems

Continuous Integration



- Popular **software development technique** that allows developers to easily check that their **code can build successfully** across various system environments.[1]

Regression Testing

- Performed between two different versions of software in order to **provide confidence** that the newly introduced features of the working copy **do not conflict** with the existing features. [2]

Test Suite Minimisation

"do fewer" approach, removing possible redundancies.

Test Case Selection

"do smarter" approach, selecting only relevant tests, given the type of change.

Test Case Prioritisation

also "do smarter" approach by running some tests first, increasing probability of early detection

Regression Testing - Test Suite Minimisation

Definition 1

Given a test suite T, a set of test requirements $R = r_1, ..., r_n$ that must be satisfied to yield the desired "adequate"testing, and subsets of T , $T_1, ..., T_n$ such that any test case t_j belonging to T_i can be used to achieve requirement r_i.[2]

Goal:

Try to find a subset T' of T: $T' \subseteq T$, that satisfies all testing requirements in R.

Possible Solution:

The union of test cases t_j in T_i's that satisfy each r_i, forming T'. (NP-complete problem)

Regression Testing - Test Case Selection

Definition 2

Given a program P, the version of P that suffered a modification, P', and a test suite T, find a subset of T, named T' with which to test P'.[2]

A lot alike to Minimisation, but with different goals:

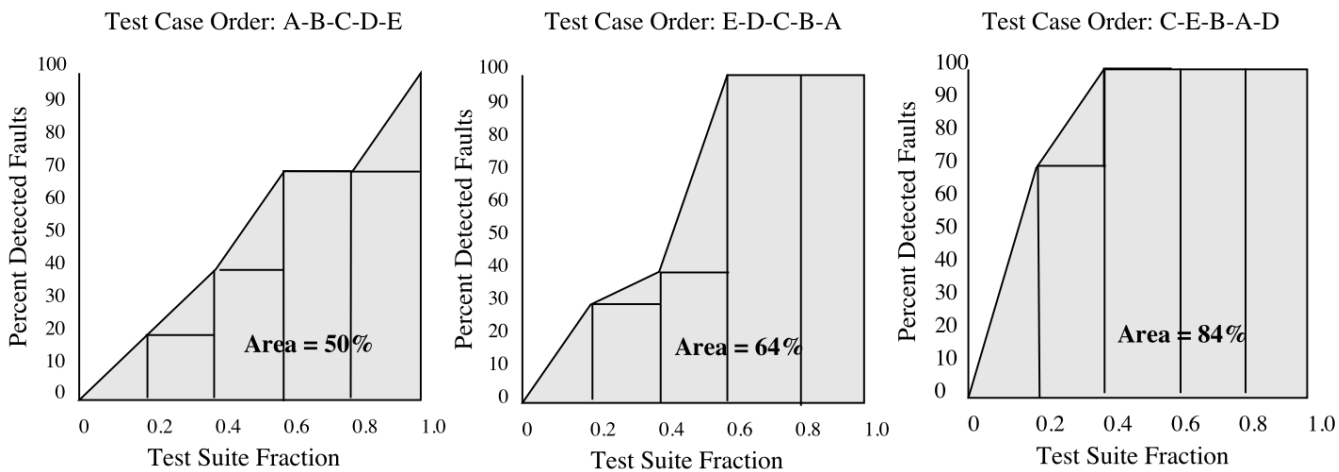
1. **Minimisation** - Apply the minimal amount of tests, without compromising code coverage in a single version, eliminating redundant tests.
2. **Selection** - Focus on changes made from a previous version to the current one, "cherry-picking"only relevant tests.

Definition 3

APFD Let T be a test suite containing n test cases and F the set of m faults revealed by T. Let TF_i be the order of the first test case that reveals the i^{th} fault.[2]

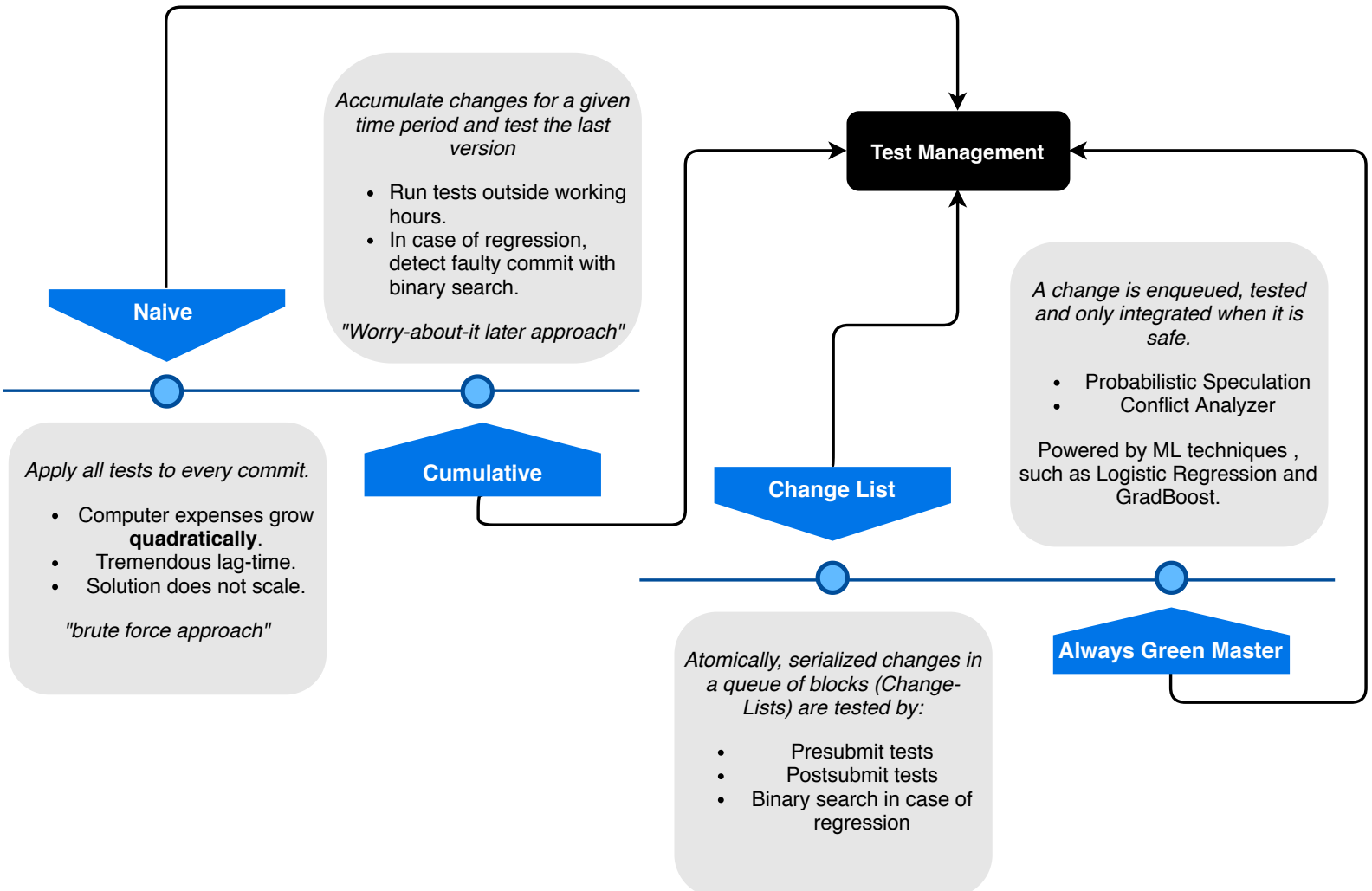
$$APFD = 1 - \frac{TF_1 + \cdots + TF_n}{nm} + \frac{1}{2n}$$

test	fault									
	1	2	3	4	5	6	7	8	9	10
A	x				x					
B	x				x	x	x			
C	x	x	x	x	x	x	x			
D					x					
E								x	x	x



- Higher values of APFD, imply high fault detection rates.

Strategies


Test Management
Naive
Apply all tests to every commit.
Computer expenses grow quadratically.
Tremendous lag-time.
Solution does not scale.
"brute force approach"
Cumulative
Accumulate changes for a given time period and test the last version
Run tests outside working hours.
In case of regression, detect faulty commit with binary search.
"Worry-about-it later approach"
Change List
Atomically, serialized changes in a queue of blocks (Change-Lists) are tested by:
Presubmit tests
Postsubmit tests
Binary search in case of regression
Always Green Master
A change is enqueued, tested and only integrated when it is safe.
Probabilistic Speculation
Conflict Analyzer
Powered by ML techniques , such as Logistic Regression and GradBoost.

Comparative Analysis

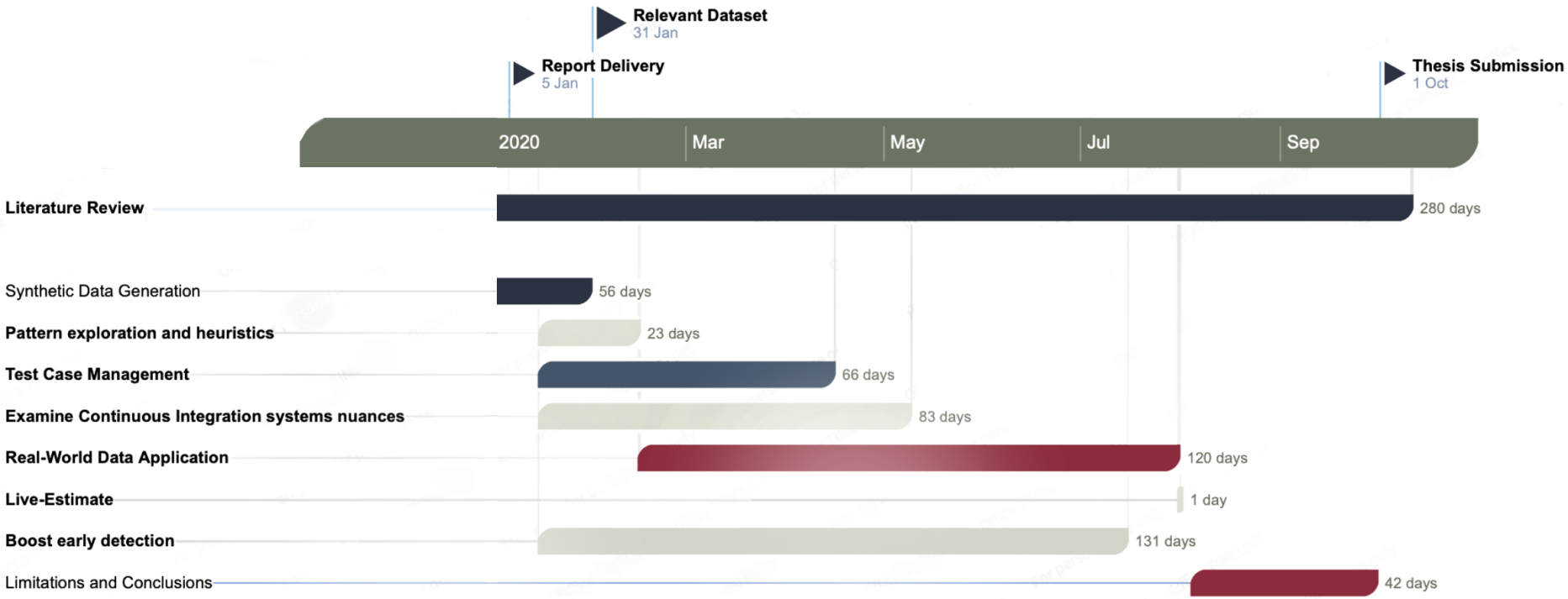
- There is a **trade-off** between **correctness** and **speed**.

Approach	**Correctness**	**Speed**	**Scales?**
Naive	Very High	Very Low	No
Cumulative	Medium	Low	No
Change-List	Medium	Medium	Yes
Always Green Master	High	High	Yes

Objectives

- **Detect common usage patterns**, in a controlled environment, by generating synthetic data.
  - Learn heuristics to automate fault detection process
- **Optimize systems using real world data.**
  - Analyse how different system configurations affect Continuous Integration
  - Reduce fault detection time
  - Provide a Live-Estimate of the Status of a Project
  - Given a commit, choose which chain of tests minimize pass/fail uncertainty

Planning


Relevant Dataset
31 Jan
Report Delivery
5 Jan
Thesis Submission
1 Oct
2020
Mar
May
Jul
Sep
Literature Review
280 days
Synthetic Data Generation
56 days
Pattern exploration and heuristics
23 days
Test Case Management
66 days
Examine Continuous Integration systems nuances
83 days
Real-World Data Application
120 days
Live-Estimate
1 day
Boost early detection
131 days
Limitations and Conclusions
42 days

References

[1] Mark Santolucito et al. *Statically Verifying Continuous Integration Configurations*. Paper on technical issues regarding Continuous Integration systems. 2018. arXiv: 1805.04473.

[2] Yoo Shin. "Extending the Boundaries in Regression Testing: Complexity, Latency, and Expertise". Tese de doutoramento. King's College London, 2009.

[3] Sundaram Ananthanarayanan et al. "Keeping Master Green at Scale". Em: *Proceedings of the Fourteenth EuroSys Conference 2019*. This source encompasses Uber's repository configuration and explains what's behind their fault detection process. ACM, 2019, 29:1–29:15.

[4] Celal Ziftci e Jim Reardon. "Who Broke the Build?: Automatically Identifying Changes That Induce Test Failures in Continuous Integration at Google Scale". Em: This source encompasses Google's repository configuration and explains what's behind their fault detection process. IEEE Press, 2017, pp. 113–122.

[5] Marko Ivankovic et al. "Code coverage at Google". Em: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, pp. 955–963.