Lay summary

Software

Computers, smartphones, cars, or even much simpler devices like alarm clocks and microwaves, every digital device that exists today consists of two distinct parts. The first, physical part is the hardware, which is the combination of mechanical bits and electrical wiring that enable a device to interact with the real world. The type of interaction can range from either very primitive to extremely complex, such as emitting an LED-light, producing a sound, or launching a rocket. The second part is the software, which is installed on the hardware of the device. Software is developed by software engineers using programming languages and instructs the hardware on what to do, and when.

Testing

Deciding on what is the "best" approach towards the development of software is an entire science on its own with two main conceptions. The traditional approach starts with a thinking phase, followed by a programming phase and finalised by a testing phase. In the first phase, the developers create a detailed design document that describes the required functionality of the final application. Next, the developers write computer code that implements the desired functionality. When this process is completed, the quality assurance team thoroughly tests the application. This testing phase exists in hardware as well. Consider, for example, crash tests conducted by car manufacturers. The purpose of these tests is to detect potential issues and anomalies (bugs) in the application before its end-users do. This phase is critical because bugs can result in financial loss or incur other disastrous effects, such as the explosion of two space rockets in the previous decade, mere seconds after ignition.

Continuous Integration

The urge to confine financial losses is even more prominent today, in the wake of the world economic crisis and the more recent COVID-19 induced crisis. While the aforementioned traditional approach works well for small projects, it suffers from severe scalability issues when the size of the application increases at today's pace, since the testing phase consumes valuable time, and time equals money. As a result, software

developers have shifted towards an Agile development approach. This approach encourages software developers not to release the entire application at once, but release an initial version with a reduced functionality set as soon as possible and add extra features iteratively. Additionally, the developers must include automated software tests and execute these every time they make a change, to reduce the probability of introducing bugs. Because this is a tedious task, additional software has been created that automatically executes these tests after every change, under the name of Continuous Integration (CI), as illustrated in Figure 1.

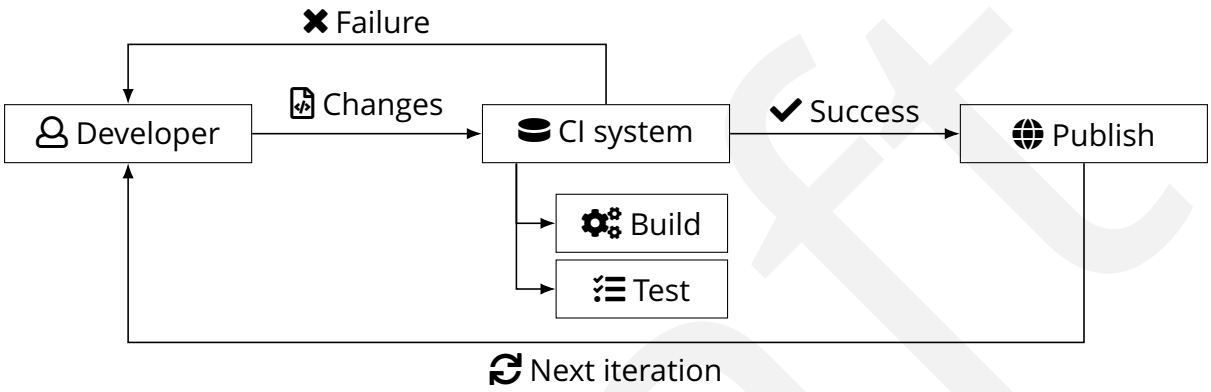

Figure 1: Continuous Integration (simplified).

Scalability

Nevertheless, Continuous Integration is not a silver bullet. In the initial stage of the project, the number of test cases will be rather small, therefore providing fast feedback to the developers in case of failure. However, as time progresses and the application grows, more test cases will be added that all need to be executed after every change. Eventually, this will consume a significant amount of time as well, thereby nullifying these benefits.

Solution

This thesis focuses on resolving this problem by introducing three techniques. The first two techniques are *Test Suite Minimisation* and *Test Case Selection*. These techniques attempt to predict which test cases are likely to fail, and as such, only execute those test cases with a high probability of failing. The third technique is *Test Case Prioritisation (TCP)*. As the name suggests, this technique will execute every test case in a specific sequence. The order of this sequence is determined by the predicted chance that the test case will fail, executing the most likely failing test cases as soon as possible.

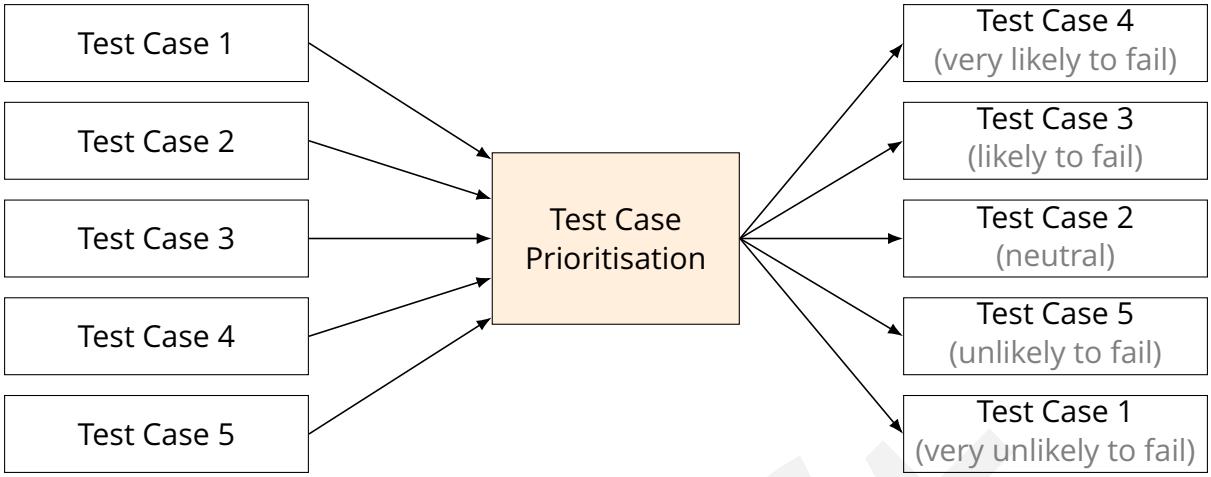

Figure 2: Test Case Prioritisation.

This thesis concentrates on TCP (Figure 2) since this technique guarantees that every failing test case will be executed eventually, even if a test case fails unexpectedly.

Results

The benefit of applying TCP on two existing applications has been analysed. The results are promising, the implemented optimisation framework executes, on average, only between $3\% - 5\%$ of the test cases. When examining the time it takes to detect a failing test case, the results indicate a reduction of more than $30 - 50$ times compared to the original, unprioritised execution.

Contents