Keeping Master Green with Machine Learning

João Lousada (84274)
Projeto MEFT

Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

8 de Janeiro de 2020

Problem Description

Crucial to have an efficient and reliable way to keep track of software changes

Specially in large and fast-paced companies, due to the **large amount of code** produced daily.[1]

Code Conflicts cause performance decrease.

There is not only the **problem of breakage** itself, but also the **need to detect exactly when it occurred and fix it**. [2, 1, 3]

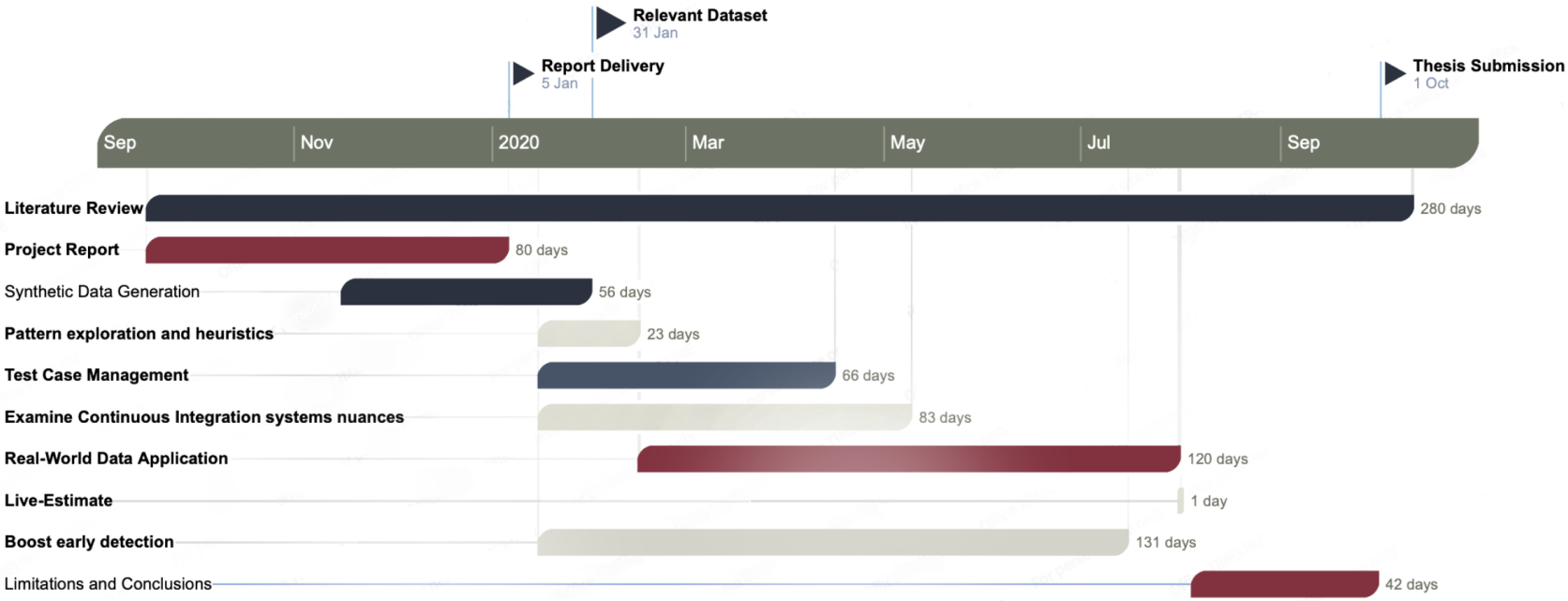
High demand for automated techniques to help developers keep the master green

Trying to **find a scalable solution**, given a constraint of time and computer resources.[1]

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
Sep
Nov
2020
Mar
May
Jul
Sep
Literature Review
280 days
Project Report
80 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

Repository, Working Copy & Branches

Repository

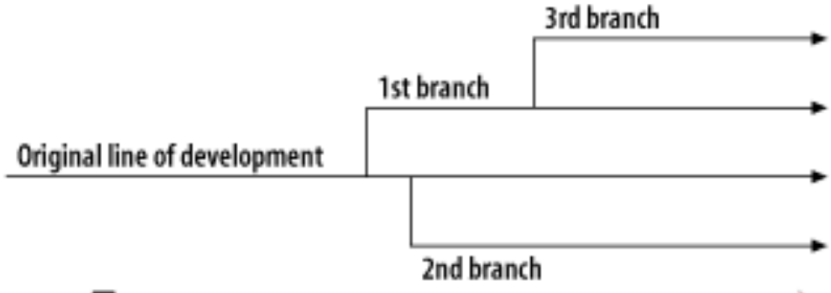
Data structure where all the current and historical files are **stored** and possibly, remotely, **accessible by others**. [4]

Working Copy

Local copy of a particular version, containing files or directories, on which the user is free to work on. [4]

Branch

When a change is uploaded from the working copy to the repository, a new version is created. **A branch contains multiple versions.** [4]



Continuous Integration

- Popular **software development technique** that allows developers to easily check that their **code can build successfully** across various system environments.[5]

How?

By conducting tests, assessing the **quality** of the software produced. [6]

Tests

- Validate if **code functionality matches a given criteria**.
- Check if **code dependencies** are **not affected**.
- **PASS/FAIL** outcome. (not very insightful)
- A **failed test** causes a **regression**.

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. [7]

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.[7]

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'.[7]

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 one previous version to the current one.

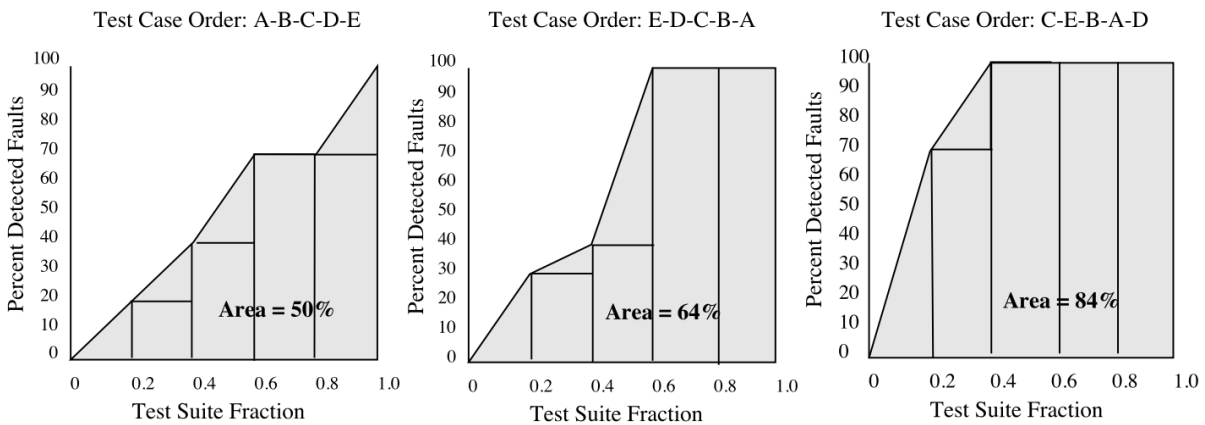
Regression Testing - Test Case Prioritisation

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.[7]

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

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

There are several ways to configure Continuous Integration systems:

1. **Naive Approach**
2. **Cumulative Approach**
3. **Change List Approach**
4. **Always Green Master Approach**

Naive Approach

Strategy

Run **every** test for every single commit.

Issues

1. As teams get bigger, both the number of commits and tests grow **linearly**, computer resources grow **quadratically**.
2. Continuous integration becomes **expensive** and **unefficient**.
3. **Solution does not scale.**

Cumulative Approach

Strategy

Accumulate changes for a given period of time and test whole batch. (*"Worry-about-it-later"approach)*

What if a regression is caused?

Find exactly which change caused the "bug", resorting to a search-algorithm.

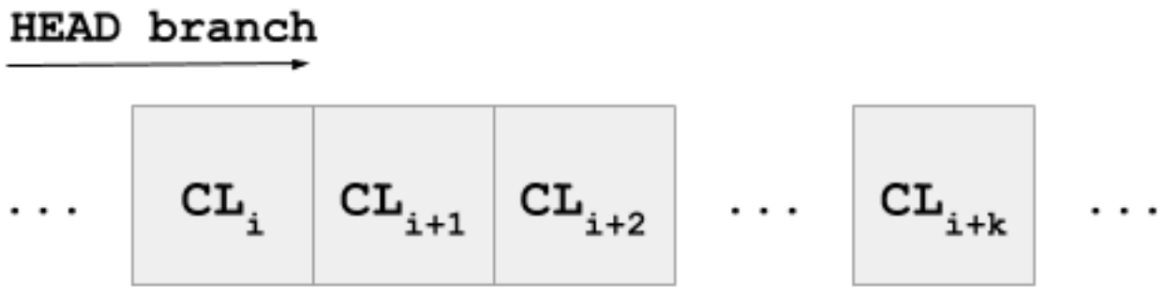
Issues

1. Great lag-time between failure and detection.
2. Possible development over faulty version.
3. heuristics to speed up detection are not implemented.
4. **Solution does not scale.**

Change-List Approach

Strategy

Atomically, serialize changes in a queue of blocks, each one denoted as a Change-List.[1]



Presubmit Tests

Run relevant small/medium tests before submission.

Postsubmit Tests

Execution of large/enormous time consuming tests.

Always Green Master Approach

Strategy

A change is **enqueued, tested** and only later on **integrated in the mainline branch**, once it is **safe**.[2]

Probabilistic Speculation

Guess the outcome of a given change, with machine learning, and then, for example, select the changes that are **more likely to succeed**.

Conflict Analyzer

Detection of changes that have **no dependencies**, pruning search tree, saving time.

Final Remarks

- 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

Outline

1. Learning patterns and "short-cuts" that can **reduce fault detection time significantly**.
2. Generating **synthetic data** and **simulating** each approach, to **learn heuristics automatically**.
3. Applying these techniques to a *real-world* dataset.

References

[1] 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.

[2] 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.

[3] 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.

[4] Michael Pilato. *Version Control With Subversion*. O'Reilly & Associates, Inc., 2004.

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

[6] B. Meyer. "Seven Principles of Software Testing". Em: *Computer* (2008). Introduction on the fundamental pilars of software testing in a general way, useful in defining concepts., pp. 99–101.

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