The Effectiveness of Different Test Case Prioritization Techniques in Continuous Integration

University of Oulu
Information Processing Science
Bachelor's Thesis
Toni Bomström
30.03.2021

Abstract

Regression testing is often performed in software development to increase the quality of the software. Especially continuous integration (CI) environments face challenges related to the efficiency of the regression testing feedback loop. It is important to get feedback about the changes made as soon as possible so that work can be started on fixing the potential issues caused by the changes. Regression testing is optimized by prioritizing test cases into an order that increases the early fault detection rate. However, different test case prioritization (TCP) approaches and techniques set different requirements, such as access to source code, for the environment. As the CI environments set high time and resource constraints for the TCP, not all techniques are applicable and some care should be put into selecting a suitable TCP technique. This thesis consists of a literature review that aims to consider CI environments's special requirements, map out the most common TCP techniques as well as evaluate their applicability to these CI environments. Based on the benefits and drawbacks noted for the approaches, it seems likely that history-based TCP approaches could be the best fit for these resource intensive CI environments.

Keywords
regression testing, test case prioritization, continuous integration

Supervisor
Mika Mäntylä

Contents

1. Introduction

Software development is a process that is prone to errors. End user needs and requirements can be understood incorrectly, important requirements might go unnoticed, design might be flawed or incomplete, and the implementation likely contains bugs or unintended features. Mahali, Prasad and Mohapatra (2018) go as far as to say that “most of the software products are not accepted by the customers because they fail to satisfy the user requirements" (Mahali, Prasad & Mohapatra, 2018, p. 1064). In order to detect the faults and errors software testing is conducted. Roongruangsuwan and Daengdej (2010) describe software testing as a “process of validating and verifying that a program functions properly” (Roongruangsuwan & Daengdej, 2010, p. 45). This process consists of comparing the software against its functional and non-functional requirements (Roongruangsuwan & Daengdej, 2010). For small projects it may still be a feasible approach to run all of the test cases every time a new change is made, but it becomes less feasible as the number of test cases and the size of the software increase and the cost of executing the tests increases (Dahiya & Solanki, 2019; Dahiya, Solanki, Dalal & Dhankhar, 2020; Dahiya, Solanki & Dalal, 2019; Paygude et al., 2019; Mahdieh, Mirian-Hosseinabadi, Etemadi, Nosrati & Jalali, 2020; Khatibsyarbini, Isa, Jawawi, Hamed & Suffian, 2019).

To produce quality software, the software needs to be tested effectively. Regression testing is conducted to check that the previous functionality of the software has not been affected by the new changes made after the tests have passed the first time. Newly introduced faults can be detected by continuously testing the software upon making every change. However, the test suite size becomes a big overhead for large software projects that utilize continuous integration. The execution of all test cases for each change will at some point require more time and resources than is realistically feasible, so the testing effort needs to be further optimized. The use of test case prioritization (TCP) is a valid approach as the testing is carried out in a cost-effective way without affecting the quality of testing. In this thesis I will be assessing the applicability of the most common TCP techniques in the context of continuous integration. The research conducted by Paygude, Joshi, Bhattacharyya and Kim (2019) provided a strong baseline for this thesis and helped the development of the research question as well as the selection of focus (Paygude, Joshi, Bhattacharyya & Kim, 2019).

Continuous integration (CI) is a software development practice in which the developers integrate their changes rapidly (Mårtensson, Ståhl & Bosch, 2019; Elbaum, Rothermel & Penix, 2014; Spieker, Gotlieb, Marijan & Mossige, 2017). This is achieved by using CI environments that “automate the process of building and testing software” (Liang, Elbaum & Rothermel, 2018, p. 688). In order to optimize this automation, the testing aspect can be improved to allow for a more efficient feedback loop. This is where test case prioritization can be applied.

With the increasing popularity of CI it becomes important to consider how testing automation shall be handled in such highly dynamic and resource intensive environments. One solution

for this is to adopt TCP for reducing the required time to find potential faults. To contribute to the research of TCP's applicability to CI this thesis aims to answer the following research question:

- *How effective are the most common test case prioritization techniques in continuous integration development environments?*

To simplify the analysis and to direct the study, three sub questions are presented:

- *What are the most common TCP techniques?*
- *What are the context specific requirements for TCP in CI?*
- *What is the most important set of objectives for TCP in CI?*

This thesis is organized as follows. Chapter 2 introduces the background literature used for this thesis. The research method is explained in chapter 3. The results of the literature review are presented in chapter 4. Chapter 5 discusses the results. The thesis is concluded with chapter 6.

2. Background

This section introduces the concepts of regression testing, continuous integration and test case prioritization.

2.1. Regression testing

Regression testing is performed to find out whether changes made to the software have introduced new faults to the software's existing functionalities (Dahiya & Solanki, 2019; Dahiya et al., 2019; Dahiya et al., 2020; Paygude et al., 2019; Mahdieh et al., 2020; Khatibsyarbini et al., 2019). The tests are handled as test cases that test specific features, requirements, modules or areas of the code. These test cases are bundled up into a test suite that covers all of the test cases for a software, system or module under test.

However, as it is most often not feasible to carry out fully exhaustive testing in industrial projects, the test suites need to be constructed so that the maximum benefit is achieved with the least amount of test cases. Fortunately, solutions such as combinatorial testing allow generating test suites automatically based on input modeling. As the project advances, the test suite is likely going to increase in size, and each full execution of the suite is going to be more costly in terms of time and resources. This resource intensiveness can become a problem in continuous integration environments as it causes the feedback loop to increase as well.

2.2. Continuous Integration (CI)

Yu, Alégroth, Chatzipetrou and Gorschek (2020) make the claim that the goal of CI is to “continuously verify quality aspects of a software” in relation to its requirements (Yu, Alégroth, Chatzipetrou & Gorschek, 2020, p. 1). Additionally, Marijan, Gotlieb and Liaaen (2019) bring up that CI practices can help prevent and reduce the costs in fixing problems related to integration (Marijan, Gotlieb & Liaaen, 2019). One way to retain software stability and quality is to carry out regression testing for every commit as has been discussed by Liang et al. (2018). In these kinds of CI environments the developers commit their changes to a version control repository that is monitored by an integration build server. When this server detects a new commit it retrieves a copy of the changed code, makes a build out of it, runs the tests and finally informs the developer with a report of the results. (Liang et al., 2018.) The goal is to keep the codebase stable (Elbaum et al., 2014). The need for time efficiency of regression testing is especially important in projects that utilize continuous integration (Marijan et al., 2019). As the commits come in as a stream in CI, there is little to no time for computationally heavy analysis (Liang et al., 2018; Elbaum et al., 2014).

2.3. Test Case Prioritization (TCP)

A few different approaches for handling the cost-effectiveness of regression testing have been proposed. The most commonly mentioned approaches include Test Suite Minimization (TSM), Test Case Selection (TCS) and Test Case Prioritization (TCP) (Bian, Li, Guo & Zhao, 2018, Dahiya et al., 2020; Dahiya & Solanki, 2018). Both TSM and TCS aim to reduce the size of the test suite (Dahiya et al., 2019; Dahiya et al., 2020; Bian et al., 2018). Dahiya et al. (2019) explain that the TSM approach achieves this by discarding redundant test cases. In contrast to TSM, the TCS approach achieves the test suite reduction by selecting only the relevant test cases that deal with the parts of the software that are affected by the changes made. (Dahiya et al., 2019.)

The TCP approach differs from the two others as it does not alter the number of test cases but rather focuses on their order of execution (Mahdieh et al., 2020; Bian et al., 2018; Mahali et al., 2018). By arranging the test cases so that the faults and errors are found early on, there is no need to run all of the test cases to find out whether the tests pass or not. The quality of testing is not reduced as no test cases are left out (Paygude et al., 2019; Bian et al., 2018; Haghighatkhah, Mäntylä, Oivo & Kuvaja, 2018). This prioritization can be done in many different ways by using different factors and by adjusting how they are weighed. The terms TCP technique and TCP approach are used interchangeably in this thesis even though the terms are generally used for different things. Usually TCP approaches include multiple different TCP techniques. Different TCP techniques or approaches consider different factors and weights and may have differing goals and objectives for what is the preferred prioritization order.

TCP techniques aim to meet specific performance goals or objectives by arranging and scheduling the test cases into an optimal order (Roongruangsuwan & Daengdej, 2010; Rothermel, Untch, Chu & Harrold, 2001; Dahiya et al., 2020; Khanna, Chaudhary, Toofani & Pawar, 2019; Malishevsky, Ruthruff, Rothermel & Elbaum, 2006). The optimal order depends on the objective or objectives of the TCP. Paygude et al. (2019) conclude that the main objective common to most TCP techniques is the “early detection of faults with minimum cost” (Paygude et al., 2019, p. 1265). The motive for this is that the faults present in the system should be found early in the testing process so that the developers can start working on them as soon as possible (Rothermel et al., 2001). Other major objectives include “code coverage, minimum execution time, severe fault detection and customer requirement priority” (Paygude et al., 2019, p. 1265). As for how to select a technique, Rothermel et al. (2001) bring up that the intent behind the selection should be to increase the likelihood of meeting the important goals in comparison to a random ordering of test cases (Rothermel et al., 2001).

3. Research method

The research method chosen for this thesis is the narrative literature review. The research question this thesis aims to answer is the following.

RQ: *How effective are the most common test case prioritization techniques in continuous integration development environments?*

Sub-questions are used to answer the research question. The following sub-questions were chosen.

Sub-RQ1: *What are the most common TCP techniques?*

Sub-RQ2: *What are the context specific requirements for TCP in CI?*

Sub-RQ3: *What is the most important set of objectives for TCP in CI?*

Sub-RQ4: *How effective are the most common TCP techniques in achieving these objectives?*

Scopus was used for the searching of the literature, and some of the sources used in relevant articles and papers were sought out as well for further details. The search queries presented below were performed on Scopus, and the relevant articles and papers were hand-picked from the results. The study selection was not systematic in the sense of systematic literature reviews, but some self-defined guidelines were used for it. The first step of evaluating relevance was to consider the titles of the found papers and articles. Most titles with mentions of or implications toward the subjects of improving test automation, evaluating or presenting TCP techniques or addressing testing activities in CI were

considered further. These papers and articles were taken in for closer inspection. The second evaluation step was to consider the abstracts. The general idea of study selection at this stage was to evaluate whether the study would contribute either to the background knowledge or to answering the research questions. In addition to the previously mentioned study selection criteria based on the title, the studies providing more insight into the constraints and other characteristics of the CI environments were included for further inspection. These other characteristics consist of all the CI specific attributes that differentiate them from other types of software development environments.

```
1. TITLE-ABS-KEY ( ( software W/6 ( develop* OR engineer* ) ) AND
"test case priorit*" )

2. TITLE-ABS-KEY ( ( software W/6 ( develop* OR engineer* ) ) AND
test* AND "continuous integration" )

3. TITLE-ABS-KEY ( ( software W/6 ( develop* OR engineer* ) ) AND
(test* W/6 automat*) AND "continuous integration" )
```

The first query aims to gather all of the studies considering TCP in the context of software development or software engineering. The second one does the same for CI. The third one narrows down the second one by adding the constraint of testing automation. The scope was kept broad so that many different aspects of testing, TCP and CI were included. This helped with narrowing down the focus of the research questions as I explored the subject areas.

However, these queries were selected and executed at a time when the research questions had not yet been narrowed down to the point where they are now. The background research started with TCP, expanded into the evaluation of TCP techniques and finally took upon the constraint of CI environments.

4. Literature review

This section presents the included literature in relation to the research questions.

4.1. TCP techniques

The literature review begins with mapping the most common TCP techniques. Figure 1 depicts the TCP techniques that were identified from the literature. The literature review conducted by Dahiya et al. (2019) highlights the categorization of TCP approaches into search-, coverage-, history-, requirement-, fault-, cost- and risk-based. The authors decided to bundle up the rest of the approaches as the “other approaches” whilst mentioning the

model- and web application -based techniques and the techniques “to be applicable on real-world systems”. (Dahiya et al., 2019, p. 1524.) The requirement-, coverage- and history-based techniques are also mentioned by Roongruangsuwan and Daengdej (2010) as well as Paygude et al. (2019) (Roongruangsuwan & Daengdej 2010; Paygude et al., 2019). For a comprehensive list of which studies mentioned which techniques, see table 1. Studies that directly address the TCP technique or less directly describe the approach are considered to mention the specific technique. Studies that mention a technique in a reference to other works are also included. However, techniques mentioned directly only in a single included study are noted to be techniques only if another study mentions them.

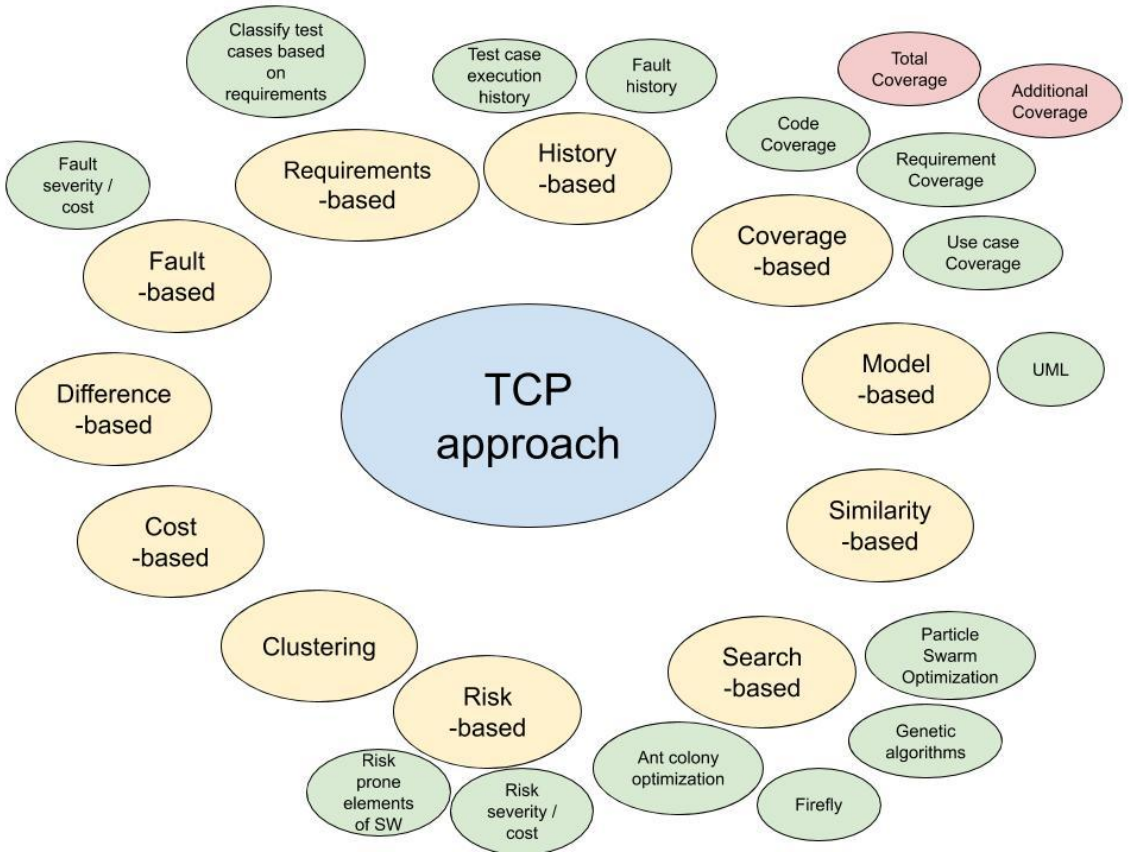

Figure 1. Identified TCP techniques

Table 1. Mentioned TCP techniques

TCP technique	**Studies that mention it**
Coverage-based (17)	Bian et al., 2018; Chen, Zhu, Chen, Towey, Kuo, Huang & Guo, 2018;Dahiya et al., 2020; Dahiya & Solanki, 2018; Dahiya & Solanki, 2019; Dahiya et al., 2019; Dahiya et al., 2020;

	Elbaum et al., 2014; Haghighatkhah et al., 2018; Khanna et al., 2019; Mahali et al., 2018; Mahdieh et al., 2020; Malishevsky et al., 2006; Marijan et al., 2019; Paygude et al., 2019; Roongruangsuwan & Daengdej, 2010; Rothermel et al., 2001; Spieker et al., 2017;
History-based (17)	Bian et al., 2018; Chen et al., 2018; Dahiya & Solanki, 2018; Dahiya & Solanki, 2019; Dahiya et al., 2020; Dahiya et al., 2019; Elbaum et al., 2014; Haghighatkhah et al., 2018; Khanna et al., 2019; Mahali et al., 2018; Mahdieh et al., 2020; Malishevsky et al., 2006; Marijan et al., 2019; Paygude et al., 2019; Roongruangsuwan & Daengdej, 2010; Rothermel et al., 2001; Spieker et al., 2017;
Fault-based (12)	Chen et al., 2018; Dahiya & Solanki, 2018; Dahiya et al., 2019; Dahiya et al., 2020; Khanna et al., 2019; Khatibsyarbini et al., 2019;

	Mahali et al., 2018; Mahdieh et al., 2020; Malishevsky et al., 2006; Paygude et al., 2019; Roongruangsuwan & Daengdej, 2010; Rothermel et al., 2001;
Search-based (11)	Bian et al., 2018; Dahiya & Solanki, 2018; Dahiya & Solanki, 2019; Dahiya et al., 2019; Dahiya et al., 2020; Elbaum et al., 2014; Haghighatkhah et al., 2018; Khanna et al., 2019; Khatibsyarbini et al., 2019; Paygude et al., 2019; Spieker et al., 2017;
Model-based (9)	Chen et al., 2018; Dahiya & Solanki, 2018; Dahiya et al., 2019; Haghighatkhah et al., 2018; Mahali et al., 2018; Mahdieh et al., 2020; Marijan et al., 2019; Paygude et al., 2019; Spieker et al., 2017;
Requirements-based (9)	Dahiya & Solanki, 2018; Dahiya & Solanki, 2019; Dahiya et al., 2019; Dahiya et al., 2020; Khanna et al., 2019; Mahali et al., 2018; Paygude et al., 2019; Roongruangsuwan & Daengdej, 2010;

	Rothermel et al., 2001;
Clustering (8)	Chen et al., 2018; Dahiya & Solanki, 2018; Dahiya & Solanki, 2019; Dahiya et al., 2019; Khanna et al., 2019; Mahdieh et al., 2020; Roongruangsuwan & Daengdej, 2010; Spieker et al., 2017;
Cost-based (7)	Dahiya & Solanki, 2018; Dahiya & Solanki, 2019; Dahiya et al., 2019; Dahiya et al., 2020; Malishevsky et al., 2006; Roongruangsuwan & Daengdej, 2010; Rothermel et al., 2001;
Similarity-based (6)	Chen et al., 2018; Haghighatkhah et al., 2018; Khanna et al., 2019; Khatibsyarbini et al., 2019; Mahdieh et al., 2020; Paygude et al., 2019;
Risk-based (3)	Dahiya & Solanki, 2018; Dahiya et al., 2020; Mahali et al., 2018;
Difference-based (2)	Malishevsky et al., 2006; Roongruangsuwan & Daengdej, 2010;

The TCP approaches identified in this thesis are not definitive nor are they comprehensive. They have a bit of overlap and a TCP implementation or solution could be utilizing multiple of these TCP approaches simultaneously. For example, an approach can be considered to be both risk- and cost-based at the same time if it bases the prioritization on the costs of identified risks. Additionally, the definitions for these approaches are not in no way comprehensive as only the most evident approaches are covered.

The **coverage-based** TCP approaches aim to prioritize test cases by considering the coverage metrics for each test case. As an example, the typical coverage metrics include code coverage, requirement coverage or use case coverage. The general idea is to maximise the coverage of a specific metric by prioritizing test cases that increase the coverage the most over those that overlap with those that have already been prioritized higher. The **history-based** TCP approaches take the historical testing information into account in prioritization. The exact nature of this information or its utilization is rather vague, but typically test case's prior occasions of uncovering faults causes the test case to be prioritized higher than test cases that have not uncovered faults in the past. The **fault-based** TCP approaches, as the name suggests, have a focus on the faults themselves. The TCP could be based on finding specific faults or faults in specific parts of the system, but this TCP approach considers the faults from a more central viewpoint. The **search-based** TCP approaches utilize the search algorithms for solving the prioritization problem. The TCP itself needs to be mapped into the algorithm-specific problem format in order to be solvable using it. The **model-based** TCP approaches base the TCP on the use of models. This approach considers the test cases from the viewpoint of these models, but the exact metrics and the prioritization itself is left more open ended. For example, coverage-based TCP approach could be combined with a model-based TCP approach to create a solution that aims to maximise the model coverage.

When requirements are in the center of the prioritization, the approach can be considered **requirements-based**. This may overlap somewhat with coverage-based TCP, but the key difference is that coverage-based approaches aim to optimize coverage whereas requirements-based approaches may consider other attributes as well. As an example, different requirements may have differing priorities, which would not be considered in prioritization when using coverage-based approaches. **Clustering** is an approach in which test cases are clustered based on certain attributes. For example, Chen et al (2018) used K-means and K-medoids clustering algorithms to cluster test cases based on their number of methods and objects as well as the similarities of their object and method invocation sequences (Chen et al., 2018). **Cost-based** approaches carry out test case prioritization based on expected costs associated with the test cases. Malishevsky et al (2006) described that the costs can be, for example, the costs of running the test case or the costs of the potential faults that could be found with the test case (Malishevsky et al., 2006). Somewhat similarly to clustering, the **similarity-based** approaches utilize metrics by which they evaluate the similarities between different test cases. The idea is that similar test cases tend to find similar faults and dissimilar test cases tend to find dissimilar faults. (Haghighatkhah et al., 2018.) The **risk-based** approaches focus on utilizing system's risk information for prioritizing test cases. For example, test cases that have the highest potential for finding high risk faults are prioritized over others. Malishevsky et al (2006) introduced the approach of **difference-based** TCP. This approach compared the versions prior and after the changes to identify lines of code that had been modified. Prioritization was given to the test cases that covered the most areas in code that had been modified. (Malishevsky et al., 2006.)

Paygude et al. (2019) provide a comparison of the most studied TCP techniques. Their comparison table displays some differences between coverage-, history- and model based

approaches. This comparison table has been included into this thesis as table 2. The code coverage- and history based TCP are noted to have high average percentage of faults detected (APFD) values indicating that they are highly efficient at early fault detection. However, code coverage based approaches are noted to be highly time consuming whereas history based approaches are considered only moderately time consuming. Model based approaches tend to have lower APFD values but also have less demanding time requirements. (Paygude et al., 2019.)

Table 2. (Paygude et al., 2019)

Parameter	**Code Coverage**	**History Based**	**Model Based**
Data used	Code coverage data	History of faults and test cases	UML model, OOP models
Overhead	Continuous updation of code data such as lines of code, conditions, decisions etc	Maintaining history of fault and test covering faults	updation of model as per new code churn
Bug detection in	testing phase	testing phase	design phase
Access to source code	required	sometimes required	not required
APFD	High	High	Moderate
Time requirement	High	Moderate	Less
Rate of detecting severe faults	Moderate	High	Moderate
Application best suited for	Small	Moderate	High

The most common TCP techniques, based on the number of included studies discussing them, are coverage-based, history-based, fault-based and search-based. Paygude et al. (2019) had somewhat similar results as they noted the coverage- and history-based as the most commonly used TCP approaches as well. However, they also mentioned the model-based TCP as one of the most common. (Paygude et al., 2019.) According to the results in this literature review, model-based TCP seems to be moderately common rather than among the most common. The popularity of model-based TCP is also brought up by Dahiya et al. (2019) as they claim its popularity is increasing among researchers thanks to its profitability (Dahiya et al., 2019).

The search based TCP approaches are among the most mentioned approaches. The selection of a search-based TCP algorithm is noted problematic by Bian et al. (2018). They

bring up that many search algorithms exist and that there is a lack of guidance for selecting an algorithm. Further complicating the matter is that while the state-of-the-art algorithms may perform well in certain scenarios, changes in the testing scenario might render them less effective. The authors propose a concrete framework that aims to “select heuristic algorithms dynamically for various TCP optimization objectives and various subjects.” (Bian et al., 2018, p. 960.)

Based on the number of mentions within the included studies, clustering is among the fairly well researched approaches of TCP. Consequently, one of the included studies focuses on said approach. Chen et al. (2018) describe that the idea behind clustering is to divide the test cases into different clusters so that all test cases within a cluster are similar in some regard and different from test cases from other clusters. After the clustering, the prioritized order is formed by selecting the test cases from the clusters using a sampling strategy. (Chen et al., 2018.)

Somewhat similarly well researched, model-based TCP, is also among the moderately often mentioned techniques. As is the case with clustering, one study focuses on this technique specifically as well. Mahali et al. (2018) propose an approach where the system is modeled using Unified Modeling Language (UML), test cases are generated based on the models, a “frequent pattern of affected node” is generated using “association rule mining” and finally the test cases are prioritized based on their business criticality values. (Mahali et al., 2018, p.1066.) Mahali et al. (2018) explain that the benefit of the model-based approach is that the dependencies between different classes and functions are more clear than in code coverage -based approaches. In addition to this, the advantage of their proposed approach takes into account the different types of faults. However, the drawback of this approach is noted to be that it does not consider the non-functional aspects of the system. (Mahali et al., 2018.)

The validity of the results in this study is heavily affected by the sample of studies included in this literature review. The diversity among the included studies is low. The included literature consists mostly of recently published works, and a few of them are from the same authors. These characteristics might, for example, skew the perceived TCP approach commonality.

4.2. TCP evaluation metrics

One commonly applied metric for measuring the effectiveness of a TCP algorithm is the average percentage of faults detected (APFD) introduced by Rothermel et al. (2001). APFD is a value between 0 and 100 and a higher value depicts a higher fault detection rate. (Rothermel et al., 2001.) However, as this metric only considers the most common goal of TCP techniques, the early fault detection, its use leads to one-sided results. In realistic environments other goals such as cost-efficiency, fault severity and related requirements likely also need to be considered. For example, CI environments face a constant stream of changes and therefore cost-efficiency is a necessity to avoid congestion and long delays (Liang et al., 2018).

To include the cost aspect in the evaluation, Malishevsky et al. (2006) enhanced the metric creating the cost-cognizant APFD that they refer to as APFDc. The original APFD focuses on the relation between the percentage of faults detected and the percentage of the test suite executed. Slightly differently, APFDc considers the percentage of total fault severity detected in relation to the percentage of total test case cost incurred. This allows the APFDc to account for the test case and fault costs using the “cost of a test case” and “severity of a fault” as metrics for determining the effectiveness of the order. The authors left these metrics open for interpretation so that they could be adjusted to fit different scenarios. Their examples for determining the cost include test execution time, hardware costs and engineers’ salaries. (Malishevsky et al., 2006.)

As for the testing criteria of TCP, Khanna et al. (2019) mention that there is a separation to uni- and multi-objective. This separation expands to the classification of the TCP problems into single objective and multi-objective optimization problems. The uni-objective testing criteria considers one factor that is to be either maximized or minimized. Conversely, the multi-objective testing criteria considers multiple factors some of which are to be minimized and the rest maximized. (Khanna et al., 2019.)

4.3. Continuous integration

Liang et al. (2018) discuss the challenges related to testing efforts in CI. They claim that it is difficult to apply the traditional TCP techniques to CI (Liang et al., 2018). This was also noted by Elbaum et al. (2014). There is no time for the data gathering and analysis required for the prioritization due to the high number of commits (Elbaum et al., 2014; Liang et al., 2018). Therefore TCP techniques that involve program analysis and code instrumentation can not be effectively adapted to CI environments (Elbaum et al., 2014; Liang et al., 2018). As a result, Liang et al. (2018) make the claim that the TCP techniques used in CI environments tend to deal with test suites instead of test cases. They continue to discuss that the approach of prioritizing commits instead of test suites is likely to improve the early detection of faults. However, they do note that the usefulness of this commit-based prioritization increases as the pace of commits surpasses the number of computing resources so that the commits start to queue up. (Liang et al., 2018.)

The goal of early fault detection in CI is also brought up by Spieker et al. (2017). They also make a note of the limited time available for testing in CI environments. Because of this they consider that time-effective algorithms should be prioritized over more costly and complex ones. In the paper they introduce a supposedly lightweight reinforcement learning approach that utilizes both TCS and TCP in CI environments. The approach uses the preceding CI cycles’ historical data of the test cases in assessing the test cases’ capabilities to detect failures. (Spieker et al., 2017.)

The algorithm introduced by Liang et al. (2018) has four highlighted characteristics; It is fast, continuous, commit-focused and resource-aware. The operational speed of CI motivates the

prioritization's need to be lightweight and effective. As the commits come in as a stream, the continuous re-prioritization becomes a desired trait. Similarly, the focus on prioritizing commits instead of test cases or test suites is important for improving early fault detection in the higher scale posed by the CI environment. Due to the continuous nature of CI testing, the computing resources might change over time and therefore the adaptability to resource availability is valued. (Liang et al., 2018.)

Elbaum et al. (2014) make the division of CI testing into pre- and post-submit testing. The pre-submit testing is the initial testing made to detect integration errors before the changes are merged into the codebase. Correspondingly, the post-submit testing is performed on the commits that have been submitted to the codebase. The approach introduced by the authors is based on the use of test suite history in the form of "failure windows". TCS is used for the pre-submit testing and TCP for the post-submit testing. The results of the empirical study conducted by the authors seem to indicate that the prioritization technique that they used performs better than having no prioritization. Overall, the approach of using TCS for a pre-submit test run and TCP for a post-submit test run aims to balance the test execution time with the early failure detection. The pre-submit testing brings a potential delay to the fault detection. (Elbaum et al., 2014.)

Another point brought up by Elbaum et al. (2014) is that most studied TCP techniques can be divided into two periods of time. Some of the analysis can be performed in a "preliminary period" of testing, which takes place prior to the actual testing. For example, the code coverage data can be gathered or the system models can be created. The information that these tasks produce can then be utilized in the second period, the "critical period" of testing. The critical period includes carrying out the actual TCP and testing and the strive for efficiency in the feedback loop is constraining the available time. However, the authors note that this division is typically not present in CI environments. (Elbaum et al., 2014.)

Marijan et al. (2019) introduced an algorithm for reducing test case redundancy in CI environments using code coverage and history analysis. However, their focus was on the development of highly configurable software, which means their results may not be as well generalizable to the entire scale of CI software development. (Marijan et al., 2019.) Even with this limitation, it seems plausible that other CI environments could benefit from reducing redundancy as well.

It comes as no surprise that a dynamic and complex problem such as optimally prioritizing test cases according to some metrics leads to the development of solutions that utilize artificial intelligence. Khatibsyarbini et al. (2019) implemented a swarm intelligence -based firefly algorithm for carrying out the TCP. The firefly algorithm, to put it simply, views the optimization problem in terms of fireflies, their positioning and the light intensity of the fireflies. The objectives are mapped to denote higher light intensity and the behaviour of fireflies is determined by the light intensity of other fireflies. The solution for the problem is represented as the shortest path that passes through all of the fireflies. As for solving the TCP problem, the test cases are the fireflies, the light intensity of the firefly is represented by the weight given to the test case using "Term Frequency-Inverse Document Frequency", and the higher similarity of test cases is considered as a longer distance between them. A path

that visits all fireflies once is considered a prioritized ordering of test cases. A path that has the shortest overall distance is considered the optimal solution therefore depicting the optimal order of test cases. (Khatibsyarbini et al., 2019.)

Somewhat similar approach is discussed by Haghighatkhah et al. (2018). Their work focuses on achieving a high variation and diversity in test cases by applying a “similarity based test prioritization“. This TCP technique assigns higher priority to test cases that differ the most from the ones that have already been prioritized. The approach was noted to be more efficient than random test case ordering. This result bears the implication that prioritizing variation and diversity in test cases increase early fault detection rate. (Haghighatkhah et al., 2018.)

5. Discussion

In order to answer the research question, the literature review aims to answer the sub-questions.

5.1. Sub-RQ1: What are the most common TCP techniques?

Based on the studies included in this literature review, it is not possible to unambiguously and indisputably determine which TCP techniques are the most common. According to Dahiya & Solanki (2018) there have been a few changes in the focus of the research regarding the different approaches (Dahiya & Solanki, 2018). To determine the most common TCP techniques based on the research focus alone would do injustice to the industry’s point of view. The inclusion of grey literature could have provided a better understanding of the industry’s considerations regarding the use of TCP techniques.

To have a way of justifiably determining the most common techniques, the number of mentions of the techniques within the included studies is used as a metric here. Both direct mentions and indirect descriptions of the TCP techniques are considered as mentions of the technique in question. Using this metric, coverage- and history-based techniques are the most common TCP techniques within the included studies. Fault- and search-based are slightly less common, but still among the most common techniques.

5.2. Sub-RQ2: What are the context specific requirements for TCP in CI?

The characteristics that Liang et al. (2018) highlighted in their algorithm could be seen as the desired requirements for the TCP techniques adapted to CI environments. The algorithm was noted to be fast, continuous, commit-focused and resource-aware. (Liang et al., 2018.) However, as this thesis focuses on the traditional TCP techniques, the focus on commits rather than test cases or test suites is out of the scope. The requirement to be both effective yet lightweight means that analysis intensive techniques lose viability in the search for an optimal solution. As Liang et al. (2018) put it, the “prioritization needs to be fast enough so that the gains of choosing the right commit are greater than the time required to execute the prioritization algorithm” (Liang et al., 2018, p. 691). The continuity and resource-awareness should not directly affect the selection of TCP technique but rather pose requirements for the specific algorithm itself. Additionally, the required extra computation is directly influencing the lightweightness of the solution.

Chen et al. (2018) bring up the common division of testing into black-box and white-box testing in the context of TCP. They classify code coverage, program model and fault detection history as part of white-box information and the test input and output information as black-box information. (Chen et al., 2018.) With this division in mind it could be stated that the TCP in CI tends to lean towards black-box testing with some regards to white-box testing. Heavy analysis of the information available in white-box testing increases the amount of computation required and therefore slows down the process to a level not viable in CI environments. However, the viability of the analysis of fault detection history as part of TCP efforts in CI has been demonstrated by Spieker et al. (2017). The approach of lightweight analysis and lack of code instrumentation was also applied by Elbaum et al. (2014). They note that these characteristics make their approach appropriate for the CI process. (Elbaum et al., 2014.)

5.3. Sub-RQ3: What is the most important set of objectives for TCP in CI?

Liang et al. (2018) focused on optimizing their TCP algorithm around early fault detection. This objective was also noted to be the most common objective among different TCP techniques (Paygude et al., 2019). The benefit of early fault detection is that the feedback time is reduced allowing the developers to start working on the faults even before the testing is fully completed (Elbaum et al., 2014). The limitation of time for computation in CI environments means that the cost-effectiveness becomes a limiting factor for the selected approach. There is little value in a high early fault detection rate if the CI system can not overcome the number of changes made due to slow and heavy computational analysis used

for the prioritization (Liang et al., 2018). Spieker et al. (2017) also seem to lean towards more cost-effective solutions for TCP in CI.

5.4. Sub-RQ4: How effective are the most common TCP techniques in achieving these objectives?

Some of the included studies concluded with the claim that higher efficiency would be achieved by making hybrids out of multiple different TCP techniques (Dahiya & Solanki, 2019; Mahdieh et al., 2020). The effectiveness of a hybrid in comparison to the traditional techniques has been demonstrated by Mahdieh et al. (2020). They enhanced a coverage based TCP technique by including the fault-proneness estimations into the prioritization. This fault-proneness is approximated by analysing the bug history of the software so the approach is a hybrid of coverage- and history-based TCP techniques. (Mahdieh et al., 2020.)

Using a theoretical example, Liang et al. (2018) pointed out that even with an optimal test suite prioritization, which they referred to as optimal intra-commit order, the fault detection would not change significantly in an CI environment. In comparison, the optimal commit-based prioritization, which they referred to as optimal inter-commit order, remarkably outperformed the optimal test suite prioritization order. Although being a single case and focusing on the optimal prioritization orders, this example goes to show that inter-commit prioritization can outperform intra-commit prioritization to a meaningful extent. (Liang et al., 2018.)

However, the scope of this thesis is on the traditional TCP techniques rather than the hybrids or adaptations. While more optimal approaches do exist, the goal here is to evaluate the effectiveness of the traditional TCP techniques in the context of CI. Seeing as the goal of cost-effectiveness in CI has seen praise from the researchers (Spieker et al., 2017), it seems appropriate to consider it as a baseline for the evaluation of different TCP techniques. The metric of early fault detection rate has been heavily used for assessing the effectiveness of TCP techniques and has been utilized, for example, by Spieker et al. (2017) and Liang et al. (2018).

The model-based approach requires up-to-date models, which could cause an overhead for TCP computation. This is somewhat in disagreement with the requirements set by CI environments. The computation required to keep the models up-to-date does not fit in the time and resource constrained environments and causes an overhead for an otherwise agile way of working. Similarly, the coverage-based approach also often requires heavy computation (Spieker et al., 2017). While coverage-based TCP techniques are noted to be very effective as well as well researched and used in practice (Malishevsky et al., 2006), the cost of the analysis decreases its viability for the CI environments.

The most applicable TCP technique for CI environments seems to be the history-based TCP. This approach is widely known and studied. Analysis of the test cases' historical data is reasonably lightweight in comparison to other approaches, and has proven useful and

efficient in different TCP implementations. The CI environments' heightened requirement for efficiency and lightweightness can therefore both be met reasonably well using history-based TCP techniques.

6. Conclusions

The objective of this thesis was to evaluate the applicability of different TCP approaches to CI environments. A literature review was conducted to map out the most common TCP techniques and to evaluate their applicability to the CI environments. The CI environments are constrained by the rapid stream of changes that are to be scheduled for builds, TCP and testing. Different TCP techniques attempt to optimize the testing feedback loop by increasing the early fault detection. Early fault detection provides the developers the necessary information to start fixing the problems caused by the changes as soon as possible while the changes made are still fresh in their minds. However, as the time and resources are highly utilized and therefore limited in CI environments, heavy analysis for an efficient TCP is not a viable option. The TCP techniques need to be both computationally lightweight and efficient in early fault detection to be applicable for CI.

White-box testing includes the use of data from within the system. Coverage-based TCP approaches are mostly dealing with white-box information, and are considered to be both effective as well as well researched and used in practice (Malishevsky et al., 2006). However, the approaches based on white-box information often require a computational overhead for analysing said data. Conversely, black-box testing focuses on using the outer data such as test case execution history, test case inputs and outputs. These data require less analysis making the TCP techniques based on them more lightweight and therefore more suitable for CI. However, the efficiency needs to be considered as well. The use of historical data for TCP is one of the most common approaches for TCP. The fault history of test cases is noted to be an efficient predictor for early fault detection rates.

The validity of this thesis' results depends on just a few authors as quite a few of the references are from the same authors. While the papers are fairly recent and cover almost exactly the topic of this thesis, the inclusion of these papers introduces a slight bias in the results. As the search queries were not narrowed down, the search results provided many studies to consider. A more systematic approach would have yielded more reproducible and considerable results. Software development, test automation and CI are very large areas and do not narrow down the results much even when combined into a single query. The inclusion of a query that combines both TCP and CI could have been beneficial as it could have provided studies with higher relevancy. A systematic approach would also have benefitted the classification of the TCP approaches as the current system introduces overlap and disambiguation between approaches for different techniques.

Future research could consider implementing some specific techniques from the wide array of different TCP approaches in CI environments to see how well they apply and whether it would be feasible to apply them in industrial contexts. Especially the hybrid, similarity-based and difference-based approaches context of CI do not seem to be widely represented in the literature as of yet.

The topic of test case prioritization remains relevant as different approaches are developed and existing algorithms are improved upon. There is no single solution for achieving optimal results as different projects' environments set different requirements, goals and constraints for the testing effort. In the modern world where autonomous solutions such as continuous integration and continuous deployment are necessary tools for increasing software development productivity and decreasing time-to-market, the software testing effort needs to keep up with the pace so that an efficient feedback loop can be established.

References

Bian Y., Li Z., Guo J. & Zhao R. (2018). Concrete hyperheuristic framework for test case prioritization. *Journal of Software: Evolution and Process*, Vol. 30 issue 11, 959-962.

Chen J., Zhu L., Chen T. Y., Towey D., Kuo F.-C., Huang R. & Guo Y. (2018). Test Case Prioritization for Object-Oriented Software:An Adaptive Random Sequence Approach Based on Clustering. *Journal of Systems and Software*, Vol. 135, 107-125.

Dahiya O. & Solanki K. (2018). A systematic literature study of regression test case prioritization approaches. *International Journal of Engineering & Technology*, Vol. 7 issue 4, 2184-2191.

Dahiya O., Solanki & K. (2019). Comprehensive Cognizance of Regression Test Case Prioritization Techniques. *International Journal of Emerging Trends in Engineering Research*, Vol. 7 No. 11, 638-646.

Dahiya O., Solanki K. & Dalal S. (2019). Comparative Analysis of Regression Test Case Prioritization Techniques. *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 8 No. 4, 1521-1531.

Dahiya O., Solanki K., Dalal S. & Dhankhar A. (2020). Regression Testing: Analysis of its Techniques for Test Effectiveness. *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 9 No. 1, 737-744.

Elbaum S., Rothermel G. & Penix J. (2014). Techniques for Improving Regression Testing in Continuous Integration Development Environments. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 235-245.

Haghighatkhah A., Mäntylä M., Oivo M. & Kuvaja P. (2018). Test Case Prioritization Using Test Similarities. In: *Kuhrmann M. et al. (eds) Product-Focused Software Process Improvement. PROFES 2018. Lecture Notes in Computer Science* (pp. 243-259). Springer.

Khanna M., Chaudhary A., Toofani A. & Pawar A. (2019). Performance Comparison of Multi-objective Algorithms for Test Case Prioritization During Web Application Testing. *Arabian Journal for Science and Engineering*, Vol. 44, 9599-9625.

Khatibsyarbini M., Isa M. A., Jawawi D. N. A., Hamed H. N. A. & Suffian M. D. M. (2019). Test Case Prioritization Using Firefly Algorithm for Software Testing. *IEEE Access*, Vol. 7, 132360-132373.

Liang J., Elbaum S. & Rothermel G. (2018). Redefining Prioritization: Continuous Prioritization for Continuous Integration. *Proceedings of the 40th International Conference on Software Engineering*, 688-698.

Mahali P., Prasad D. & Mohapatra P. (2018). Model based test case prioritization using UML behavioural diagrams and association rule mining. *International Journal of System Assurance Engineering and Management*, Vol. 9(5), 1063-1079.

Mahdieh M., Mirian-Hosseinabadi S.-H., Etemadi K., Nosrati A. & Jalali S. (2020). Incorporating fault-proneness estimations into coverage-based test case prioritization methods. *Information and Software Technology*, Vol. 121.

Malishevsky A. G., Ruthruff J. R., Rothermel G. & Elbaum S. (2006). *Cost-cognizant Test Case Prioritization* (Technical Report No.TR-UNL-CSE-2006-0004). Lincoln, Nebraska, U.S.A: University of Nebraska-Lincoln.

Marijan D., Gotlieb A. & Liaaen M. (2019). A learning algorithm for optimizing continuous integration development and testing practice. *Software: Practice and Experience*, Vol. 49 issue 2, 192-213.

Mårtensson T., Ståhl D. & Bosch J. (2019). Test activities in the continuous integration and delivery pipeline. *Journal of Software: Evolution and Process*, Vol 31 issue 4.

Paygude P., Joshi S. D., Bhattacharyya D. & Kim T. (2019). Comparative analysis of Test Case Prioritization Approaches in Regression Testing. *International Journal of Advanced Trends in Computer Science and Engineering*, Vol. 8 No. 4, 1260-1267.

Roongruangsuwan S. & Daengdej J. (2010). Test case prioritization techniques. *Journal of Theoretical and Applied Information Technology*, Vol. 18 issue 2, 45-60.

Rothermel G., Untch R. H., Chu C. & Harrold M. J. (2001). Prioritizing Test Cases For Regression Testing. *IEEE Transactions on Software Engineering*, Vol. 27 No. 10, 929-948.

Spieker H., Gotlieb A., Marijan D. & Mossige M. (2017). Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration. *Proceedings of the 26th International Symposium on Software Testing and Analysis*, 12-22.

Yu L., Alégroth E., Chatzipetrou P. & Gorschek T. (2020). Utilising CI environment for efficient and effective testing of NFRs. *Information and Software Technology*, Vol. 117.