Glossary

CI Continuous Integration. 2

MapReduce a programming paradigm that allows large amounts of data to be processed in a distributed manner. 41

TCP Test Case Prioritisation. 2

TCS Test Case Selection. 2

TSM Test Suite Minimisation. 2

VCS Version Control System. 2

Chapter 1

Introduction

Given the complexity and rapid pace at which software is being built today, it is inevitable that sooner or later, bugs will emerge. These bugs can either be introduced by a malfunctioning new feature, or by breaking existing functionality (*a regression*). In order to detect bugs in an application before its users do, we require an adequate *testing infrastructure*.

This testing infrastructure consists of multiple *test cases*, collectively referred to as the *test suite* of the application. The quality of a test suite can be assessed in multiple ways. The first and most commonly used method is to measure which fraction of the source code is tested by at least one test case, a ratio which is indicated as the *coverage* of the application. Another possibility is to apply transformations to the source code and validate whether or not this results in a failed test case, a process indicated as *mutation testing*.

Ideally, this testing process should be automated and performed after every change to the source code. This process is generally very time-consuming, and as such has led to the creation of various automation frameworks and tools, collectively called Continuous Integration (CI). Common examples of CI practices are automatically running the test suite and estimating the code coverage after every pushed change to the Version Control System (VCS).

However, applying these practices and maintaining a qualitative test comes at a cost. Every addition or modification to the source code must be followed by at least one test case to validate its correctness. As a result of the speed at which the source code tends to grow, the test suite suffers from severe scalability issues. While it is desirable and ideally required to execute every single test case in the test suite, there are examples known to literature where this is not possible since this incurs an increasing delay in the development process, which in turn results in economic loss.

We can take three approaches to resolve this issue and reduce the time waiting for the test results: Test Suite Minimisation (TSM), Test Case Selection (TCS) and Test Case Prioritisation (TCP). The main subject of this thesis will be to implement a framework for TCP.

The structure of this thesis is as follows. The next chapter will introduce essential concepts used in modern software engineering. **??** will elaborate more on the three mentioned approaches and present accompanying algorithms. The implementation details of the new framework will be discussed in **??**. Afterwards, **??** will evaluate the performance of this framework and provide insights into the characteristics of a typical test suite. More specifically, this chapter will investigate the probability of (repeated) test failure and the average duration of a test run. Finally, **??** will present additional ideas and improvements to the framework.