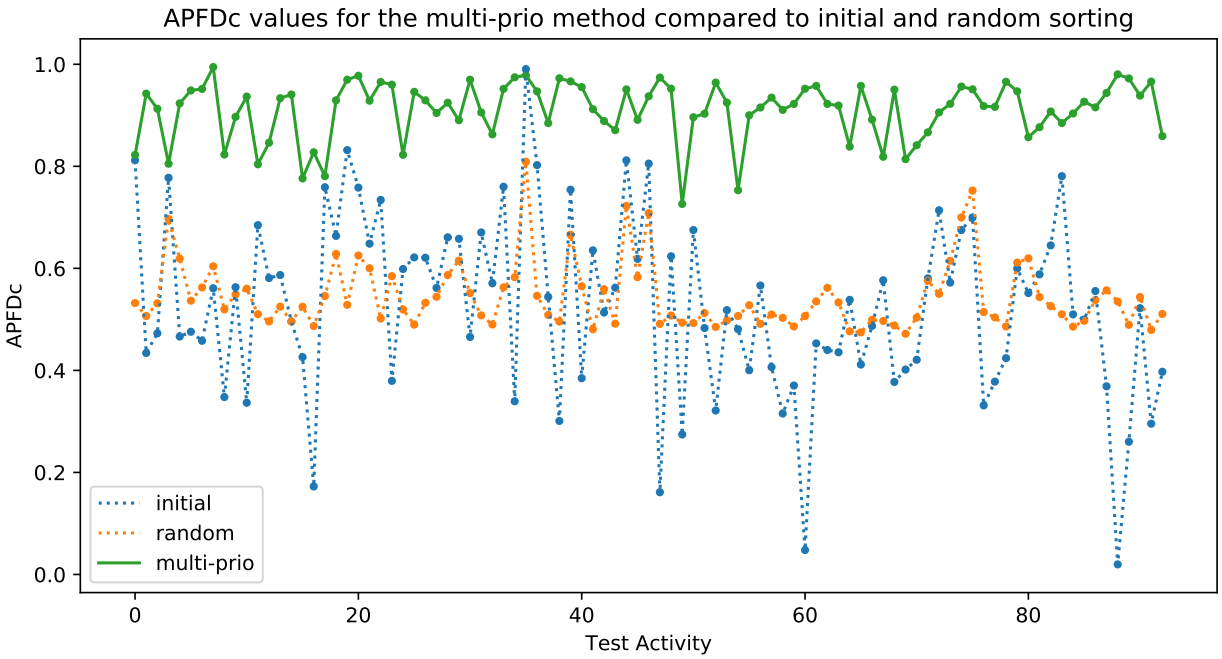

Test Case Prioritization for Automated Driving Systems

A Cost-Cognizant, Version-Specific, Multi-Objective Black-Box Approach

Master's thesis in Software Engineering

CHRISTOFFER KARLSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Test Case Prioritization for Automated Driving Systems

A Cost-Cognizant, Version-Specific, Multi-Objective Black-Box Approach

CHRISTOFFER KARLSSON

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Software Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Test Case Prioritization for Automated Driving Systems
A Cost-Cognizant, Version-Specific, Multi-Objective Black-Box Approach
CHRISTOFFER KARLSSON

Supervisor: Robert Feldt, Department of Computer Science and Engineering
Advisor: Tony Heimdahl, Volvo Car Corporation
Examiner: Christian Berger, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Software Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The graph comes from the experiment that was performed in this thesis. It shows APFDc values for 93 prioritized test activities for three different prioritization methods. The solid green line corresponds to the proposed MULTI-PRIO method, the dashed blue line corresponds to the INITIAL method, and the dashed orange line corresponds to the RANDOM method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

Typeset in LaTeX
Gothenburg, Sweden 2019

Test Case Prioritization for Automated Driving Systems
A Cost-cognizant, Version-specific, Multi-objective Black-box Approach
CHRISTOFFER KARLSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Testing and verification of self-driving cars play an important role in ensuring that the vehicles work as intended and that the strict safety requirements are met. Automotive companies are becoming increasingly agile and are adopting the principles of continuous integration, where short feedback loops are essential. Test case prioritization (TCP) techniques can help increase testing efficiency by promoting early fault detection, which enables defects to be found and fixed earlier. This thesis is done at Volvo Car Corporation (VCC), with the aim to investigate the applicability of TCP techniques within the complex autonomous drive verification domain, and to propose a TCP method that can be used in a continuous integration environment. One challenge is that access to historical test data at VCC is limited, as continuous integration has not been fully implemented yet. This is one of the reasons why VCC has developed a system called Verification Data Model and Generator (VeriDaG), which produces a simulated regression test history that is made to mimic the real system in a realistic way. In this way, VCC strives to have a prioritization strategy ready when continuous integration and test automation become fully implemented, as it may otherwise be too late. The thesis is conducted as a case study involving action research. A pre-study is conducted to form the basis of an experiment, in which different TCP techniques are compared. While the results of the thesis indicate that many existing TCP techniques are unsuitable within this domain, there still exist some types of techniques that seem applicable. A proposed black-box TCP method considering multiple factors is implemented and evaluated using the Average Percentage of Faults Detected per Cost ($APFD_C$) metric, which is used to measure the efficiency of TCP techniques. It is black-box in the sense that it does not require access to the source code of the system under test (SUT). Expert-review evaluations are also conducted at the end of the thesis, to get further insights into the usability of the proposed method. Combined evaluation of the proposed method suggests that it can help improve testing efficiency at VCC and likely also at similar companies.

Keywords: testing, prioritization, TCP, RTO, value-based, software engineering, automotive, autonomous drive, cost-cognizant, version-specific, multi-objective, black-box

Acknowledgements

I'd like to thank my academic supervisor Robert Feldt for providing me with guidance during the thesis and valuable insights into the test case prioritization research field. I'd also like to thank Christian Berger for his advice and constructive feedback as an examiner. Finally, a special thanks to Erik Sternerson, Piret Saar, and Tony Heimdahl at Volvo Car Corporation for their hospitality and active collaboration.

Christoffer Karlsson, Gothenburg, June 2019

Contents

List of Figures

List of Tables

Acronyms

AD autonomous drive. 40, 52
ADAS advanced driver-assistance systems. 40, 52
ANOVA analysis of variance. 20, 36
APFD Average Percentage of Faults Detected. 10, 11, 19–21, 28
APFD$_C$ Average Percentage of Faults Detected per Cost. v, xi–xiii, xv, xvi, I–III, V–VIII, X, 5, 7, 11, 19–21, 28, 31, 35–37, 45–51, 56–62
APTTF Average Percentage of Time to Failure. xv, xvi, 11, 35, 36, 45–47, 49, 50, 57, 59, 60, 62
ASDM Active Safety Domain Master. 40, 52

CMQ Current Model Quality. 43

DNN deep neural network. 23

ECU electronic control unit. 4, 20, 22, 39

HIL hardware-in-the-loop. 4, 20, 22, 39, 40, 52, 54

IEEE Institute of Electrical and Electronics Engineers. 28

JSON JavaScript Object Notation. xi, 12, 15, 31

KPI key performance indicator. 62

MIL model-in-the-loop. 22, 23

OCR optical character recognition. 12

PIL processor-in-the-loop. 22

RTO regression test optimization. 3, 6, 9, 10, 17, 21, 28
RTS regression test selection. 21

SAE Society of Automation Engineers. 7, 41
SIL software-in-the-loop. 22, 23
SPAS Simulation Platform Active Safety. 52
SUT system under test. v, 3–6, 9–12, 21, 30, 32, 35, 42, 45, 55, 56, 59–62

TAAS Test Avoidance and Automated Selection. 5, 28, 39, 40, 52, 54, 60
TCM test case minimization. 10, 28

TCP test case prioritization. v, 3–7, 10, 12, 17–21, 28, 31, 32, 41, 42, 44, 45, 55, 56, 60, 62, 63
TCS test case selection. 10, 19, 21, 28, 44, 45
TSI traffic sign information. 12, 42

UML Unified Modelling Language. 28

VBF Versatile Binary Format. 39, 42
VCC Volvo Car Corporation. v, 4–7, 9, 12, 13, 22, 25, 27–29, 32, 34, 37–39, 41, 44, 45, 54–56, 59–63
VeriDaG Verification Data Model and Generator. v, xi, xvi, 7, 12, 13, 15, 27, 28, 31, 34, 36–38, 40, 52–54, 59–61

1
Introduction

Automated driving systems (i.e. *self-driving cars*) are becoming increasingly popular and are likely to play a big part in our future daily lives. The rising levels of automation require more and more complex technology [4] that has to meet strict safety requirements [5], which introduces several challenges. Testing and verification of automated driving systems play an important role in ensuring that the vehicles work as intended and that the safety requirements are met [5].

Automated driving systems can be tested on several levels (model, unit, component, domain, system, vehicle, etc.), and in several different ways [4] (in-vehicle testing, replaying recorded sensor data, real systems in virtual environments, software components in simulations, signal stimulation, software unit tests, etc.). Identifying the most appropriate testing techniques and test cases in a certain scenario is a complex task with many parameters to take into account. In general, studies show that it is often difficult to compare testing methods against each other [6, 7]. Researchers also report a lack of guidelines about which test techniques to use for different testing objectives, and that this is a typical issue in the industry [8, 9].

The wide range of testing activities and increasing levels of complexity [10, 11] often result in a lengthy and expensive testing process, with long feedback loops. This is problematic as it slows down the development process, and the problem becomes even more apparent when testing is done frequently, such as when following a continuous integration practice, where frequent deliveries and short feedback loops are central concepts [12]. Regression test optimization (RTO) techniques aim to improve the efficiency of testing, usually by prioritization, selection, or minimization of test cases. One such technique is TCP, where a common goal is to improve the rate of fault detection. This is done by ordering test cases in a way such that faults are expected to be detected earlier [13], as to enable quicker feedback. This can help developers find and resolve bugs earlier [14], thus saving time and money.

Most currently existing TCP methods are often restricted in some sense. For instance, they might require access to the source code (e.g. [15–17]), only prioritize based on a single objective (e.g. [18]), discard differences in test costs and fault severities (e.g. [18, 19]), or ignore information about the changes that have been made in the SUT (e.g. [12, 19–21]). There are quite many existing studies which explore ways of overcoming these issues separately, but many fewer studies have been made that consider ways of overcoming combinations of them at the same time. This thesis considers multi-objective cost-cognizant version-specific black-box TCP — a combination which, to the best of our knowledge, never has been studied before — especially within the field of autonomous drive verification.

This study is done in collaboration with VCC. VCC is currently undergoing a transformation towards automated testing and continuous integration, and want to have a TCP strategy prepared before testing has become fully automated. One challenge is that access to historical test data is limited, as the continuous integration process has not been fully implemented and automated yet. The the aim of this thesis is to investigate how TCP applies to the complex context of autonomous drive verification, and to propose an existing method, a combination of methods, or a new method for prioritizing test cases in a way that can guide and control both manual verification activities, as well as automated continuous testing activities, for specific releases of the SUT.

1.1 Problem Statement

There are several aspects of complexity related to verification of autonomous drive systems. One aspect is the complex nature of the system itself. A full autonomous drive system can include over 100 individual hardware components (electronic control units (ECUs), sensors, etc.) [22], many of which have different software running on them. The full range of complexity needs to be considered when testing an autonomous drive system, from individual software components to the whole system. This means that applying just a single testing technique is not enough [23], but instead, a wide range of different verification methods is needed throughout the verification process, where each method has its own complexity. Both theoretical analyses and practical experiences have shown that combining different testing techniques is beneficial [23]. Another aspect of complexity is that it is common that a lot of the components in the system are developed by external suppliers [24], making verification more difficult [5].

The difference in the cost of using different verification methods is something that also adds to the overall complexity. VCC reports that stimulated component tests are costly in terms of hours needed to develop them, sensor playback data is costly to store, virtual environments are costly to run, hardware-in-the-loop (HIL) environments are costly to build and maintain, and in-vehicle testing is costly in terms of labor hours, as it cannot be reliably automated until the autonomous drive system itself is ready. Time is also a restricting factor, as the availability of test rigs is limited to some number of parallel tests. Approaching this issue by scaling the test environments to allow for more parallelism is an expensive approach. Rather, in order to overcome this issue, the test environments need to be utilized in a more efficient way.

Further adding to the complexity is that there for each verification method exists an array of test activities, test cases, scenarios, etc. to choose from, and that these usually are not equally valuable, for instance with respect to cost and time.

Each release of the SUT is also unique, with changes of different kinds and severities that might only affect certain areas of the SUT. For instance, a release could consist of a major new feature or a minor bug fix. Identifying suitable verification methods and test cases based on the release, is another aspect of complexity as different verification methods and test cases may be particularly valuable for a certain release.

For instance, consider a scenario where a car has already been thoroughly tested, and only its headlights are changed in a release. It would seem reasonable to be interested in prioritizing testing the headlight functionality, as that was what was changed, while there are likely many seemingly unrelated features that can be assigned a lower priority and be tested later.

Lastly, a final aspect of complexity is that the value of an individual activity, test case, or scenario cannot be determined within the immediate context of itself. Instead, a broader understanding of the domain and how different verification methods impact the overall confidence of the system is needed.

These different aspects of complexity make it difficult to make fact-based decisions on what verification methods, test activities, test cases, or scenarios in said verification methods actually provide the most value, especially for a specific release of the SUT.

Developing a suitable TCP technique that takes these complexities into account is the main challenge. It should preferably support prioritization of test cases from different testing methods, of different costs and durations, and take version-specific changes into account, as to maximize the efficiency of testing. And perhaps most importantly, the method has to be applicable within the autonomous drive domain, where testing is often done in a black-box fashion.

1.2 Purpose of the Study

The purpose of this study is to propose a test case prioritization method for verification of specific versions of an autonomous drive system.

Various companies working within the automotive and autonomous drive domain either have, or are planning to build, an infrastructure that provides the data potentially required for the previously mentioned TCP method to work [25], as well as solutions that could use the output of an implementation of said method to control the verification scope of an autonomous drive system release (e.g. Test Avoidance and Automated Selection (TAAS)) in the case of VCC).

1.3 Research Questions

Three main research questions were defined for the scope of the thesis. As the thesis is conducted as a case study at VCC, the research questions primarily target the case company. However, the intention is also that the findings from this study can be generalized to other companies within the same domain.

RQ1: To what degree are existing test case prioritization methods applicable to the autonomous drive domain?

RQ2: What existing method, or combination of methods, gives a sufficiently good[1]

[1]Determining what is considered a "sufficiently good" test case prioritization can be done quantitatively by comparison to a baseline. For instance, by evaluating the prioritized test case order against no order and random order, using some metric, such as $APFD_C$. It can also be based on qualitative measures, such as through an expert-review evaluation.

test case prioritization?

RQ3[2]: What new method gives a sufficiently good test case prioritization, and what data does it require?

1.4 Delimitations

Trying to understand the details of how each verification method works is not part of the scope of this study. Instead, the focus will be on common properties of the verification methods and their test cases, and what they are able to achieve. For instance, each verification method and test case requires some time, incurs some cost, and produces some results. Aspects such as these will be of interest during the course of this study.

An assumption about the use case of the proposed prioritization method, as specified by VCC, is that it should be applied after basic static testing of the SUT has already been performed (i.e. the testing that always needs to performed in order to prove that the fundamental functional and safety requirements of the car have been met). This method would then be applied to prioritize the remaining testing activities that can be executed after basic safety-conformance testing has been performed.

1.5 Contributions

Automated driving systems are becoming increasingly popular, with several systems being developed globally at many prominent automotive companies. Many of these systems are still in a relatively early phase of their development. The shift of operation and decision-making from humans to machines requires a lot of trust, which is established through strict safety requirements. These safety requirements in combination with the need for increased speed of software development, make continuous integration and automated verification an essential part of development. Increasing levels of complexity [4] and an ever-growing number of test cases, in combination with more and more frequent deliveries, make TCP an important step in increasing the efficiency of the testing process.

The majority of the studies that have been done in the field of TCP has been done for white-box testing [18, 20, 21], i.e. for systems where access to the source code is available. This project, on the other hand, considers a domain where testing is almost always done in a black-box manner [4, 26]. Many TCP techniques are also single objective, and they seldom take aspects of cost into account [27]. Only a few authors have explored the area of multi-objective RTO [28], and authors such as Harman [29] argues that it is an important area to investigate. This thesis focuses on multi-objective cost-cognizant black-box prioritization. The technique proposed in this thesis will also be version-specific, meaning that it will prioritize the testing activities according to the specific release of the system, whereas many other TCP

[2]If RQ2 would indicate that an existing method or combination of methods can be used to give a sufficiently good test case prioritization, then less focus would be spent on RQ3.

techniques focus on general TCP, where arguably valuable information about the release is ignored.

Many of the previous studies have also been done on relatively small systems, while this study is done in an industrial setting, within a complex autonomous drive verification domain, where prioritization can be done within and across multiple testing levels. In general, few TCP studies have been done within the automotive domain [4], and the industry lacks knowledge on how to test more efficiently. This problem will also likely become worse over time, as deliveries are getting more and more frequent and the complexity of testing increases as the complexity of the driving systems increases (e.g. increasing SAE levels require more and more advanced testing).

The combination of multi-objective cost-cognizant version-specific black-box prioritization, as studied in this thesis has, to the best of our knowledge, never been studied before — especially within the field of autonomous drive verification. The outcome of this study provides strategies for verification of autonomous drive systems (and to some extent, likely also other systems of similar complex nature), that is of interest both for the industry (e.g. VCC) as well as the research community.

1.6 Methods

This study is performed as a case study involving action research at VCC. It starts with an exploratory phase, which can be seen as a pre-study through literature review, analysis of work artifacts, and focus group interviews. Then, based on the findings from the pre-study, appropriate TCP methods are identified and implemented, and later evaluated using the APFD_C metric. VCC is currently undergoing a transformation towards continuous integration and automation of testing activities. Access to historical test data is limited, as the continuous integration process has not been fully implemented and automated yet. For this reason, VCC has developed a system called VeriDaG (see section 2.5), that produces a simulated regression test history that is made to model reality accurately. The proposed method is quantitatively evaluated on this regression test history, and finally qualitatively evaluated using expert-review evaluations, to get a broader understanding of the method's performance and usability.

1.7 Thesis Outline

Chapter 2 introduces relevant concepts and theory related to the research topic.

Chapter 3 presents related work and positions the thesis in relation to current research.

Chapter 4 describes the methodology and methods used when conducting the study. In other words, a description of the steps that were done in order to answer the research questions.

Chapter 5 presents the results found from the steps involved in doing the project, i.e after following the process described in chapter 4.

Chapter 6 discusses the implications of the results found in 5, and connects them to the initial research questions presented in chapter 1.

Chapter 7 concludes the findings and implications of the thesis project, in a short and concise manner.

2
Background

2.1 Case Company

This thesis was done in collaboration with VCC. It was conducted at the Continuous Integration division of the Active Safety & Autonomous Drive department.

VCC is working within the field of autonomous drive (i.e. self-driving cars), and are moving towards an agile way of working. They are currently undergoing a transformation where processes such as continuous integration are being introduced to automate the testing and verification processes of their cars.

2.2 Black-Box Testing

Black-box testing is a method of testing that does not consider the internal functionality of the SUT, as opposed to white-box testing [30]. Instead, the SUT is treated as a black-box, where the internal functionality is unknown to the tester. Black-box tests are often based on specifications [26], where the tested functionality receives some particular input and is expected to produce a certain output [30]. Exactly *how* it produces the output is not of interest within black-box testing. This type of testing is often used when there is no access to the source code of the SUT.

2.3 Continuous Integration

Continuous integration is a practice that encourages frequent integration of software. A central aspect of continuous integration is automatically building and testing the system, as soon as changes are checked into the version control repository [31]. This increases the chance of discovering defects as soon as they are integrated, instead of during late-cycle testing. Another important concept within continuous integration is continuous feedback [31], where the aim is to achieve short feedback loops [12] by reducing the time from when a defect is introduced to when it is discovered and fixed [31].

2.4 Regression Test Optimization (RTO)

There exist different approaches for optimizing regression testing. So called RTO techniques aim to improve the efficiency of testing using techniques such as by

prioritization, selection, or minimization of test suites.

Test case selection (TCS) techniques strive to select a subset of the total test suite, to reduce the cost of testing [14]. The aim is to select a subset that effectively tests the modified version of the SUT [14]. This is often achieved by considering modified lines of code, and code coverage of test cases, such that test cases that cover the modifications are selected.

Test case minimization (TCM) techniques also select a subset of the total test suite, but with the intention of removing redundant test cases, such that the coverages of the total test suite and the minimized test suite are equal, according to some criterion [14]. For instance, a test case might be considered redundant if the lines of code it covers already have been covered by some other test case in the minimized test suite.

TCP techniques provide a different approach to RTO. These techniques strive to order test cases in a test suite according to some criterion, such that high priority test cases are executed before lower priority test cases [14]. The formal definition of the *test case prioritization problem* can be seen in section 2.4.1. A common goal is to prioritize for early fault-detection, so that test cases that are prone to detect faults are executed earlier. TCP techniques belong to the family of safe RTO techniques, as they do not discard any test cases, as opposed to TCS and TCM techniques. This ensures that the optimized test suite still contains all test cases, which can be important in safety-critical systems, such as autonomous drive systems.

2.4.1 The Test Case Prioritization Problem

The test case prioritization problem is, according to [14], formally defined as:

Definition 1. *The Test Case Prioritization Problem*

Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.

Problem: Find $T' \in PT$ such that $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$.

In the definition above, PT corresponds to the set of all possible prioritizations (permutations) of the test suite T. f corresponds to a function that produces an award value for any such permutation, which evaluates how good the prioritized test sequence is. The goal is to find some prioritized test sequence T', such that its award value is equal to or higher than the award value of all other prioritized test sequences. The definition assumes, for simplicity and without loss of generality, that higher award values are preferable to lower ones.

2.4.2 Average Percentage of Faults Detected (APFD)

Average Percentage of Faults Detected (APFD) (see equation 2.1) is the most commonly used metric to measure the performance of prioritization algorithms [32], in terms of rate of fault detection. APFD is, however, limited in the sense that it treats all test cases as if they were equally costly, and all faults as if they had the same severity. In practice, the costs and severities can, however, vary greatly, which can

make APFD rather misleading [33, 34]. Variants of APFD have been proposed to counter these limitations (see e.g. section 2.4.3).

$$APFD = 1 - \frac{TF_1 + TF_2 + \cdots + TF_m}{nm} + \frac{1}{2n} \tag{2.1}$$

In the formula presented above, n represents the number of test cases in a test suite T, m denotes the number of faults revealed by T, and TF_i corresponds to the position of the first test in a permutation T' of T that exposes fault i.

2.4.3 Average Percentage of Faults Detected per Cost (APFDc)

Following the limitations and criticism of APFD, Malishevsky et al. [33] defined a new cost-cognizant metric, APFD_C (see equation 2.2), similar to APFD, which takes differences in test costs and fault severities into account, inspired by principles from Value-Based Software Engineering [34]. APFD_C enables evaluation of prioritization methods that belong to what is often referred to as *cost-cognizant test case prioritization* [33].

$$APFD_C = \frac{\sum_{i=1}^{m}(f_i \times (\sum_{j=TF_i}^{n} t_j - \frac{1}{2}t_{TF_i}))}{\sum_{j=1}^{n} t_j \times \sum_{i=1}^{m} f_i} \tag{2.2}$$

In the formula presented above, n represents the number of test cases in a test suite T, m denotes the number of faults revealed by T, TF_i corresponds to the position of the first test in a permutation T' of T that exposes fault i, t_j is the cost (e.g. duration) of test case j, and f_i represents the severity of fault i.

2.4.4 Average Percentage of Time to Failure

APTTF is a novel cost-cognizant metric that is proposed to be used to measure the efficiency of test suites in continuous integration environments. The metric is very similar to the APFD_C metric, but instead of considering detected faults, it is considering test case failures. This removes the need of having to be aware of specific faults in the SUT, which is often achieved using fault seeding. The idea of the APTTF metric is to find how long time it takes, on average, for test cases in the test suite to result in a failure. The value corresponds to the fraction of the total test suite execution time. The value ranges between 0 and 1, and an APTTF value closer to zero is preferable, as it implies that test cases which result in a failure are executed early on average, which is desired within continuous integration, following the principles of early feedback. For instance, $APTTF = 0.21$ means that test case failures, on average, occur after 21% of the total test time has passed.

$$APTTF = \frac{\sum_{i=1}^{m} \sum_{j=1}^{TF_i} t_j}{m \times \sum_{j=1}^{n} t_j} \tag{2.3}$$

In the formula presented above, n represents the number of test cases in a test suite T, m denotes the number of failed test cases in T, TF_i corresponds to the position of the test in a permutation T' of T that results in failure i, and t_j is the duration of test case j.

2.5 Verification Data Model and Generator

The Verification Data Model and Generator (VeriDaG) is a solution developed by VCC for generating large amounts of realistic data that represent both the current state of data related to continuous integration and testing, as well as a few desired upcoming states. The simulated data is generated to mimic the data of the real system, but in much larger quantities, and in a way that makes evaluation of different concepts at VCC much faster and significantly less costly. The model is used for a number of purposes, such as defining the entities that provide input to and output from agile development and continuous integration, as well as defining requirements for the remaining development of the continuous integration flow at VCC. VeriDaG is implemented in C# and produces simulated data in JSON format about things such as deliveries and historical test results (similar to the tool *Simultron*, developed and used by Ekelund [35]), for different scenarios (see an example in figure 2.1). This also makes VeriDaG a relevant tool to use when investigating the possibilities of TCP techniques. An accompanying data model diagram together with the generated JSON output, were used to gain insights into VeriDaG, and the data available within continuous integration at VCC. Among other things, VeriDaG shows us what data can be extracted from a delivery — which can be useful to know when doing version-specific TCP, and what data is stored after a test has been run — which can be useful when doing history-based TCP. Overall, VeriDaG provides useful information about the possibilities and limitations of the data that is available for use by the proposed prioritization algorithm.

VeriDaG can also output so called oracle data, which reveals some underlying information about how the scenarios in the simulation were constructed. This can be useful when users of VeriDaG want to verify their assumptions about the generated data. The oracle data can, for instance, tell us all the possible faults a test case can find. The oracle data is only used for evaluation and debugging purposes, and is never used when constructing models, as this data is generally not available outside the simulated setting.

While creating a perfect simulation is hard, VeriDaG tries to mimic reality in an accurate way. Some of the concepts that are taken into account are presented below:

Testing is performed on different integration levels (e.g. unit, component, domain, complete car) and using different methods (e.g. simulation, stimulation, sensor replay). Testing methods have different costs and execution times, and some are used more than others. These concepts are directly borrowed from real-world verification configurations used to verify autonomous drive systems.

Each modelled software component has a set of feature areas (e.g. optical character recognition (OCR), traffic sign information (TSI)). A release consists of one or more changes to a selection of these feature areas. A software component feature area is typically best tested at a certain integration level and by some specific testing method.

Each modification in the SUT has a chance of generating faults. Each modification

affects one or more feature areas, has a type (e.g. bug fix, changed feature), and a severity (e.g. trivial, major).

Each test case has a duration and a stability, and is connected to a specific integration level and test method. A test case is generally focused on finding problems in a specific feature area. A test case can result in a success, a failure, or an error. A change in the feature area that is connected to the test case has a higher chance of affecting the result of the test case. Test cases are valuable in different ways (e.g. some take a long time to run, but manage to find hard-to-discover faults).

Faults can be of different severities (e.g. normal, catastrophical) and have varying discoverability (i.e. some faults are intermittent). Faults can be nested behind other faults. It typically takes a number of versions until a fault is fixed. Fixing a fault can introduce new problems.

VeriDaG is based on many years of experience of testing highly complex software systems within the automotive and telecom industry. VCC also reports that the data model has been inspected by several people with extensive experience in autonomous drive verification, and that no major flaws have been detected.

The key concepts and data produced by VeriDaG, that are likely to be of interest in this thesis, are summarized below:

Each Baseline has different versions (BaselineVersion), where each version is made up of Releases of Releasables. These are general concepts, which can be thought of as: every new software version (BaselineVersion) consists of a set of specific versions (Release) of components (Releasable). It also contains information about the test activities, test suites, and test cases, that were run to verify the particular BaselineVersion.

Every Release contains a reference to the Releasable and the areas of it that were changed in the version (ModifiedFeatureArea), as well as the type and severity of the changes. The ModifiedFeatureArea is a part of the desired data, and has not yet been fully implemented in the currently existing test automation flow, but is available for theoretical scenarios in VeriDaG (some meta-data about the changes are however available today, albeit not in a standardized or machine-readable format).

Every TestCase contains a test result, with a verdict and a duration. Similarly, every TestActivity and TestSuite each contains an aggregated test result. It is also possible to see which faults a test case discovered, if any. FaultsDiscovered belongs to the oracle part of VeriDaG, meaning that it should not be used to *construct* prioritization algorithms, but may be used to *evaluate* them. ReleasableFocuses describes the purpose of a test case, i.e. what feature of what part of the system it is supposed to test. ReleasableFocuses is a part of the desired data and has not yet been implemented in reality.

In conclusion, the model contains general concepts which can be found in many software testing systems. Essentially, it has information about different versions of the software, with high-level information about the changes, and the test results obtained from testing the different versions. It also contains some data that might

be implemented in the future testing flow, such as more detailed and standardized information about the modifications of the software version, as well as the purpose of different test cases. Using this information in the prioritization algorithm could be interesting to get an idea of the possibilities of prioritization in the future. However, it is probably a good idea to construct a method that does not rely upon this data, as it is not available in the real test automation flow at the moment.

```
{
  "Baselines": [{
    "Id": "Baseline:2",
    "Versions": [{
      "Version": 67,
      "Releases": [{
        "Version": 20,
        "Releasable": "Releasable:10",
        "ModifiedFeatureAreas": [{
          "FeatureArea": "FeatureArea:15",
          "ChangeType": "BugFix",
          "Severity": "Major"
        }]
      }],
      "TestActivities": [{
        "Id": "TestActivity:1",
        "Method": "Simulation",
        "TestSuites": [{
          "Id": "TestSuite:1",
          "TestCases": [{
            "Id": "TestCase:3",
            "ReleasableFocuses": [{
              "Releasable": "Releasable:10",
              "FeatureAreas": "FeatureArea:15"
            }],
            "TestResult": {
              "Duration": 225038.1,
              "Verdict": "Failure",
              "FaultsDiscovered": ["Fault:1", "Fault:2"]
            }
          }]
        }]
      }]
    }]
  }]
}
```


Figure 2.1: Example JSON data showing the structure of the data produced by VeriDaG. The data has been simplified to only highlight the most relevant data (some data is omitted, for instance). Note that ModifiedFeatureAreas and ReleasableFocuses are a part of the desired data, and that FaultsDiscovered is a part of the oracle data.

3
Related Work

3.1 Regression Test Optimization (RTO)

Harman [29] talks about multi-objective RTO and different value objectives (to be maximized) and cost objectives (to be minimized) that are likely to be useful within that area. Harman also mentions some constraints that may need to be taken into consideration by the RTO algorithms. Many TCP techniques are restricted in the sense that they only do prioritization based on a single objective, such as code coverage. But in reality, many objectives are often of interest. Harman discusses why single objective optimization is often not sufficient and argues that a multi-objective approach is long overdue. Some studies, albeit a lot fewer, have been made on the subject of multi-objective TCP (e.g. Wang et al. [15]). Also, Harman mentions that very little research has been done on industrial software systems, and on RTO methods considering non-code-based coverage.

Cost objectives identified by Harman [29]:

- Execution Time
- Data Access Costs
- Third Party Costs
- Technical Resources Costs
- Setup Costs
- Simulation Costs

Value objectives identified by Harman [29]:

- Code-based Coverage
- Non-code-based Coverage (Harman mentions that these have been significantly less studied in RTO literature)
- Fault Model Sensitive
- Fault History Sensitive
- Human Sensitive
- Business Sensitive

3.2 Test Case Prioritization (TCP)

Roongruang et al. [3] identify four classification categories for TCP techniques, into which they classify previously done research into.

Classifications of TCP techniques, as identified by Roongruang et al. [3]:

- Customer Requirement-based Techniques
- Coverage-based Techniques
- Cost Effective-based Techniques
- Chronographic History-based Techniques

They also propose a new TCP process, *2R-2S-3R*, consisting of the following stages in chronological order: select prioritization technique (i.e. from the four different classifications), specify coverage or factors, re-assign weight value, re-calculate priority value, and lastly, re-order test cases. They also propose a list of practical weight factors to be used within TCP, divided into four main categories: *Cost*, *Time*, *Defect*, and *Complex*. Finally, they propose a new TCP method called *Multi-Prioritization*, which takes these practical weight factors into consideration. They use the 100-point method [36] to assign weights to the prioritization factors.

Four main categories of TCP factors, as identified by Roongruang et al. [3]:

- Cost
- Time
- Defect
- Complex

Wang et al. [15] talk about multi-objective TCP. They present a technique integrating five typical test objectives into a single objective, by combining them using a weighted sum. Their approach requires access to the source code. It's combining methods based on code coverage, fault-exposing-potential, requirement property relevance, history, and time spending into a single objective. The value is calculated by normalizing the individual values and summing them using weights for each factor. The weights used in the weighted sum are assigned to be equal in their experiment, but they can be changed depending on the desired goal of utilizing the prioritization method (e.g. to prioritize test cases of shorter duration).

Prioritization factors used by Wang et al. [15]:

- Coverage (statements in code)
- Fault-exposing-potential (based on mutation analysis)
- Requirement Property Relevance (number of requirements covered)
- History (historical execution data, such as historical fault detection)
- Time Spending (execution time)

Most existing TCP techniques require access to the source code [28]. For instance, a common objective within TCP is to maximize the code coverage of the test cases ("*order the test cases according to their code coverage in descending order*"). There are, however, TCP techniques that do not require access to the source code. For instance, some of these techniques have value objectives that are based on non-code-based coverage, such as requirements coverage (e.g. [20]). Many of the techniques also consider the failure-revealing history of the test cases as an objective to maximize. Fazlalizadeh et al. [19] propose a history-based multi-objective method for TCP. Their method consists of an equation, considering the historical performance in fault detection during the lifetime of the test case, the previously assigned priority of the test case to smooth out sudden major changes in the priority, as well as the

time that has passed since the test case was last run, to ensure that it eventually gets run.

Prioritization factors used by Fazlalizadeh et al. [19]:

- Historical fault detection performance
- Time since last execution
- Last assigned priority

Engström et al. [2] extend the work of Fazlalizadeh et al. by adding two more factors to the original prioritization method, namely a static priority for each test case, as well as the creation date for each test case. In this way, new test cases can be assigned a higher priority. They refer to the method by Fazlalizadeh et al. as the *Faz approach* and their extended version as the *ExtFaz approach*.

Lachmann et al. [20] talk about system-level TCP. They use a supervised machine learning approach, which considers natural language test case descriptions (describing the different steps to be executed by the tester and the expected results) as well as black-box meta-data (i.e. data that requires no access to the source code) (*Requirements Coverage*, *Failure Count*, *Failure Age*, *Failure Priority*, *Test Execution Cost*). They perform natural language processing of the test case descriptions to improve their prioritization. They talk about component-based systems, the common usage of external developers, and the connection to the automotive industry. They also evaluate their method against systems with connections to the automotive industry. They evaluate it based on early fault detection capabilities using APFD.

Prioritization factors used by Lachmann et al. [20]:

- Test Case Description
- Requirements Coverage
- Failure Count
- Failure Age
- Failure Priority
- Test Execution Cost

Malishevsky et al. [33] adapt four already existing code-coverage-based TCP methods to take test costs and fault severities into account (i.e. to be cost-cognizant). They then evaluate the methods in a case study, using the $APFD_C$ metric, which they earlier presented in the same report. They also mention different strategies for estimating fault severities, based on module criticality and test criticality.

Marijan et al. [12] present another cost-cognizant TCP technique for continuous integration called *ROCKET*, based on historical execution times and fault-detecting abilities of the test cases. They evaluate their method using $APFD_C$. Their method can be seen as a hybrid approach, as it also incorporates an upper time limit for the total test suite, which closely resembles the constraints seen in TCS techniques.

Qu et al. [21] propose a TCP method for black-box testing, based on fault types. They define a relation matrix R, where test cases are grouped based on the types of faults they detect, based on historical data. Later during run-time, the idea is that when a fault of a certain type has been found, other test cases that reveal the same type of fault are given a higher priority. In their discussion about future work, they mention that one could try incorporating other information than fault types

when constructing the relation matrix, such as constructing R by grouping test cases based on their test objectives instead of fault types.

Elbaum et al. [37] show that version-specific TCP can improve the rate of fault detection significantly. They present different difference-based techniques where tests are prioritized according to their coverage of changed lines of code. Similarly, Mukherjee et al. [32] report that researches have indicated that TCP techniques that are based on program change information outperform static or dynamic techniques. Elbaum et al. perform an analysis of variance (ANOVA) test to check for significant differences between the mean APFD values of the compared techniques. Khatibsyarbini et al. [27] report that several researches have used ANOVA tests to evaluate TCP techniques. Elbaum et al. also compare fine-granularity techniques (e.g. source code statement level) to coarse-granularity techniques (e.g. function level), and find that fine-granularity techniques generally outperform techniques that operate on a coarser granularity, but only by a relatively small margin. This showed that the lower precision of the coarse-granularity techniques did not significantly reduce their fault-detection abilities.

Srivastava et al. [38] explore TCP in large scale software development environments. They develop a system called Echelon, which uses binary matching to detect changes between two binary versions of the software, and combines this information with historical coverage of test cases as a basis for prioritization. Their method and results are interesting, but Echelon has been criticized by authors such as Zheng et al. [39] as it is a large proprietary Microsoft internal product that requires an advanced infrastructure and an underlying binary code modification engine.

Dadwal et al. [4] have performed a systematic literature review to evaluate publications related to TCP within the automotive domain. They report that literature about prioritization within the automotive domain is scarce. They mention that several studies have shown that automotive systems are becoming increasingly complex and that external suppliers are one of the reasons that almost all testing within automotive system testing is black-box. Different types of testing that are often used within automotive testing are also presented, such as testing ECUs on HIL rigs and running scenario-based simulations. Dadwal et al. also present an overview of some recurring problems that studies within the field are targeting, such as time consumption, cost, and complexity.

Malz et al. [25] report that test managers from different companies — especially within the automotive domain, mention that TCP has to be performed at their companies, but that much of prioritization is currently done manually, based on the intuition of the test managers. Malz et al. also report that the subject companies often already have access to relevant tools and information sources for TCP, such as test management tools, fault management tools, and change management tools, but that these are not utilized in an efficient way. The overwhelming amount of available information in combination with a restricted time budget leads to an inefficient prioritization, as a result of the intuition-based prioritization.

Multiple authors (e.g. Mukherjee et al. [32] and Khatibsyarbini et al. [27]) report that the most commonly used TCP evaluation metric is APFD or some variant of it, such as the cost-cognizant metric APFD_C, which also takes cost (e.g. time) and

fault severity into account. Khatibsyarbini et al. [27] report that $APFD_C$ is the most commonly used metric to measure the effectiveness of history and fault-based approaches. The APFD metric is arguably an insufficient metric in the context of this study, as the aspect of difference in cost (e.g. time) between different testing activities is complete ignored by it, as it treats all test cases as if they are equally costly, which is far from true in the context of autonomous drive verification.

3.3 Test Case Selection (TCS)

This thesis does not focus on TCS, but studying TCS techniques and principles might still provide insights into the problem at hand. For instance, TCS techniques often take changes in the SUT into consideration, similar to how version-specific TCP techniques work. Intuitively, test cases identified as most valuable by a TCS technique would likely also be assigned a high priority by a version-specific TCP technique, as they have some things in common. Hence, some relevant studies within TCS were also investigated.

Biswas et al. [40] perform a survey regarding regression test selection (RTS) techniques. They present a number of different test selection techniques and categorize them according to certain properties. Similar to TCP, many TCS techniques also require access to the source code (e.g. data-flow analysis-based techniques, slicing-based techniques, module level firewall-based techniques, differencing-based techniques, control flow analysis-based techniques, design model-based techniques). There are, however, some TCS techniques that do not require access to the source code. Biswas et al. mention specification-based techniques — which utilize a mapping between requirements and test cases, as well as meta-content-based techniques — which utilize meta-content attached to each release by the suppliers, with relevant information about the changes that have been made, for cases when there is no access to the actual source code for analysis of the changes. Techniques like this require good communication between suppliers and testers.

Ekelund [35] proposes a method for TCS based on regression test history. A tool called the Difference Engine is developed to correlate code and test cases at a package level, and then recommends a subset of test cases that have a strong correlation to the packages that were changed in the latest release of the SUT. Ekelund also develops and uses a tool called the Simulatron, which is used to create a simulated regression test history, for evaluation of the method.

3.4 Automotive & Autonomous Vehicle Testing

Previous work done within the field of automotive and autonomous vehicle testing (not necessarily related to RTO or TCP) can also provide valuable insights, as this thesis is about TCP within the context of autonomous drive verification, which is a part of the automotive domain.

Sundmark et al. [5] conduct an exploratory industrial case study at the Swedish automotive company Scania and describe how testing is performed within the automotive

domain. They also observe common challenges that arise within this context, such as problems related to the complex nature of the automotive systems themselves, the strict requirements that are put on safety and reliability, and complications of testing due to the tendency of sub-contracting the development of system components. The work of Sundmark et al. provides insightful information that is closely related to the context of this study. A lot of the challenges they identify in their case study at Scania are also challenges that are present at VCC (e.g. verification complications due to the use of subcontractors).

Lachmann et al. [26] talk about suppliers and testers, and their typical relation within automotive testing. They talk about different challenges (as well as suggested approaches to solve them) that are related to automotive software testing, such as black-box test scenarios, specifications based on natural language, and testing activities that are highly manual. They suggest having a unified test case specification to simplify analysis of test cases and to make them comparable. This can allow analysis of aspects such as if different components or features are covered more than others or if there are redundant test cases.

Bringmann et al. [41] describe the characteristics of model-based testing of automotive systems. They mention that model-based development is becoming increasingly popular within the automotive industry, with software components being modeled using tools such as MATLAB & Simulink, instead of being handwritten by developers in C or Assembler code. These model-based methods enable high-level models to be generated and used for simulations at early stages of the development process, which often allows the system to be tested before expensive integration has been done. They also describe the different integration levels that are often found within model-based automotive testing, starting from model-in-the-loop (MIL), where the embedded system and its environment are entirely simulated without any hardware, through software-in-the-loop (SIL), processor-in-the-loop (PIL), HIL, and test rigs, all the way to testing on the complete car level, where the final ECU is tested in a real car. MIL, SIL, and PIL do not require tests to be run in real-time. Bringmann et al. also also talk about the importance of test automation, as a result of high quality assurance standards and an iterative development process, that often leads to a considerable number of interim releases of the system, where each version needs to be tested.

Koopman et al. [42] explore and identify major challenges in autonomous vehicle testing and validation. They describe problems related testing and validation that arise due to the often non-deterministic nature of many technologies used in autonomous vehicles. They argue that it can be hard to test specific edge-case situations, but also that it is hard to know whether the test results are correct or not, as defining assertions for randomized behavior is a complex task. Multiple test cases testing the same scenario might often be required to build enough confidence in the system. Koopman et al. also discuss the frequent use of machine learning systems in autonomous vehicles, and the challenges that arise when these systems are to be validated. From a safety perspective, they argue that it is hard to prove that a machine learning model has not been overfitted on the training data. Another problem related to validating machine learning models is that they are often hard for humans

to understand. A convolutional neural network almost becomes a black-box, where the internal structure and learned decision rules may be hard to comprehend. Finally, they talk about the combinatorial explosion of scenarios, edge-cases, and other factors, and argue that these make traditional testing unfeasible. They mean that the use of fault injection and robustness testing can play a useful role in overcoming these issues, and mention that recent fault injection techniques have been successful in finding defects in autonomous vehicles.

Tian et al. [43] explore the area of deep neural networks (DNNs) and how autonomous cars that are built using the technology can be tested. They point out that it is hard to build safety-critical systems that are robust, by only using manual test cases. Tian et al. mention that recent advancements within machine learning and deep neural networks have lead to several major car manufacturers utilizing DNNs in their autonomous cars, such as Tesla, Ford, BMW, and Waymo/Google. The DNNs often take input from several sensors, such as cameras and LIDARs, to understand the car's surroundings. They point out that DNN-based software and traditional software are fundamentally different, in the sense that traditional software is created by a human writing code, to define the software logic, while a DNN automatically learns its logic by observing a large amount of data, without much human interaction. Following this argument, they also mention that traditional objectives within systematic software testing, such as to maximize code or branch coverage, often are not applicable within DNN-based software, as the logic is not expressed using control flow statements, but instead using concepts such as weights between neurons and nonlinear activation functions. In their paper, they design and implement a systematic testing tool called DeepTest, that can help detect erroneous behavior in autonomous vehicles that are built using DNNs. Instead of maximizing code-coverage, DeepTest tries to find inputs that maximize neuron coverage in the tested DNN, which also leads to a higher output diversity. The proposed system was tested on top-performing DNN from the Udacity self-driving car challenge, and results show that DeepTest managed to find thousands of erroneous behaviors, out of which many lead to potentially fatal car crashes.

Rosique et al. [44] perform a systematic literature review on the use of simulation in autonomous vehicle development. They point out many reasons why simulations are useful and widely used within this domain. For instance, regulations often restrict the possibilities of testing autonomous vehicles in real life, and field tests are often very expensive and time-consuming. They mention that the V-model is the most widely used development process within the automotive industry, and similar to Bringmann et al. [41], they also talk about how simulation can be used at different abstraction levels (e.g. MIL, SIL, etc.). They compare different tools intended for different types of simulation, such as MATLAB & Simulink, Unity 3D, Gazebo, and Carcraft/Waymo. These different tools can be used for a wide variety of simulation testing, from basic MIL tests in MATLAB & Simulink, to advanced 3D simulations in game and physics engines. They discuss the implications of high fidelity autonomous driving, in the sense that it requires very extensive testing of a wide range of scenarios, but that achieving this through physical tests is often very complicated, expensive, time-consuming, and in some cases even impossible, due to factors such as legal regulations. They conclude that simulations can manage to

fill this gap. They also mention that many simulation methods together can cover the whole development process, and the general possibilities of simulation within autonomous vehicle development are very broad.

4
Methodology

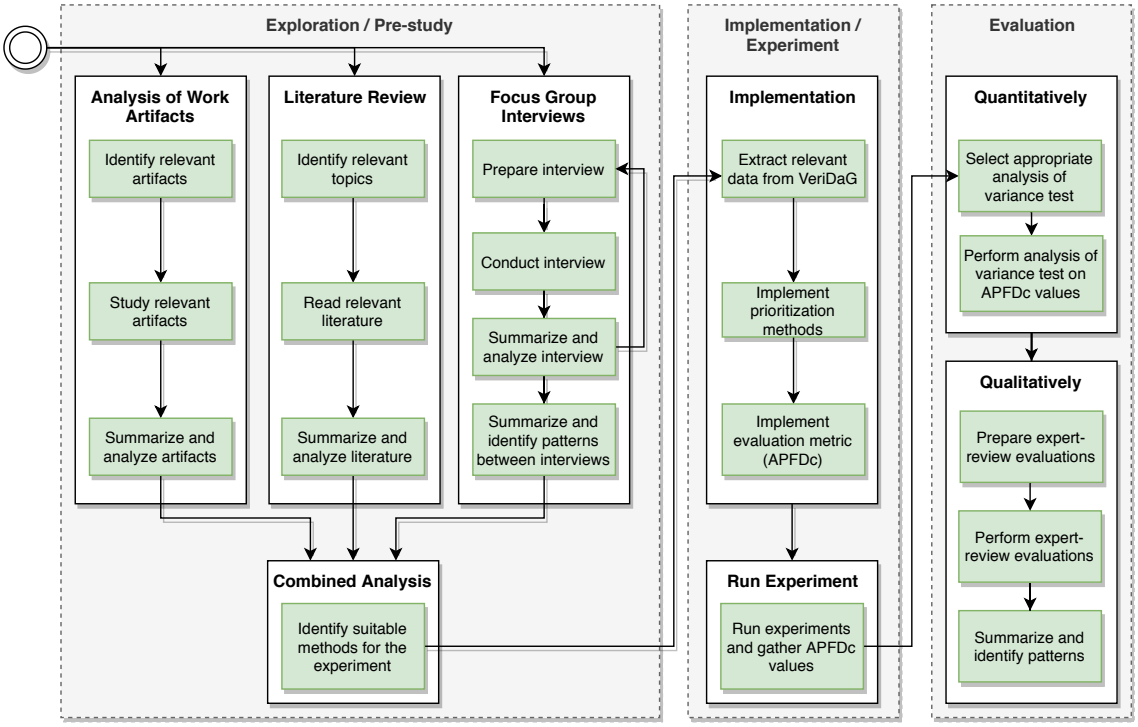

Figure 4.1: A schematic representation of the project at different abstraction levels: grey blocks represent major parts of the project, white blocks represent major steps in those parts, and green blocks represent the tasks involved in the aforementioned steps.

4.1 Research Methodology

This study was conducted as a case study involving action research at VCC (see section 2.1). Conducting focus group interviews and analyzing work artifacts, in combination with studying related work, were used as methods to gather domain knowledge and the data needed to form the basis for the experiment that was to later conducted to evaluate the proposed prioritization method. Expert-review evaluations were also conducted at the end of project, in the form of semi-structured interviews.

4.1.1 Case Study

A case study is an empirical research methodology focused on studying phenomena in their natural context [45], i.e. in a real world setting. Historically, case studies have often been used primarily for exploratory purposes, but the objective of a case study may, for instance, also be descriptive, explanatory, or improving [45]. Within software engineering, case studies are often focused on improvement, which resembles the action research methodology (see section 4.1.2). The studied *case* can be almost anything which is a "contemporary phenomenon in its real-life context" [45]. For instance, it may be an individual, a company, or a technology.

The case study research process consists of five major steps, according to Runesson et al. [45]:

1. Case study design: objectives are defined and the case study is planned
2. Preparation for data collection: procedures and protocols for data collection are defined
3. Collecting evidence: execution with data collection on the studied case
4. Analysis of collected data
5. Reporting

Ethnographic methods, such as interviews, focus groups, and observations, are often used during the data collection process [45]. It it also common that case studies start with a literature search [45]. The concept of triangulation is an important aspect that can help researchers reach stronger conclusions. Using multiple data sources (i.e. data triangulation), and multiple data collection methods (i.e. method triangulation) is encouraged [45]. While case studies are often primarily based on qualitative data, a combination of both qualitative data and quantitative data is often preferred, as it can provide stronger results [45].

4.1.2 Action Research

Action research (see section 4.1.2) is another well-known research methodology that is related to case studies [45]. The purpose of action research is to influence or change aspects of what is being studied [45], as opposed to case studies, where the researcher typically has more of an observing role [45]. The primary objective of studies following an action research process is often to improve some aspect of the phenomenon that is being studied [45]. The typical steps involved in the action research process, as described by McKay et al. [1], can be seen in figure 4.2.

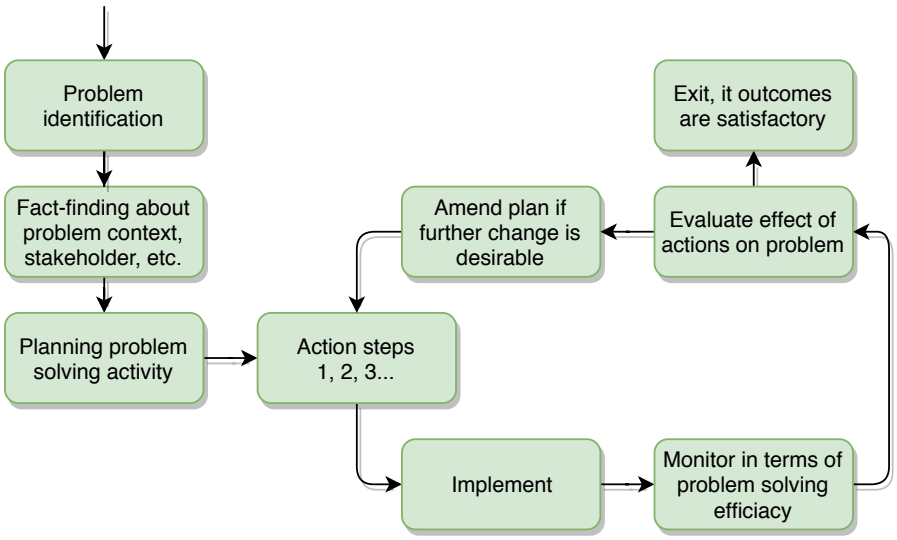

Figure 4.2: The steps involved in the action research process, as described by McKay et al. [1].

4.2 Pre-Study

This can be seen as the fact-finding phase of the action research process [1]. Different sources of information, following the principles of method and data triangulation [45], were used with the intent of seeing a broader picture and being able to make stronger conclusions.

4.2.1 Analysis of Work Artifacts

To gain further insights into the domain and the case at hand, various already existing work artifacts were studied to understand the challenges, limitations, and possibilities involved.

VCC stores and shares a lot of documentation in a digital knowledge base (Confluence), where employees document their systems and share knowledge with each other. This was a natural source of information to gain knowledge about the systems and techniques used at VCC, which would be useful to understand when, for instance, talking to different stakeholders and understanding the possibilities of the test infrastructure that exists today, and is planned to exist in the future. Taking part of this information made it easier to scope the project in a realistic way.

There are also various systems involved in the continuous integration testing process at VCC. The most relevant systems were analyzed to gain further knowledge about the domain and the case.

Systems used at VCC:

- VeriDaG (the data related to testing at VCC, see section 2.5)
- Volare (the continuous integration system used at VCC)

Documentation on Confluence:

- Terminology (e.g. terminology for autonomous drive in general, but also terminology related to the systems used at VCC (such as the entities of the VeriDaG domain model, as well as concepts used within Volare)
- Unified Modelling Language (UML) diagrams (e.g. for the VeriDaG domain model, showing concepts, and how they are connected. This information is useful in the sense that it shows what kind of information is available, or planned to be available, within the development and release flow)
- Flow charts (e.g. describing the continuous integration flow in Volare, all the way from deliveries from external suppliers, to displaying the test results in a continuous integration visualization software)

4.2.2 Literature Review

Previous studies with relation to the thesis topic were examined to gain further insights into the domain and the research questions. Studies done within general RTO were studied, with a primary focus on TCP techniques, but to some extent also TCS and TCM techniques, as they contain concepts that are related to version-specific TCP. Also, literature and studies covering autonomous vehicle and general automotive testing were investigated, as the work on automotive TCP is sparse. As the context of autonomous drive verification in combination with TCP is a bit special (e.g. almost always being black-box), various relevant concepts within TCP were studied. Some of the search terms that were included and combined when finding related work were: TCP, RTO, black-box, automotive, autonomous drive, cost-cognizant, time-aware, value-based, version-specific, changed-based, multi-objective, component-based, APFD, and APFD_C. Frequently cited studies from renowned sources, such as Institute of Electrical and Electronics Engineers (IEEE), were prioritized when identifying relevant studies.

4.2.3 Focus Group Interview

Focus group interviews were conducted to gain further domain knowledge and insights. Two groups with relevant participants working within autonomous drive development and verification were invited to separate group interviews to discuss topics related to the thesis. The main purpose of the focus group interviews was to gain a broader understanding of the autonomous drive verification domain and to gain more insights into the different research questions (mainly RQ1, but to some extent also RQ2 and RQ3). The results from the focus group interviews together with the findings from analyzing work artifacts and related work, provide a broader picture and different viewpoints on the same problem, following the principles of method triangulation.

The participants of the focus group interviews were carefully selected according to their field of expertise (e.g. solution architecture, VeriDaG, TAAS, virtual testing, active safety test methods, continuous integration). The participants all have in common that they are working within autonomous drive development or verification, but that their specific expertise belong to slightly different fields, which would

enable the focus group interviews to get different viewpoints on the discussed topics. The author's managers at VCC, with lots of knowledge about the thesis topic as well as the organization, suggested suitable people after hearing about the purpose and agenda of the focus group interviews. Multiple sources, e.g. Krueger [46], recommend that the number of focus group interview participants should be 5-10. Eleven subjects (who appeared to be available at the given time slot according to the appointment booking system used at VCC) were invited to the first focus group interview, as inviting a few extra participants is recommended in case there are subjects who do not show up. Participants in the second focus group interview were recruited in a similar fashion.

The focus group interviews started with a short introduction of myself and the thesis, followed by an outline of the meeting agenda, as well as an overview of established focus group interview guidelines. A number of topics and open-ended questions were prepared prior to the focus group interviews (see table 4.1). These questions formed the basis of discussion during the two sessions. The focus group interviews were also recorded, to facilitate the subsequent analysis, as only taking notes would likely result in some valuable information being forgotten.

The systematic analysis process of the focus group interview results was primarily based on identifying frequently recurring themes and discussed ideas (e.g. formulations, words, and concepts) that were emphasized and mentioned by multiple participants, both within and across the two sessions. The analysis of the individual interviews occurred immediately after the interviews had been conducted, while information and impressions were still fresh in mind. After both focus group interviews had been done, common themes between the two sessions were identified.

Question and Example
1. Which complexities, restrictions, or technical difficulties exist within autonomous drive verification? For instance, most deliveries are in binary format, which makes white-box testing difficult, as access to the source code is highly restricted.
2. What makes a test activity considered valuable, or what is considered a good prioritization order? For instance, objectives could be to find as many faults as early or cheaply as possible, to cover as many requirements as quickly as possible, to primarily test the changed features, etc.
3. How do you prioritize what to test as of today? When a new delivery is received from the suppliers, how do you prioritize what to test as of today? In other words, based on which information and with which objectives in mind?
4. Which factors should be considered when prioritizing test cases (i.e. be part of the proposed prioritization algorithm)? For instance, factors such as previous execution time of the test case, fault-revealing history of the test case, and modified features in the release of the SUT (feature coverage of the test cases) could be considered.
5. Rank the factors by importance/use the 100-point method to weight them (assuming they are normalized) The prioritization weight factors in table 4.2 were ranked by perceived importance, using the 100-point method.

Table 4.1: The open-ended questions and topics that were prepared prior to the focus group interviews.

Factor	Description
Execution cost	The cost incurred from executing the test case.
Execution time	The amount of time it takes to run the test case.
Fault-revealing history	The historical effectiveness of the test case. How many faults the test case has previously detected.
Changed feature coverage	Does the test case cover the features that were changed in the release?
Requirement coverage	How many requirements the test case covers.
Static priority	A priority assigned manually by some person.
Creation date	When the test was created. Can be used to prioritize new test cases.
Time since last execution	Was it a long time since the test case was run? Can be used to ensure that test cases eventually get executed after a period of time.

Table 4.2: Prioritization weight factors and their descriptions. The factors were ranked by the focus group participants using the 100-point method. The factors were based on prioritization factors observed in related literature (e.g. [2, 3]).

4.2.4 Data Analysis

The results from the different steps of the pre-study were combined and evaluated to identify methods that seemed suitable to proceed to investigate in the upcoming experiment. This can be seen as the step of the action research process that is related to planning the problem solving activity.

4.3 Implementation

4.3.1 Prioritizer

A program called *Prioritizer* was implemented by the author to evaluate and compare different TCP methods against each other. *Prioritizer* extracts and parses data from VeriDaG, and uses this data in the different prioritization algorithms. It also implements a method to evaluate the performance of the algorithms, using the $APFD_C$ metric. The program was implemented in Python, using Jupyter Notebook[1]. This corresponds to the implementation phase of the action research process.

4.3.2 Constructing the Prioritization Methods

Using *Prioritizer*, relevant JSON data was extracted and calculated from VeriDaG in order to implement and evaluate the prioritization methods.

[1] https://jupyter.org/

For instance, data such as expected duration and historical fault-detecting abilities for the different test cases were calculated by iterating over the historical test results for the test cases that had been executed in previous versions (BaselineVersion) of the delivered SUT. The expected duration can be calculated by averaging the historical durations, and the historical effectiveness with regards to fault-detecting abilities can be calculated by dividing the number of failures by the number of runs, for each test case. The average stability of each test case can be calculated in a similar fashion.

The idea, in order to perform version-specific TCP given the fairly limited information available about the changes made, is to calculate a correlation between changed Releasables and failing test cases. This is achieved by looking at the Releases in the current BaselineVersion and the previous one, and comparing which Releases of which Releasables are different. This resembles treating the granularity of change on a rather high level, which can be compared to knowing which files have changed, instead of knowing which lines of codes have changed, in a specific version of the SUT. The hypothesis is that this abstraction level still provides some value when prioritizing, compared to totally ignoring information about the changes. Hopefully this correlation enables us to say things like: "when this particular file (Releasable) changes, these test cases often result in a failure". Let us consider an example for illustrative purposes:

BaselineVersion:384 has verdict Success.
BaselineVersion:385 has verdict Failure, where TestCase:29 is failing.

Thus, it's likely that some change in BaselineVersion:385 causes TestCase:29 to fail, at least if it's not an intermittent failure.

BaselineVersion:384 contains:

- Release:126 of Releasable:70
- Release:131 of Releasable:14
- Release:140 of Releasable:30

BaselineVersion:385 contains:

- Release:126 of Releasable:70
- Release:131 of Releasable:14
- Release:141 of Releasable:30

Here, we can see that a new version (Release:141) of Releasable:30 is a part of the delivery. Thus, we can draw the conclusion that it's likely that a change in Releasable:30 causes TestCase:29 to fail. But once again, we don't know for sure, as the failure of TestCase:29 could be due to an intermittent behaviour. However, the hypothesis is that it, on average, should be possible to find a correlation between changes in specific Releasables and specific test cases. This information should be useful when doing version-specific TCP, as information about what Releasables have changed is available as of today in the test automation flow at VCC. Once again, these are general concepts that are not specific to VCC, but instead, Releasables can be seen as changed software components in a new software version.

Prioritization Factors

The identified prioritization factors that were extracted and calculated from the data are described in more detail below:

GFR (General Failure Rate)

$GFR(t, v)$ denotes the historical fault-detecting abilities of test case t based on the first v executions.

Rationale. *Tests that usually fail are likely to fail again.*

$$Verdict(t, v) = \begin{cases} Success, & \text{if test case } t \text{ resulted in a success on execution } v \\ Failure, & \text{if test case } t \text{ resulted in a failure on execution } v \\ Error, & \text{if test case } t \text{ resulted in an error on execution } v \end{cases}$$

$$Failures(t, v) = \sum_{i=1}^{v} \begin{cases} 1, & \text{if } Verdict(t, i) = Failure \\ 0, & \text{otherwise} \end{cases}$$

$$\mathbf{GFR(t,v)} = \frac{Failures(t, v)}{v}$$

D (Duration)

$D(t, v)$ denotes the average historical duration of test case t based on the first v executions.

Rationale. *If two test cases are equally good at finding faults, then the short one should be prioritized.*

$Duration(t, v)$ denotes the duration of test case t on execution v.

$$\mathbf{D(t, v)} = \frac{\sum_{i=1}^{v} Duration(t, i)}{v}$$

VSFR (Version-specific Failure Rate)

$VSFR(t, v)$ calculates the average $Correlation(r, t, v)$ for a test case t, as many Releasables might change between two BaselineVersions.

Rationale. *Tests that usually fail when a particular file (*Releasable*) changes are likely to fail again when that file changes.*

$Correlation(r, t, v)$ provides a mapping between changed Releasables and test cases. It tells us how often t results in a failure when Releasable r changes, based on the first v executions.

$$Correlation(r, t, v) = \frac{\text{number of times } t \text{ failed when } r \text{ changed in the first } v \text{ executions}}{\text{number of times } r \text{ changed in the first } v \text{ executions}}$$

$ChangedReleasables(v)$ = the Releasables that changed between version $v-1$ and v

$$\mathbf{VSFR}(\mathbf{t}, \mathbf{v}) = \frac{\sum_{r \in ChangedReleasables(v)} Correlation(r, t, v)}{|ChangedReleasables(v)|}$$

RCF (Recent Consecutive Failures)
$RCF(t, v)$ is used to put an upper limit (3) on the number of failed executions that is considered when returned by $FailsInRow(t, v)$.

Rationale. *Tests that failed at the last execution are likely to fail again, as the faults they discovered probably haven't been fixed yet. Consecutive failures indicate that the failures are consistently occurring. This should be taken into account, but only to some extent (thus the upper limit), as test cases that have failed for many executions might otherwise diminish the RCF values of other test cases that just recently started failing.*

$FailsInRow(t, v)$ denotes the number of failed executions in a row leading up to execution v of t, if and only if execution v of t resulted in a failure.

$$FailsInRow(t, v) = \begin{cases} 1 + FailsInRow(t, v-1), & \text{if } Verdict(t, v) = Failure \\ 0, & \text{otherwise} \end{cases}$$

$$\mathbf{RCF}(\mathbf{t}, \mathbf{v}) = min(3, FailsInRow(t, v))$$

From VeriDaG, we theoretically also have access to:

- The ReleasableFocus of each test case
- The ModifiedFeatureAreas of each Release

However, as stated earlier, this data is not yet a part of the test automation flow at VCC. It currently only available for theoretical purposes in VeriDaG, to model a possible future in which this information is available in reality. Thus, in order to develop a method that is as applicable and general as possible, these theoretical factors were not considered in this thesis.

Normalization
The values of the different factors $GFR(t, v)$, $D(t, v)$, $VSFR(t, v)$, and $RCF(t, v)$ are re-scaled using min-max normalization to fit the values in the range [0,1], to facilitate comparing and combining the values with each other.

$$MaxValue(\mathbf{f}, v) = \max_{t=1}^{n} \mathbf{f}(t, v)$$

$$MinValue(\mathbf{f}, v) = \min_{t=1}^{n} \mathbf{f}(t, v)$$

Each factor $\mathbf{f}$ is re-scaled using the normalization function $\mathbb{N}$:

$$\mathbb{N}(\mathbf{f}, t, v) = \frac{\mathbf{f}(t, v) - MinValue(\mathbf{f}, v)}{MaxValue(\mathbf{f}, v) - MinValue(\mathbf{f}, v)} \tag{4.1}$$

The Proposed Prioritization Method

When prioritizing version number v, the proposed method calculates the value $V(t, v-1, w_1, w_2, w_3, w_4)$ for each test case t by combining the four factors using a weighted sum. Note that w_2 is multiplied with $1 - \mathbb{N}(D(t, v))$ as a short duration is considered more valuable than a long duration:

$$\begin{aligned}\mathbf{V}(t, v, w_1, w_2, w_3, w_4) = w_1 * \mathbb{N}(\mathbf{GFR}(\mathbf{t}, \mathbf{v})) \\ + w_2 * (1 - \mathbb{N}(\mathbf{D}(\mathbf{t}, \mathbf{v}))) \\ + w_3 * \mathbb{N}(\mathbf{VSFR}(\mathbf{t}, \mathbf{v})) \\ + w_4 * \mathbb{N}(\mathbf{RCF}(\mathbf{t}, \mathbf{v}))\end{aligned} \tag{4.2}$$

The weights can either be assigned manually or automatically using hyper-parameter optimization techniques such as *grid search* or *random search*. During the experiment, the weights were optimized using *grid search*, which guaranteed the optimal set of weights to be found (out of the possible combinations). The different combinations, WS, were generated by combining all values between 0 and 100 using increments of 5 ($X = \{0, 5, 10, \ldots, 100\}$), such that the sum of the combination was equal to 100 (e.g. $\{20, 10, 45, 25\}$). The number of possible values in X was 21, which gave a total of $21^4 = 194481$ combinations prior to filtering, and 1771 combinations where the sum was equal to 100. As can be seen, the number of combinations grows rather rapidly. Grid search can be a reasonable approach if the number of weights (and possible values for each weight) is relatively low, otherwise an approach such as *random search* might be more efficient.

$$WS = \{(w_1, w_2, w_3, w_4) \in X^4 \mid w_1 + w_2 + w_3 + w_4 = 100\}$$

Each set of weights in WS was then evaluated on historical data (same approach as described in section 4.4.1) and optimized using the APTTF metric. APFD_C could also have been used, but for the sake of proposing a method that is as applicable as possible in reality, APTTF was selected as it does not require present faults to be known or seeded into the SUT. The set of weights that achieved the highest average APTTF score for the MULTI-PRIO prioritization method was picked as the optimal set of weights, and was later used in the final experiment.

Constructed Prioritization Methods

A number of different prioritization methods can be constructed from the prioritization factors. The following prioritization methods were constructed and compared to each other during the proceeding experiment.

- INITIAL: Sort test cases according to the initial sequence (do nothing)
- RANDOM: Sort test cases according to a random permutation
- GFR-PRIO: Sort test cases in descending order according to $GFR(t, v)$.
- D-PRIO: Sort test cases in ascending order according to $D(t, v)$.
- VSFR-PRIO: Sort test cases in descending order according to $VSFR(t, v)$.
- RCF-PRIO: Sort test cases in descending order according to $RCF(t, v)$.
- MULTI-PRIO: Sort test cases in descending order according to $V(t, v, w_1, w_2, w_3, w_4)$.
- MULTI-EQ-PRIO: Sort test cases in descending order according to $V(t, v, 25, 25, 25, 25)$.

4.4 Evaluation

Using *Prioritizer*, the prioritization methods were evaluated through an experiment utilizing the $APFD_C$ metric. They were also evaluated through expert-review evaluations, following the principles of method triangulation. This corresponds to the evaluation phase of the action research process.

4.4.1 Experiment

The prioritized test case sequences produced by the prioritization methods were compared against the initial test case sequences (INITIAL), as well as randomized sequences of the test cases (RANDOM), for the test cases in test activities used to test different BASELINEVERSIONS. Using VeriDaG, a regression test history was generated for 56 baselines with test results from previous BASELINEVERSIONS. For each of these baselines, the last BASELINEVERSION was considered for prioritization, and the test results gathered from previous versions were used for analysis of historical performance. Thus, the prioritization was only made using information from previous test executions. 49 out of the 56 selected BASELINEVERSIONS contained detected faults and were therefore selected for analysis (as the $APFD_C$ metric requires at least one fault to be discovered by the test suite). Each BASELINEVERSION has a number of test activities, containing test suites, which contain test cases. A total of 93 test activities were used to verify the 49 faulty BASELINEVERSIONS. The 93 test activities contained a total of 3157 test cases. The test cases within each of these 93 test activities were prioritized, and $APFD_C$ values (as well as APTTF values) were gathered at each of these 93 occasions for every prioritization method. The RANDOM method prioritized every test activity 100 times according to random permutations, and reported the average $APFD_C$ and APTTF values.

The significance of the results were assessed using hypothesis testing, where the null hypothesis is that the $APFD_C$ values are equal for all prioritization methods. A Friedman test (at a significance level of 0.001) was used to test for differences between the different treatments (i.e. prioritization methods). The Friedman test is the non-parametric version of the one-way ANOVA test with repeated measures, and was selected because the assumptions necessary to run the one-way ANOVA test with repeated measures could not be guaranteed (e.g. the data has to be normally distributed). Post-hoc analysis was conducted using pair-wise comparisons utilizing a Nemenyi test. The Nemenyi test is somewhat more conservative than pair-wise Wilcoxon sign-rank tests, and is developed to account for making multiple comparisons which otherwise increases the risk of making a Type I error. Therefore, it does not require a p-adjustment (e.g. a Bonferroni correction), as opposed to pair-wise Wilcoxon sign-rank tests.

To check how much the amount of historical data affected the performance of the prioritization methods, an additional similar experiment was executed in which the number of previous versions was reduced. In this experiment, the historical test data from a maximum of five previous executions was considered as the regression history, meaning that the sixth BASELINEVERSION was prioritized based on data gathered only from five previous runs. This was done to show what could be expected when

little historical data was available, and to show how reliant the performances of the prioritization methods were on the amount of historical data. This lead to 83 test activities being prioritized, containing 2972 test cases in total.

4.4.2 Expert-review Evaluation

Expert-review evaluations were done to gain further insights into the performance and usability of the proposed method. They were conducted as semi-structured interviews, using a set of pre-defined questions and statements (see table 4.3) as the basis. A team of experts with relation to test automation at VCC were invited to participate. Apart from openly discussing the different topics and statements of the semi-structured interview, the participants also ranked each statement on a 5-point Likert scale: *Strongly disagree, Disagree, Don't know, Agree, Strongly Agree*, and were encouraged to talk freely about the statements and motivate their rankings. In this way, both qualitative and quantitative data were captured during the evaluations.

The proposed method was demonstrated by first presenting the four factors it considers, as well as the weighted sum used to combine them. Also, example data produced by VeriDaG (see figure 2.1) was presented to give the participants an idea of the data that was available for prioritization. A set of test cases (with some meta-data) used to verify a particular BaselineVersion was also provided, where the test cases were presented in both the initial order and the prioritized order (see table 4.4). This example demonstrated how the prioritization method worked in practice, which enabled the participants to get a broader understanding of the method before evaluating its performance and usability. To avoid biased opinions, the participants were not exposed to any results already gathered during the experiment (e.g. $APFD_C$ graphs) until question 9-10, where the participant got to see partial results from the experiment (figure 5.4 and table 5.3)), as this information seemed necessary for the participants to make informed statements regarding these two particular questions.

Expert-Review Evaluation Form
1. The prioritization method seems to be prioritizing test cases in an efficient way
2. The prioritization method considers valuable factors
3. The prioritization method ignores valuable factors
4. The data used by the prioritization method is available as of today at VCC
5. The data used by the prioritization method is expected to be available within the near future at VCC
6. The prioritization method could be implemented as of today at VCC
7. The prioritization method could be implemented within the near future at VCC
8. The simulated data produced by VeriDaG seems realistic
9. The prioritization method could help improve the testing efficiency at VCC
10. The prioritization method could help improve the testing efficiency at other companies working within the autonomous drive domain

Table 4.3: The expert-review evaluation form containing the statements that were ranked on a 5-point Likert scale: *Strongly disagree, Disagree, Don't know, Agree, Strongly Agree* by the experts.

Test Case	Failures	Runs	GFR	D	VSFR	RCF	V(t,v)
#3098	24	76	0.316	76 887	0.0462	0	53.34
#3099	0	76	0	178 872	0	0	10.25
#3100	24	76	0.316	270 620	0.0339	0	27.43
#3101	33	76	0.434	192 327	0.0704	0	54.67
#3102	1	76	0.013	166 797	0.0025	0	13.11
#3103	38	76	0.5	46 869	0.0727	1	83.33

Table 4.4: Example of historical data and computed factors used to sort `TestActivity:147` containing 6 test cases, used to verify `BaselineVersion:77` of `Baseline:44`. As can be seen in the *Runs* column, the test cases have been run 76 times before (as version 77 is the one currently being prioritized). The test sequence is (#3098, #3099, #3100, #3101, #3102, #3103) according to the INITIAL method, and (#3103, #3101, #3098, #3100, #3102, #3099) according to the MULTI-EQ-PRIO method, as can be seen from the *V(t,v)* column.

5
Results

5.1 Pre-study

5.1.1 Analysis of Work Artifacts

The analysis of work artifacts showed that the continuous integration system, Volare, used at VCC consists of many complex and interacting parts. Deliveries mainly come from external suppliers. The deliveries are almost exclusively in binary format (Versatile Binary Format (VBF)), thus, the test engineers have limited or no access to the source code. Deliveries in binary format are not only seen at VCC but often in other automotive companies as well (and often in component-based systems, in general) [28]. Each delivered binary targets a specific sensor or ECU and is tested in a bottom-up integration testing approach, from sensor-level all the way up to the complete car.

VCC is currently performing testing of components that will eventually go into an autonomous drive system, but continuous integration at VCC is still in development, meaning that only some testing has been fully automated thus far. In general, the whole VCC organization is currently moving towards a more agile way of working.

Depending on the target of testing, different types of testing are needed. Many of these activities require access to hardware (e.g. HIL rigs and boxcars), which have limited availability and need to be booked.

A system called TAAS is being developed to increase the efficiency of testing. TAAS is the system where an implementation of the proposed method would be integrated and used.

5.1.2 Literature Review

The results from the literature review can be seen in chapter 3.

5.1.3 Focus Group Interview

A total of six participants attended the first focus group interview (see table 5.1). Permission to record the interview was granted, and the interview was recorded. There was also a person taking notes (not included in the six participants). Five other persons attended the second focus group interview (see table 5.2), which was conducted in a similar fashion as the first focus group interview, to make the different sessions easier to compare.

Participant	Participant Role / Area of Expertise	Experience (Years)
1	Technical Expert (Active Safety Functions)	20
2	System Tester (ADAS/ASDM)	2.5
3	System Architect (Verification & Validation of AD Functionality)	4
4	System Architect (Verification & Validation of AD Functionality)	5
5	Product Owner (Active Safety HIL)	4.5
6	Senior Analysis Engineer (Autonomous Drive)	12

Table 5.1: The six participants of the first focus group interview with their corresponding roles or areas of expertise, as well as their levels of experience.

Participant	Participant Role / Area of Expertise	Experience
1	System Architect/System Verification Leader (ADAS)	19
2	SPAS Simulation Development (Virtual Verification)	7
3	Function Verification & Data Strategies (Method and Tools)	18
4	System Architect (Continuous Integration, TAAS, VeriDaG)	10
5	System Architect (Continuous Integration, TAAS, VeriDaG)	11

Table 5.2: The five participants of the second focus group interview with their corresponding roles or areas of expertise, as well as their levels of experience.

The following recurring themes and common ideas were identified from the focus group interviews:

Issues and Differences to Other Domains

Multiple participants described that having external suppliers is an issue, mainly in relation to communication. Some of the participants mentioned that they are not used to working with external developers, and explained that it was easier before when the test engineers could just go and talk to the developers in person.

Testing can be performed on multiple levels, but some of the participants don't believe it's necessarily very different from testing other types of software. However, there seems to be a consensus that they have a lot of data involved, and that autonomous drive verification requires more data than what is usually seen within

other domains. Their testing is also often non-deterministic, in the sense that probabilities are used to generate scenarios for tests, etc.

Some participants mentioned that it is hard to write requirement specifications for autonomous drive functionality, as it has to "work for everything" — not only for specific well-defined scenarios, which might be more easily defined within other verification domains. Defining a self-driving car's behaviour through requirements is a complex task, which also makes testing of said behaviour inherently complicated.

Prioritization as of Today

There seem to be slightly different perceptions between the participants about the level of prioritization that is done as of today, likely due to differences between testing at different departments that the participants belong to. VCC has not yet reached full maturity within continuous integration, so most testing is currently done manually, which can lead to differences as the process is not yet fully automated and standardized.

Some participants feel that little to no prioritization is done as of today, for various reasons. One of the reasons is that they still have the possibility of testing almost everything without a lot of prioritization, due to rather infrequent deliveries (once every few weeks) and because they do not have very many test cases. However, there is a consensus between the participants that this will not be the case in the near future, as deliveries are becoming more and more frequent (the aim is to have daily deliveries), and the number of test cases as well as the complexity of said test cases grow continuously. As an example, they mentioned the Society of Automation Engineers (SAE) levels, and how reaching higher SAE levels will require a lot more test cases, and of much higher complexity, which calls for the need of TCP. Another reason for the little prioritization is that many are currently unaware of how to prioritize, even though they might want to. The participants report having a hard time understanding what has been changed in a delivery, due to very limited, or sometimes non-existing change logs from the suppliers. They mention that even if they actively ask the suppliers about what has been changed, it is still often hard to know what to test.

Other participants, on the other hand, state that prioritization actually is being done today, albeit manually and rather informally. They describe what could be considered a manual version-specific meta-content-based TCP approach, where testing the delta of the release is prioritized (i.e. testing what was changed in the software, such as fixes for known issues). Information about the delta for the release is provided by the external suppliers as release notes in a human-readable format. The changes are usually expressed on a high abstraction level, such as at sub-system level or higher (i.e. they do not know which lines of code changed in the delivered software, but they are provided information about the changes on a higher level). Sometimes, they also get internal test reports from the testing performed by the suppliers themselves, which can guide the testing that is done at VCC. They also mention that "green tests" (i.e. tests that usually pass) have the lowest priority and are run last.

This way of prioritizing is closely related to what can often be seen in history-based TCP techniques.

Apart from the limited prioritization that is done today, based on release notes and historical performance, they also mention that they currently order, or plan to order, their test cases by increasing complexity. For instance, given that the suppliers report that TSI functionality has been updated, they would start out by testing the basics first (e.g. scenarios including stop signs), and then move towards robustness testing (e.g. scenarios including unusual traffic signs), as ensuring that the basics work as indented is perceived as more important than testing unusual corner cases.

Some participants also mention that they perform what could be seen as another kind of prioritization, for their testing activities based on testing levels, where they start out by testing on lower levels, i.e. unit level and then proceed to higher integration levels all the way up to system level. They explain that they don't move directly to testing in the car, and they prefer to avoid testing in the car if previous testing shows that the system contains errors.

> *"We are often quite unaware — we get a black box and we don't know what's inside. Prioritizing is difficult. That is why we tend to run whatever we can."*
>
> - Participant 4, *Focus Group Interview #1*

> *"We get the VBFs with delivery notes about the contents and the delta. Testing the delta is what is most important. Tests that are always green have the lowest priority, and are run at last."*
>
> - Participant 1, *Focus Group Interview #2*

Aspects That Make a Test Considered Valuable

When discussing which properties and factors make a test considered valuable (primarily focusing on other factors than those already mentioned in the example of the question), and which factors that they would recommend the prioritization method to consider, these were the different aspects that were mentioned:

Stability & Robustness: The participants argued that tests that produce consistent results given the same input are valuable, as they may otherwise be unpredictable and hard to evaluate.

Functional Complexity: Tests that verify basic functionality in the SUT are perceived as more valuable, than tests that test complex functionality and corner-cases. However, they stress that this is primarily from a project time-line perspective as it can save them a lot of time, and not necessarily from a safety perspective.

Safety Conformance: In general, the participants believe that tests that verify safety-critical features are considered more valuable, and should be prioritized over test cases that might even be allowed to fail.

Fidelity: A measure of how detailed or realistic the test is can affect its perceived value, according to some participants. In some cases, such as advanced simulations where the virtual environment has to be realistic, a high fidelity might be needed.

However, this might also lead to longer execution times. If a test of low fidelity fails, the participants argue that the failure is often perceived as more reliable than if a test of high fidelity fails, as that might be due to issues following the high complexity of the test itself.

Historical Performance: The participants mention the example from the initial question example again, and stress that testing things which have been troublesome in the past has a high priority, and that testing what always works has the lowest priority.

Current Model Quality (CMQ): One participant mentioned CMQ, which is the department responsible for handling problems that have been reported by customers (i.e. problematic functionality in cars that have already been sold to customers). These issues are often of high priority, as they directly affect the customers. The participant mentioned that it would be interesting to extract this data automatically, to prioritize test cases that test the affected functionality.

On a more general note, some participants argue that what is considered valuable depends on whom you ask. For instance, top management might say that the most important test cases are those that detect errors which the customer might experience, while the team developing sensors might say that the most valuable tests are those that test their sensors.

Participant Weighting of Possible Prioritization Factors

The participants were also asked to assign weights to different factors, using the 100-point method (i.e. where the sum of the assigned weights equals 100). A higher number indicates that the subject perceived a factor as more important than a factor with a lower weight.
The average of the weights from the two focus group interviews can be seen in figure 5.1. There seem to be some distinguishable patterns that can be identified from the graph. For instance, Changed Feature Coverage and Fault-revealing History are considered as two of the most important factors by both groups, while Creation Date is considered the least important one.

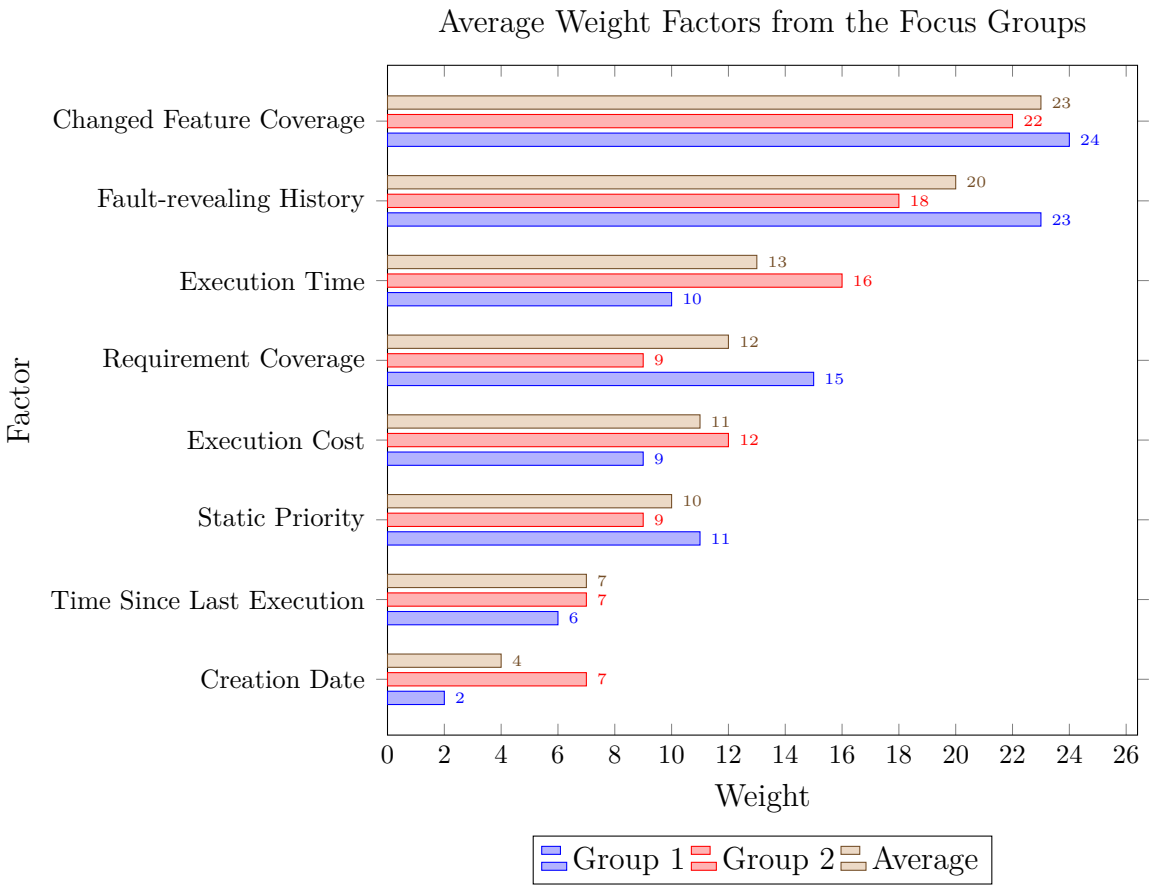

Figure 5.1: The average weights of the factors, as assigned by the focus group participants using the 100-point method. The graph shows us that, on average, the focus group participants consider Changed Feature Coverage to be the most important factor and Creation Date the least important.

5.1.4 Data Analysis and Pre-Study Conclusion

The study of related work and work artifacts, as well as the findings from the focus group interviews, show that deliveries are often delivered from external suppliers, in binary format. Many already existing test prioritization methods require access to the source code, for instance when analyzing code-coverage. Thus, in the case of VCC (and many other automotive companies), almost all prioritization methods that require access to the source code can be discarded, leaving us with TCP methods that are based on non-code-based objectives. When it comes to non-code-based TCP methods, there are, for instance, history-based methods. Within black-box TCS, there are meta-based approaches, but the performances of these methods are often highly reliant upon the level of knowledge-sharing between external suppliers and the test engineers, as in the case of VCC. As seen in the focus group interviews, the information that gets shared between external suppliers and VCC is often on a high level of abstraction, and sometimes a bit hard to understand. Together, these things indicate that it might be a good idea to propose a prioritization algorithm that is not too reliant upon having very detailed information from external suppliers. However,

some meta-data that is reasonable to expect from suppliers would still likely be useful for a prioritization algorithm, but that data will likely need to be standardized and machine-readable, to allow the prioritization algorithm to operate in an automated fashion.

Based on the findings from the pre-study, the proposed method needs to be black-box (no access to source code), multi-objective (we cannot only look at fault detecting abilities for instance, but need to also consider aspects such as duration), cost-cognizant (cost/time needs to be taken into consideration), version-specific (we want to prioritize in a way that is optimal for the specific version of the SUT). For each of these properties, inspiration can be taken from previous work, but there is no existing method that takes all of these aspects into account. Rather, something which can be seen as a combination of existing methods covering these individual aspects seems appropriate to proceed with in the experimentation phase.

Some prioritization methods that could be used for inspiration and as a basis are the ones proposed by Fazlalizadeh et al. and Marijan et al. They are both interesting and seem applicable in the sense that they are history-based (i.e. black-box) multi-objective TCP techniques, making use of data that seems to be available within this context. The approach proposed by Marijan et al. is also cost-cognizant. However, neither of these approaches is version-specific. The version-specific difference-based approaches explored by Elbaum et al. seem promising, but require access to the source code. However, the principles of difference-based coverage can likely be used on a higher level of granularity as well (e.g. changed files), without the need of source code access. One approach that operates on a higher level of granularity is the TCS approach developed by Ekelund, which correlates changes in the SUT to test cases on package level. Something similar to these version-specific approaches can likely be done in this study as well, as some high-level information about changes between software versions is available today, at VCC and likely at many other automotive companies as well.

5.2 Evaluation

5.2.1 Experiment

Experiment Run with Complete Regression History

The experiment was conducted in which the 93 test activities were prioritized at their latest baseline version, by each of the prioritization methods. Figure 5.2 visualizes the APFD_C values gathered at all 93 test activities for each prioritization method. Figure 5.3 shows the same APFD_C values visualized in a box plot, where information such as averages and variability between the gathered values can be seen for the different prioritization methods. Table 5.3 shows the average APFD_C and APTTF values in a tabular format.

The APFD_C values gathered during the experiment for each of the prioritization methods (see appendix A.1) were analyzed using a Friedman test, which showed that there was a significant difference ($\chi^2 = 517.2$, $p = 1.607^{-107} < 0.001$) between

the $APFD_C$ values depending on the prioritization method, when comparing *some* pair of prioritization methods. A total of 28 2-combinations of prioritization methods could be constructed from the 8 different methods. The number of combinations is given by the binomial coefficient ($\binom{8}{2} = 28$). Post-hoc analysis using a Nemenyi test found significant differences for 20 out of 28 pairs of prioritization methods (see table 5.4). There are no significant differences between INITIAL & RANDOM, INITIAL & D-PRIO, RANDOM & D-PRIO, GFR-PRIO & VSFR-PRIO, GFR-PRIO & RCF-PRIO, VSFR-PRIO & RCF-PRIO, RCF-PRIO & MULTI-EQ-PRIO, or MULTI-PRIO & MULTI-EQ-PRIO. This means that post-hoc analysis shows that there is a significant difference between INITIAL & MULTI-PRIO as well as RANDOM & MULTI-PRIO. The $APFD_C$ values gathered during the experiment for the MULTI-PRIO method can be seen in figure 5.4, where the $APFD_C$ values for the INITIAL and MULTI-PRIO method are also visualized as baselines for comparison. Similar graphs can be seen for the other prioritization methods in the appendix. Table 5.3 shows that the MULTI-PRIO method achieves the highest average $APFD_C$ value (0.9103) out of all the compared prioritization methods, as well as the lowest APTTF value (0.1359).

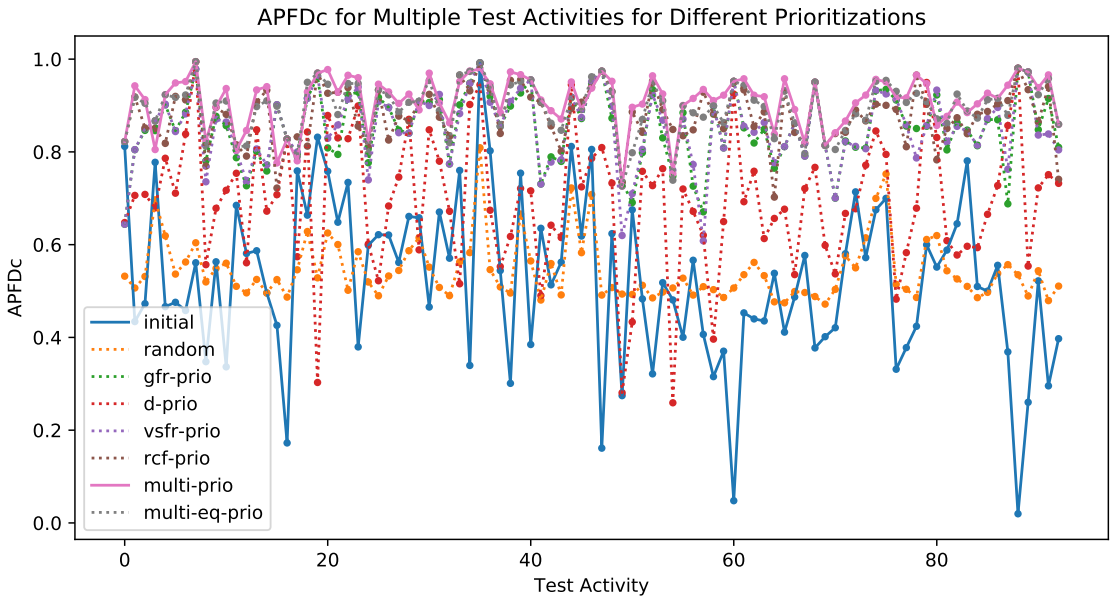

Figure 5.2: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for each of the prioritization methods. The solid pink line corresponds to the MULTI-PRIO method, and the solid blue line corresponds to the INITIAL method.

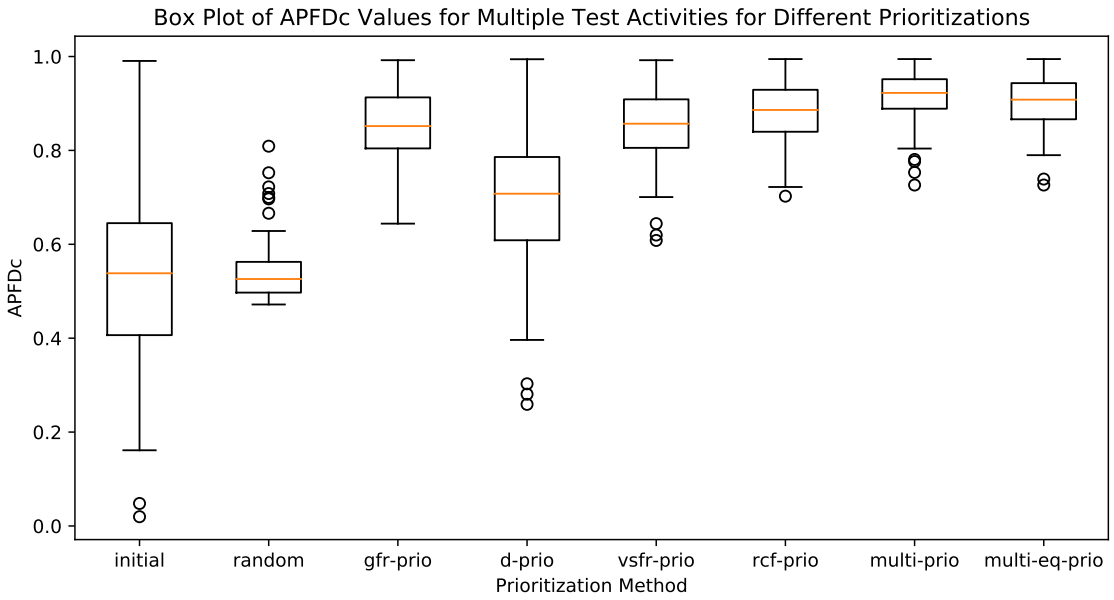

Figure 5.3: Box plot visualizing the $APFD_C$ values gathered during the experiment on the 93 prioritized test activities (based on the full test history) for each of the prioritization methods.

	Prioritization Method	Average APFDc	Average APTTF
0	initial	0.5251	0.5476
1	random	0.5439	0.5255
2	gfr-prio	0.8509	0.2092
3	d-prio	0.6952	0.3625
4	vsfr-prio	0.8510	0.2106
5	rcf-prio	0.8799	0.1767
6	multi-prio	0.9103	0.1359
7	multi-eq-prio	0.8975	0.1528

Table 5.3: The average $APFD_C$ and APTTF values gathered during the experiment for the different prioritization methods (based on the full test history). The MULTI-PRIO method has the highest $APFD_C$ value at 0.9103, followed by MULTI-EQ-PRIO and RCF-PRIO, at 0.8975 and 0.8799 respectively. The lowest values are achieved by INITIAL and RANDOM, at 0.5251 and 0.5439 respectively.

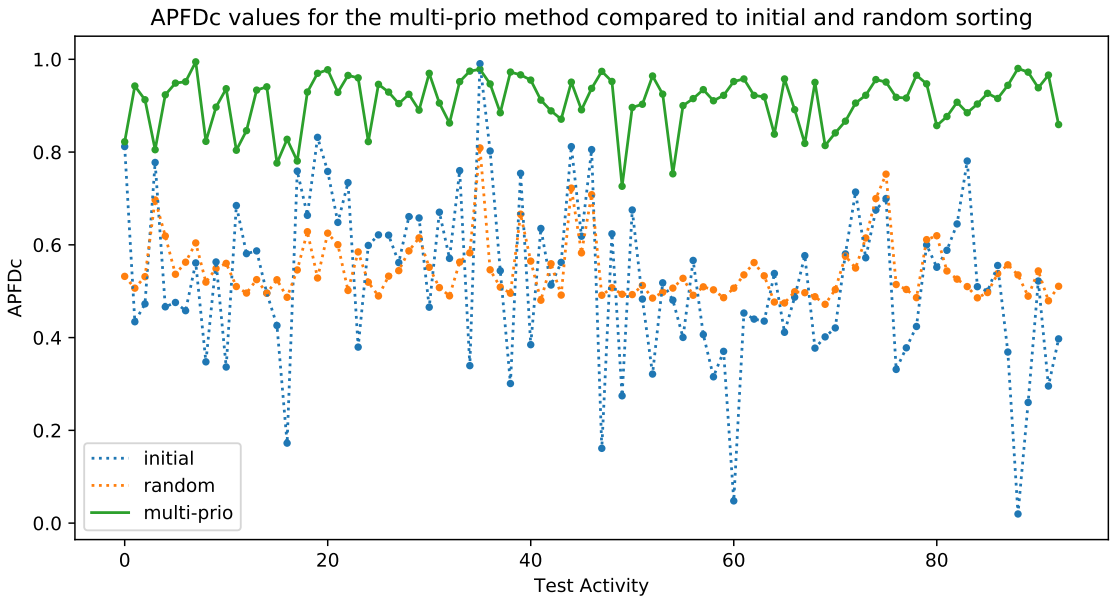

Figure 5.4: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the MULTI-PRIO method, the dashed blue line corresponds to the INITIAL method, and the dashed orange line corresponds to the RANDOM method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

	initial	**random**	**gfr-prio**	**d-prio**	**vsfr-prio**	**rcf-prio**	**multi-prio**
random	0.99999	-	-	-	-	-	-
gfr-prio	**5.7e-14**	**9.6e-14**	-	-	-	-	-
d-prio	0.00479	0.00156	**7.3e-06**	-	-	-	-
vsfr-prio	**8.4e-14**	**6.2e-14**	0.99992	**7.8e-07**	-	-	-
rcf-prio	**<2e-16**	**<2e-16**	**0.21527**	**8.6e-13**	0.44757	-	-
multi-prio	**<2e-16**	**<2e-16**	**1.3e-09**	**<2e-16**	**1.8e-08**	**0.00094**	-
multi-eq-prio	**<2e-16**	**<2e-16**	**5.3e-05**	**7.1e-14**	**0.00035**	0.29068	0.60221

Table 5.4: The results after running post-hoc analysis with a Nemenyi test on the experiment with 93 prioritized test activities (based on the full test history). Significant differences were found for 20 out of 28 pairs of prioritization methods. A cell value marked in bold indicates that there was a significant difference ($p < 0.001$) for the pair of prioritization methods that intersects at that cell.

Experiment Run with Limited Regression History

The second part of the experiment was conducted in which the 83 test activities were prioritized at their sixth baseline version, by each of the prioritization methods. Similar to before, figure 5.5 visualizes the $APFD_C$ values gathered at all 83 test activities for each prioritization method, figure 5.6 shows the same $APFD_C$ values

visualized in a box plot, and table 5.5 shows the average $APFD_C$ and APTTF values in a tabular format.

The $APFD_C$ values gathered during the experiment for each of the prioritization methods (see appendix A.2) were analyzed using a Friedman test, which showed that there was a significant difference ($\chi^2 = 394.43$, $p = 8.488^{-82} < 0.001$) between the $APFD_C$ values depending on the prioritization method, when comparing *some* pair of prioritization methods. Post-hoc analysis found significant differences for 18 out of 28 pairs of prioritization methods (see table 5.6). There are no significant differences between INITIAL & RANDOM, INITIAL & D-PRIO, RANDOM & D-PRIO, GFR-PRIO & RCF-PRIO, GFR-PRIO & MULTI-PRIO, GFR-PRIO & MULTI-EQ-PRIO, D-PRIO & VSFR-PRIO, RCF-PRIO & MULTI-PRIO, RCF-PRIO & MULTI-EQ-PRIO, or MULTI-PRIO & MULTI-EQ-PRIO. Once again, post-hoc analysis shows that there is a significant difference between INITIAL & MULTI-PRIO as well as RANDOM & MULTI-PRIO, even though prioritization is done with a limited regression test history. The $APFD_C$ values gathered during the experiment for the MULTI-PRIO method can be seen in figure 5.7, where the $APFD_C$ values for the INITIAL and MULTI-PRIO method are also visualized as baselines for comparison. Similar graphs can be seen for the other prioritization methods in the appendix. Table 5.5 shows that the MULTI-EQ-PRIO method achieves the highest average $APFD_C$ value (0.9507) out of all the compared prioritization methods, as well as the lowest APTTF value (0.0966). MULTI-PRIO performs very similarly.

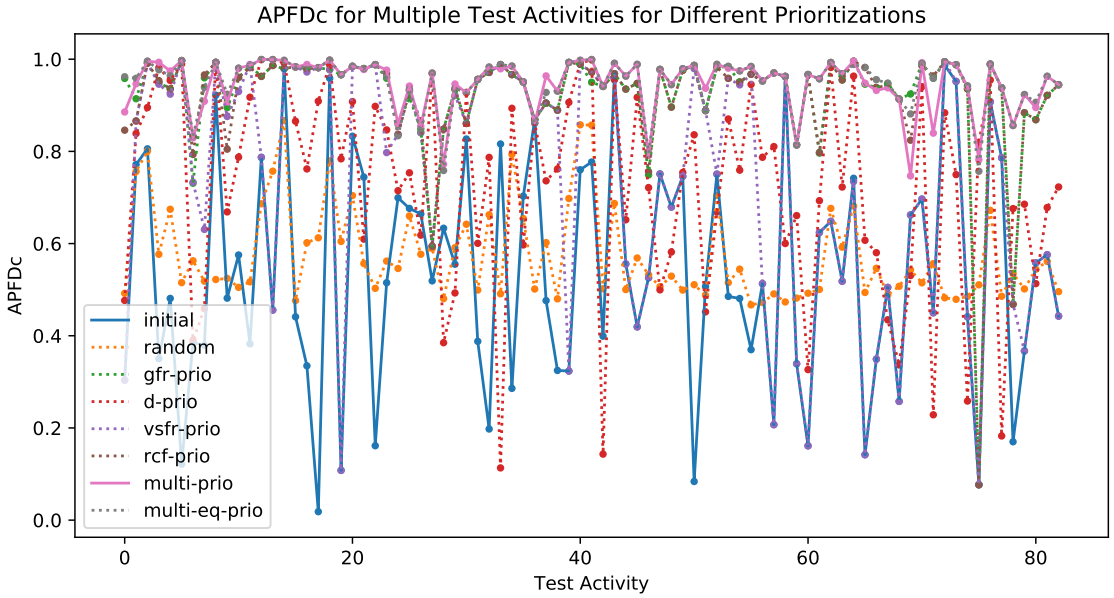

Figure 5.5: Graph showing the $APFD_C$ values for each of the 83 prioritized test activities (based on the limited test history) for each of the prioritization methods. The solid purple line corresponds to the MULTI-PRIO method, and the solid blue line corresponds to the INITIAL method. Each dot corresponds to a prioritized test activity, hence there are eight vertical dots for every test activity on the x-axis.

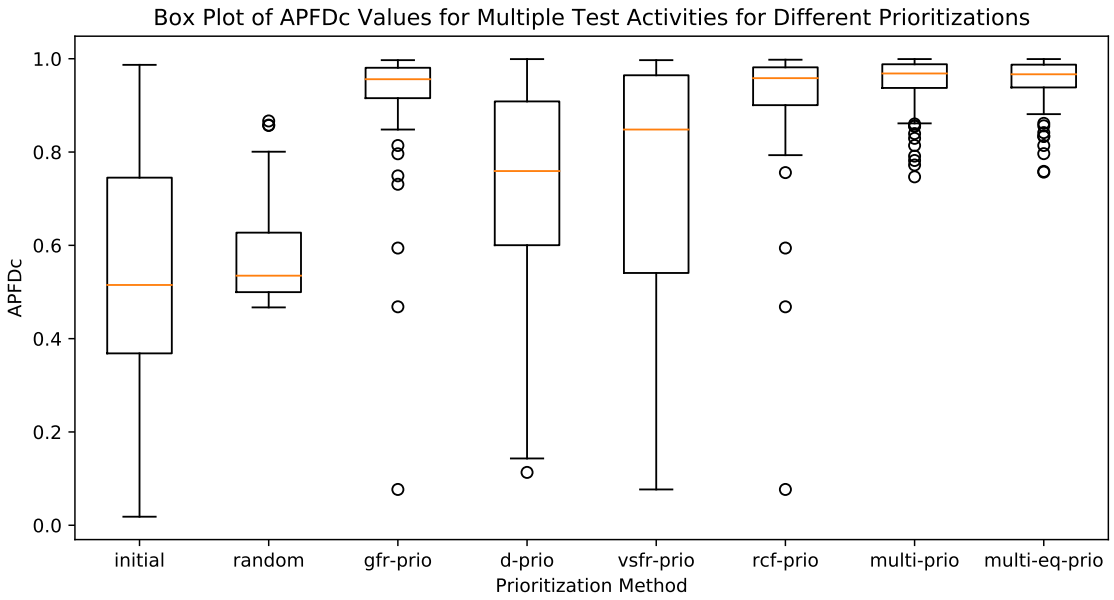

Figure 5.6: Box plot visualizing the $APFD_C$ values gathered during the experiment on the 83 prioritized test activities (based on the limited test history) for each of the prioritization methods.

	Prioritization Method	Average APFDc	Average APTTF
0	initial	0.5368	0.5516
1	random	0.5744	0.5211
2	gfr-prio	0.9213	0.1209
3	d-prio	0.7241	0.3723
4	vsfr-prio	0.7374	0.3234
5	rcf-prio	0.9189	0.1315
6	multi-prio	0.9475	0.1051
7	multi-eq-prio	0.9507	0.0966

Table 5.5: The average $APFD_C$ values gathered during the experiment for the different prioritization methods (based on the limited test history). The MULTI-EQ-PRIO method has the highest $APFD_C$ value at 0.9507 (and the lowest APTTF value), followed by MULTI-PRIO at 0.9475 (which also has the second smallest APTTF value). The lowest $APFD_C$ values are achieved by INITIAL and RANDOM, at 0.5368 and 0.5744 respectively.

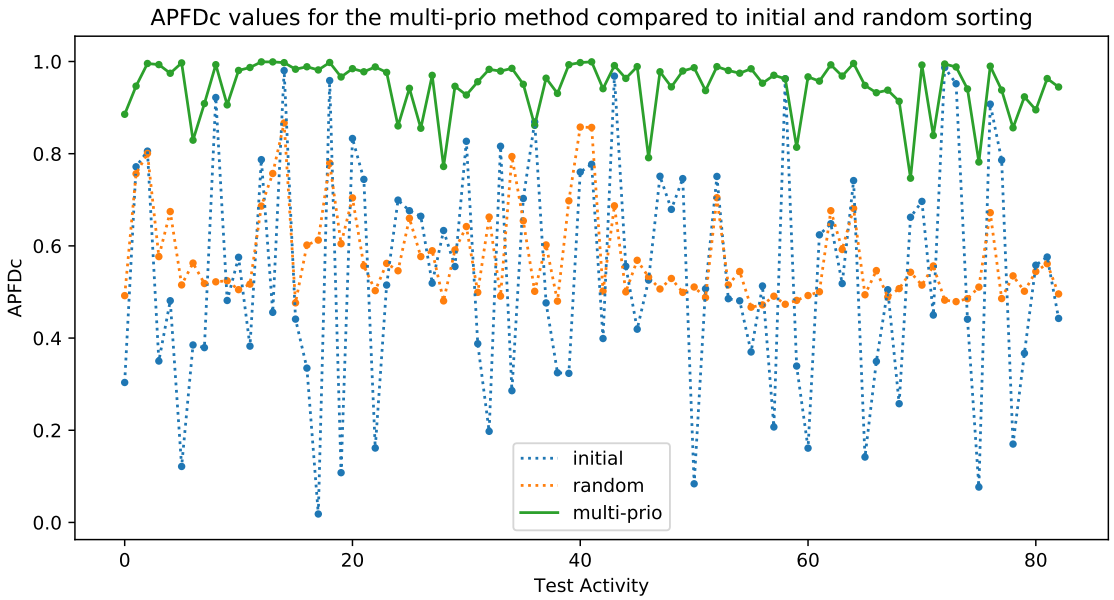

Figure 5.7: Graph showing the $APFD_C$ values for each of the 83 prioritized test activities (based on the limited test history) for three different prioritization methods. The solid green line corresponds to the MULTI-PRIO method, the dashed blue line corresponds to the INITIAL method, and the dashed orange line corresponds to the RANDOM method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

	initial	**random**	**gfr-prio**	**d-prio**	**vsfr-prio**	**rcf-prio**	**multi-prio**
random	1.00000	-	-	-	-	-	-
gfr-prio	**8.9e-14**	**9.1e-14**	-	-	-	-	-
d-prio	0.01692	0.01881	**1.3e-07**	-	-	-	-
vsfr-prio	**0.00014**	**0.00016**	**8.0e-05**	0.93553	-	-	-
rcf-prio	**7.2e-14**	**7.4e-14**	1.00000	**3.6e-08**	**2.9e-05**	-	-
multi-prio	**<2e-16**	**<2e-16**	0.04407	**5.6e-14**	**3.1e-13**	0.07926	-
multi-eq-prio	**<2e-16**	**<2e-16**	0.03637	**5.4e-14**	**2.0e-13**	0.06656	1.00000

Table 5.6: The results after running post-hoc analysis with a Nemenyi test on the experiment with 83 prioritized test activities (based on the limited test history). Significant differences were found for 18 out of 28 pairs of prioritization methods. A cell value marked in bold indicates that there was a significant difference ($p < 0.001$) for the pair of prioritization methods that intersect at that cell.

5.2.2 Expert-Review Evaluation

A total of 10 experts were interviewed for the expert-review evaluations. An overview of the participants can be seen in table 5.7, and their quantitative answers are summarized in table 5.8.

Expert	Expert Role / Area of Expertise	Experience (Years)
1	SPAS Simulation Development (Virtual Verification)	7
2	Function Verification & Data Strategies (Method and Tools)	18
3	System Tester (ADAS/ASDM)	2.5
4	Product Owner (Active Safety HIL)	4.5
5*	Product Owner (Former Manager) (Continuous Integration & Automation)	8
6	System Architect (Continuous Integration, TAAS, VeriDaG)	11
7	System Architect (Continuous Integration, TAAS, VeriDaG)	10
8	System Architect (Verification & Validation of AD Functionality)	5
9	Technical Expert (Active Safety Functions)	20
10	Senior Analysis Engineer (Autonomous Drive)	12

Table 5.7: The experts who were interviewed during the expert-review evaluations, with their corresponding roles or areas of expertise, as well as their levels of experience. All interviewees except for *Expert 5* had previously been interviewed in the focus group interviews that were conducted during the pre-study.

Person	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
1	4	4	4	4	4	3	4	3	4	3
2	4	4	3	3	4	3	4	3	4	3
3	4	4	3	2	3	4	4	4	5	4
4	4	4	3	3	3	4	4	3	5	3
5*	5	4	4	2	4	3	4	4	5	5
6*	5	4	2	3	4	3	4	4	5	5
7*	4	4	2	2	4	2	4	4	5	4
8	4	4	4	3	3	3	3	4	4	4
9	4	4	3	4	4	4	3	3	4	4
10	4	4	2	3	4	2	4	4	5	4
Mean	4.2	4	3	2.9	3.7	3.1	3.8	3.6	4.6	3.9
Median	4	4	3	3	4	3	4	4	5	4
Mode	4	4	3	3	4	3	4	4	5	4

Table 5.8: The answers from the expert-review evaluations, where the 5-point Likert scale has been converted to numerical values: *1 = Strongly Disagree, 2 = Agree, 3 = Don't know, 4 = Agree, 5 = Strongly Agree.* As can be seen in the table, the median and mode of all questions are equal. These provide an appropriate indication of the typical participant answered for each question, as the Likert-scale is an ordinal scale. Participant 5, 6, and 7 have supervised the thesis. Participant 6 and 7 have worked closely with VeriDaG.

Q1 and Q2: Overall, the participants seemed to believe that the proposed method prioritizes in an efficient way and that it considers valuable factors. No participant stated that they believe it sorts in an inefficient way, or that the method considers factors that are not valuable. Some of them asked how I came to the conclusion to select these particular factors. Many participants were engaged and came with ideas on potential factors that could be interesting to consider. These are mentioned below, in the findings for question 3.

Q3: This was probably the question that engaged the participants the most.
For future work, some participants suggested that some sort of measure of criticality or severity could be investigated. Either by historically investigating the severities of the faults that each test case has detected, or by assigning a criticality to the test case itself, indicating how important the test case is (i.e. how serious it is if the test case results in a failure). Their argument was that failures can have different severities, such that some critical test case is never allowed to fail, while some other less important test case might be more or less ignored if it results in a failure.

Some participants suggested that, for future work, the currently used factor `D` could be made a bit more complex (perhaps discrete). They mentioned a concept of "cost per time" depending on the type of test activity (stimulation, simulation etc.). Essentially, they suggested adding some way of incorporating the varying costs that are related to the test environments that the tests are being run on. On a related note, one participant mentioned that the level of fidelity of test case often affects the duration – a very detailed test might require more time to run, but might also find

important faults that are otherwise hard to detect. Another aspect related to time, is that simulations can run faster than reality, but that tests that are run on HIL rigs need to be run in real-time. Thus, simulation tests might be easier to optimize with respect to time, than compared to HIL tests.

One participant suggested adding a factor for avoiding error-prone or unstable tests, that might be due to problems in the environment that the test case is being run on, or for other reasons, such as intermittent behavior in the test case itself.

Q4 and Q5: Most participants did not know if the data that is used by the prioritization method is available at VCC as of today. Some mentioned that enough data is not present as of today, at least not at all departments. A majority of the participants believed that the data will be available within a near future, and no participant stated the opposite.

Q6 and Q7: These questions are related to Q4 and Q5, which can also be seen in the answers from the participants. Some participants mentioned that it likely would be possible to implement the prioritization method today, given that VCC starts gathering the data that is used by the different factors. One participant mentioned that it depends on the time that is needed to implement the database containing the historical test results. Looking forward, almost all participants believed it would be possible to implement the method within the near future, and no participant stated the opposite. Some participants mentioned that there currently exists no framework where the method could be integrated, but that the TAAS system is currently under development, where this method or some similar method will be implemented shortly.

Q8: A majority of the participants believed that the data from VeriDaG seems realistic, while some participants were unsure due to not knowing enough about the system, for instance due to working outside the department that is developing VeriDaG. Two of the participants (6 and 7) have been working more closely with VeriDaG than the others. No participant stated that they believe that the data seems unrealistic.

Q9 and Q10: The participants were confident in that the method could help improve the testing efficiency at VCC. A majority of the participants "strongly agreed" with the statement while the rest "agreed". Some participants mentioned that the method could help them save a lot of time, and that it would be particularly useful as the number of test cases grow, as well as at domain level, where there are many test cases in general. One participant mentioned that it likely would not make a significant improvement in short-term bench testing, but that it definitely would help for longer-running tests. One participant mentioned another benefit of utilizing the method, where the "score" that is assigned by the prioritization method (i.e. `V(t,v)`) could be used to identify test cases with a low value, so that they either could be removed or updated to improve general testing efficiency. Most participants also believed that the proposed method could help improve testing efficiency at other companies than VCC working within the autonomous drive domain.

6
Discussion

6.1 Implications for Research Questions

Three research questions were defined at the start of the thesis (see 1). This section aims discuss what the results presented in section 5 tell us about them.

6.1.1 RQ1 — Research Question 1

Recall RQ1: *To what degree are existing test case prioritization methods applicable to the autonomous drive domain?*.

The results of the pre-study show that deliveries are often delivered from external suppliers in binary format. This holds true for VCC, but research suggests that this phenomenon is common within the automotive domain in general. One disadvantage of this is that many existing TCP techniques cannot be applied within this domain, due to them often requiring access to the source code of the SUT. Thus, in the case of VCC (and many other automotive companies), almost all prioritization methods that require access to the source code have to be discarded, leaving us with TCP techniques that are based on non-code-based objectives (i.e. white-box TCP techniques), such as history-based TCP techniques.

One thing that makes autonomous drive verification a bit special is the wide variety of test methods and testing levels, where aspects such as costs and execution times can vary a lot. It seems reasonable to argue that cost-cognizant TCP techniques are generally desired within this domain, thus making non-cost-cognizant techniques less appropriate. One could also argue that multi-objective TCP techniques are preferred, probably also in the general case, but especially within this domain due to same reasons as above – a high variance in costs and execution times, but also due to varying severity of faults and test case criticality.

The different testing techniques utilized within the autonomous drive domain are also appropriate for different types of testing. A complete car contains a lot of functionality that can be tested in different ways and on different levels. Identifying what changed in the system can make it possible to prioritize appropriate test cases and test methods, that are best suited to test the area that was affected by that particular change. This is where the concept of version-specific TCP techniques comes into the picture. Due to deliveries often being in binary format (and often from external suppliers), the information that is available to perform version-specific prioritization is highly restricted. For this reason are most existing version-specific

TCP techniques not applicable to the autonomous drive verification domain, due to them operating at a level of granularity that is too low (i.e. code-level). However, research has shown that operating on a higher level of granularity can still provide valuable results. In this thesis, the level of granularity was treated on what can be seen as changed software components in the version of the SUT, instead of changed lines of code, which is more commonly used. Data at this level of granularity is also not reliant upon suppliers manually attaching meta-data about changes in their deliveries, but instead, it can typically be extracted automatically from the deliveries. This is positive as it removes the dependency between suppliers and testers, which reduces the risk of problems due to the human factor. Black-box and gray-box version-specific techniques operating on higher granularity levels similar to the example in this thesis (file-level) are likely more applicable within the context of autonomous drive verification.

6.1.2 RQ2 — Research Question 2

Recall RQ2: *What existing method, or combination of methods, gives a sufficiently good test case prioritization?*.

No existing method could be found in previous work that matches all our requirements (cost-cognizant, version-specific, multi-objective, black-box). Instead, a method called MULTI-PRIO was developed, that used a combination of existing factors and new factors, inspired by existing techniques. These factors were also treated as their own prioritization methods in this thesis, to see how well they performed on their own. Thus, does none of these single-factor methods meet our initial requirements, but they might still be useful despite that.

With reference to "sufficiently good": strictly following APFD_C, significant results are found for most of the prioritization methods when compared to the initial and random prioritization. The only prioritization method that does not result in a significant difference (at significance level $p < 0.001$) is D-PRIO, which only considers the historical duration of the test cases. This is not surprising, as it completely ignores aspects of how well the test case performs when it comes to detecting faults.

Prioritizing by considering historical fault-detecting abilities (in this thesis referred to as GFR-PRIO) is a technique (or very similar technique) that has been used extensively by others, and can therefore be seen as an existing method that provides a sufficiently good test case prioritization, at least when considering APFD_C values and the hypothesis testing that was performed during the experiment. With this said, it could very likely help improve the testing efficiency at VCC and similar companies – but there is still room for a lot of improvement. When considering other aspects such as best practices and suitability within the autonomous drive domain, GFR-PRIO on its own would perhaps not be considered as a sufficiently good method. This all comes down to how one chooses to define "sufficiently good".

VSFR-PRIO is harder to define when it comes to seeing it as a combination of methods or a novel approach. VSFR-PRIO is certainly inspired by existing techniques, but it might be hard to specify that it is a definitive combination of some particular set of functions. But as it combines two main ideas from existing methods, one

could consider it as a combination of methods. The results from the experiment show that it performs very similarly to the GFR-PRIO technique when the whole regression test history is considered, but that it has a bigger variance in the case of the limited regression test history. This is likely because the correlation between changed RELEASABLEs and failing test cases becomes more reliable over time. The GFR-PRIO technique is a bit more general (and likely less precise) in the sense that it does not require as much specific information as the VSFR-PRIO technique, which can be useful when little historical data is known. Solely based on the results in this thesis (e.g. average $APFD_C$ values and post-hoc tests) it is hard to conclude if the VSFR-PRIO technique provides an advantage over the GFR-PRIO technique, when compared to each other. However, when combined they seem to provide an advantage in the MULTI-PRIO method.

RCF-PRIO is probably the method that is most unique. There are certain existing techniques that integrate the aspect of time since failure in the fault-detecting abilities, and treat recent failures as more important than failures that occurred a long time ago. However, `RCF` is a separate metric and its main focus is to find test cases that resulted in a failure at the absolute last test execution. It can almost be seen as a Boolean flag, stating whether the test case failed at the last execution or not, combined with a multiplier rewarding longer failure streaks (up to 3). For this reason, I'd not consider this an existing method. Based on the results from the experiment, the RCF-PRIO method (`RCF` factor) achieves the highest average $APFD_C$ value out of all single factors in the experiment with the full regression test history. When evaluated in the experiment with the limited regression test history it also performs well, but the GFR-PRIO achieves a slightly higher score in this case. It makes sense that the GFR-PRIO technique performs similar to the RCF-PRIO technique when a limited test history is available, as most failures are considered recent failures when the history is very short.

6.1.3 RQ3 — Research Question 3

Recall RQ3: *What new method gives a sufficiently good test case prioritization, and what data does it require?*.

The proposed method MULTI-PRIO, and its equally weighted variant MULTI-EQ-PRIO), achieve the highest average $APFD_C$ values in both experiments (i.e. considering the full regression test history and a limited regression test history). This indicates that the proposed method is robust in the sense that is not too reliant upon the amount of historical data that is available. It also suggests that the method performs almost equally well without finding the optimal set of weights using hyper-parameter optimization (this is also indicated by the post-hoc tests, where no significant difference is found between MULTI-PRIO & MULTI-EQ-PRIO. It might however be a bit dangerous to generalize this conclusion just based on this data, as the set of equal weights could just happen to be rather suitable in this case. Performing the optimization of the weights in the reverse direction, i.e. finding the set of weights that results in the worst average APTTF, and then comparing it against the optimized version, MULTI-PRIO, might provide better insights into how much the choice of weights can affect the outcome, in the worst and best scenarios.

When doing this, the worst set of weights was $(0, 100, 0, 0)$ which effectively is the D-PRIO method. The worst set of valid weights (such that each factor is still used), is $(5, 85, 5, 5)$, which achieves an average APFD_C of 0.7607, which is quite far from the optimal at 0.9103, but still better than the INITIAL and RANDOM methods.

The optimal set of weights after doing hyper-parameter optimization is $(5, 45, 5, 45)$. As can be seen, the weights are quite different for the different factors. The reason that factors `D` and `RCF` got the highest weights is probably because duration (`D`) is the most different one when compared to the other factors, which are more focused on different types of fault-detecting abilities. One could think of it as two groups of factors, where `D` belongs to the first group, and `GFR`, `VSFR`, and `RCF` belong to the second group. My theory is that the combination of two seemingly uncorrelated factors (i.e. factors from different groups) gives the most overall information, than compared to the combination of two almost similar factors. RCF-PRIO (`RCF`) was the single factor that achieved the highest average APFD_C, thus it's not surprising that `D` and `RCF` are weighted higher than the other factors. Worth noting is that while two weights were equal to 5, no weight was equal to 0, which indicates that all factors contributed to a better prioritization. Whether or not this improvement is worth adding two extra factors (when compared to $(0, 50, 0, 50)$ depends on aspects such as time and cost of implementing the different factors. Reasoning about the different weights is interesting, but one could also argue that the particular set of weights that is selected is not that important, as they can be found automatically through hyper-parameter optimization. This removes the need of manually trying to find weights that are suitable. This, together with the results from the experiment showing that MULTI-PRIO and MULTI-EQ-PRIO performed very similarly, were the reasons for choosing to present MULTI-EQ-PRIO in the table showing an example sorting during the expert-review evaluations, to avoid focusing too much on the particular set of weights, but instead, focus on the different factors. The fact that the optimal set of weights happened to be $(5, 45, 5, 45)$ in this particular case might not actually tell us that much. The optimal set of weights will likely also change over time, and depends on the data that is available in the context in which the method is applied. For the sake of investigating whether or not all factors contributed to an overall better prioritization, the weights were allowed to be 0 and 100 in this experiment. In practice, the weights might be bound to be within some suitable range (e.g. $[15, 35]$), to ensure that no factor is too small or too big, if this seems appropriate in the context where this method is applied.

Looking at the box plots from the experiments, one can see that the proposed method has a relatively low variance when compared to the other prioritization methods (especially the INITIAL method). This indicates that the method performs relatively consistently, regardless of factors such as the type of the test activity, the number of test cases in in the test activity, or the amount of historical data for the test activity. One of the reasons for constructing the experiment in this particular way (i.e. with different test activity types, varying amounts of test cases and historical data) was to introduce some difference in the type of prioritizations that were to be done, to mimic reality as much as possible, and to build a method that could handle a variety of different prioritization challenges. The RANDOM method has a seemingly low variance, but this is because the recorded APFD_C value at every

test activity for that method was the average of 100 random prioritizations of that particular test activity.

With respect to time, prioritization only takes about second on an average laptop. The time is however dependent on the amount of data and the implementation, but in orders of magnitude of time, prioritization is quick. The time required to find the optimal set of weights is also done in a reasonable amount of time, but depends on the number of tested combinations. It, however, seems unlikely that this would be a restricting factor, especially since the weights do not need to be optimized at every run. Random search has also proved to be a more time-efficient hyper-parameter optimization technique, if this would ever become a concern.

To summarize, findings from the experiments show that the proposed method performs well, with the highest average $APFD_C$ values out of the tested methods. When considering APTTF, the proposed method also achieves the lowest averages. Comparing the APTTF of INITIAL at 0.5476 (55%), to the APTTF of MULTI-PRIO at 0.1359 (14%), we can see that failures occur much earlier on average, which enables quicker feedback in the continuous integration chain, given that the tested version of the SUT results in a failure. Statistical significance has been found between INITIAL & MULTI-PRIO as well as RANDOM & MULTI-PRIO, at the rather strict significant level of 0.001. The method also seems robust against differences in the amount of historical data that is available. The factors it considers can be expected to be available almost anywhere, which increases the general applicability of the method. Findings from the expert-review evaluations indicate that the experts are confident in that the method could help improve the testing efficiency at VCC, and very likely at other similar companies as well.

Four different factors are considered by the proposed method: `GFR`, `D`, `VSFR`, and `RCF`. `GFR` requires access to the number of failures and number of runs for each test case. `D` requires access to the historical duration of each test case. `VSFR` requires access to historical data about changed software components in the version of the SUT that was being tested, as well as the verdicts of the test cases. `RCF` requires access to historical verdicts of test cases.

In short, the factors of the proposed method requires the following data to be recorded for each tested version of the SUT:

- The verdict of each test case
- The duration of each test case
- The changed software components

These are arguably reasonable factors that can be expected to be available in almost any continuous integration environment. Results from the expert-review evaluations show that the participants believe that the data will be available, and that it will be possible to implement the method, within a near future at VCC. The applicability and general availability of the factors were some of the reasons that they were selected for the proposed method. This can be compared to another scenario in which the theoretical data (e.g. ModifiedFeatureAreas and ReleasableFocuses) that is available in VeriDaG had been included in the proposed method. Including those factors might have provided even better results (i.e. higher $APFD_C$ values

etc.), but it would have made the proposed method a lot less useful and applicable in practice.

Optimizing the set of weights can also be done using this data, thanks to the APTTF metric, as it only requires historical test case verdicts and durations. If one would prefer optimizing the weights using $APFD_C$, then knowledge of present faults in the SUT would also be required (which often is achieved through fault seeding).

6.2 Threats to Validity

This section aims to discuss weak points of the thesis and what was done to counter them.

A majority of TCP techniques, including the one proposed in this thesis, often assumes that test cases are independent, which might not always be true in reality [47]. VCC is aware of this assumption, and their plan is to detect these dependencies in TAAS, meaning that, for VCC (and perhaps other companies), this is not necessarily something that the prioritization method should be responsible for.

The experiment conducted in this thesis was evaluated in a simulated setting made to resemble the real world. While experimental simulations provide many benefits, the results from such settings generally also have a lower level of realism, as modelling an accurate representation of reality is very hard. With this said, VeriDaG seems to take many important concepts into account that contribute to a higher level of realism, as presented in section 2.5. VCC also reports that the data model has been inspected by several people with heavy experience in autonomous drive verification, and that no major flaws have been detected. Several experts at VCC have stressed that it is necessary to start investigating ways of prioritizing now (i.e. on simulated data), instead of starting doing it first when the real system is in place, as it then would be too late. VeriDaG arguably seems to take relevant entities and concepts from reality into consideration, but the realism of aspects such as underlying probabilities of events in the simulation might be harder to assess. However, during evaluation of the proposed method, experts at VCC argued that potential issues related to such parameter tuning is expected to have a low impact on the proposed method, apart from possibly changing the optimal set of weights. In order not to draw conclusions solely from the promising experiment results, expert-review evaluations were conducted to see if domain experts at VCC believed that the proposed method could improve testing efficiency at VCC, albeit only being evaluated on simulated data thus far. Results from the expert-review evaluations show that they were confident in that it could help improve the testing efficiency at VCC, and probably also at other similar companies.

The APTTF metric has to our knowledge never been used before, which could make it a bit unreliable to use when making strong conclusions. In this thesis, however, the APTTF metric is only used as a secondary metric, and is included more as a bonus. Almost all quantitative conclusions are instead based on the $APFD_C$ metric. The $APFD_C$ metric has, on the other hand, been used extensively in research, and could arguably be seen as a renowned and reliable metric.

This thesis was done at a case company. One disadvantage of case studies is that they generally result in findings with a lower generalizability. To increase generalizability, the proposed method was developed to only use data that is expected to be generally available, independent of the company where the method is applied. As mentioned in earlier sections, the factors of the method are based on data that is expected to be available almost anywhere. Literature also indicates that many companies within the automotive or autonomous drive domain have a lot in common with the case company, VCC, for instance with regards to a wide range of testing activities, external suppliers, and binary deliveries.

6.3 Future Work

There are many interesting routes and promising possibilities that could be explored for future work. This section mentions some of them.

One interesting aspect would be to explore the concept of severity, and how it could be integrated into the proposed method. Some ideas, that were also mentioned during the expert-review evaluations, would be to assign severities either to test cases or to the faults they manage to find. The goal of this would be to prioritize important test cases (i.e. that test critical functionality), or to prioritize test cases that have a history of detecting serious faults. The APFD_C metric has support for varying fault severities, and would likely still be a useful metric for evaluation.

Another interesting concept is stability, and investigating how this could be integrated to the proposed method. This was also pointed out as a possible future improvement during the expert-review evaluations. One starting point is to calculate it similar to how `GFR` is calculated: $\frac{\text{number of errors}}{\text{number of executions}}$ per test case, if the verdict *Error* indicates that the test case failed for some external reason (e.g. test environment failure). Another approach could be to repeat all tests multiple times for a test activity, to see which test cases result in different verdicts, even though nothing in the SUT changed.

A measure of test fidelity could also be explored and integrated to the proposed method, to grade how advanced or detailed the test case is. For instance, simulations can be very simple or extremely detailed. In some cases tests of higher fidelity might be needed, at the cost of longer execution times. If a test of low fidelity fails, it's a good indication that it also does not work in reality, as failing tests of high fidelity might be due to issues arising from the complexity (fidelity) of the test itself. One approach might be to have a grading scheme, where a fidelity level is assigned per test case or per test activity type. Somewhat related to test fidelity, a concept of cost per time could be explored, to account for differences in costs (e.g. test environment costs) related to different test activities, where the goal would be to prioritize testing that is done using less expensive methods.

As mentioned in the description of VeriDaG, there exists some theoretical future data (ModifiedFeatureAreas and ReleasableFocuses). It would be interesting to see how applicable these are in reality. Would it for instance be possible to tag every test case with some meta-data about what features it is supposed to test? The

current approach used in the proposed method is to reverse-engineer what could be seen as the ReleasableFocus of each test case, using the correlation that is present in the `VSFR` factor. It would be interesting to compare the two approaches to each other, to see which one performs the best, and which one is more applicable.

There exist different approaches for sorting values by their averages. `GFR` is sorting test cases by their average failure detecting abilities. It would, however, prioritize $\frac{Failures(t,v)}{v} = \frac{1}{1}$ and $\frac{Failures(t,v)}{v} = \frac{100}{100}$ equally, while one could argue that the second case probably is more reliable, as the failures have been consistent, while the first case might just be due to randomness. To improve the `GFR` metric, another way of sorting that takes the number of observations (in this case the number of executions) into account, such as Bayesian rating, might be considered.

It would also be interesting to evaluate the proposed method on real data that is not simulated, either at VCC as soon as it gets available, or within some other company working within this domain, where the data might already be available.

This thesis introduces a novel APTTF metric, designed for use within continuous integration. One possible route for future work would be to explore its applicability. According to me, it seems usable as a complement or an alternative to the APFD_C metric, when knowledge about existing faults in the SUT might not be available. I also believe it is easier to understand what the value of the APTTF metric means (e.g. $APTTF = 0.21$ means that test case failures, on average, occur after 21% of the total test time has passed). For this reason, it could be interesting to investigate it as a key performance indicator (KPI) for measuring the efficiency of continuous integration testing activities.

Finally, a general observation is that it would be interesting to investigate the possibilities of black-box and gray-box version-specific TCP techniques that operate on a higher fidelity (e.g. file-level instead of code-level), as this would make version-specific prioritization more applicable to the automotive domain.

7
Conclusion

The autonomous drive verification domain introduces various challenges related to TCP. The common use of binary deliveries is one of the reasons that make many existing TCP techniques inappropriate. Luckily, there are TCP techniques that are applicable to the domain, such as history-based techniques.

VCC and other automotive companies are currently undergoing a transformation towards continuous integration and higher automation of autonomous drive verification activities. Early feedback is essential within continuous integration, which increases the need for TCP. One way of preparing a TCP strategy before this transformation is complete, is to utilize and evaluate prioritization methods against a simulated regression test history until the real data is in place. According to some, this approach may even be necessary, despite a risk of reduced generalizability, as it otherwise would be too late.

A TCP technique within this domain should preferably be cost-cognizant, version-specific, multi-objective, and black-box. Best to our knowledge, no method that all those properties currently exists. By combining elements from existing methods, a proposed method with these properties was constructed. Results from experiments and expert-review evaluations indicate that the proposed method can help improve testing efficiency at VCC and likely also at similar companies.

There are many interesting possibilities for the future, for instance with regard to introducing the concept of test case criticality and digging deeper into version-specific black-box TCP techniques.

Bibliography

[1] J. McKay and P. Marshall, "The dual imperatives of action research," *Information Technology & People*, vol. 14, no. 1, pp. 46–59, 2001.

[2] E. Engström, P. Runeson, and A. Ljung, "Improving regression testing transparency and efficiency with history-based prioritization - An industrial case study," *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST 2011*, pp. 367–376, 2011.

[3] S. Roongruang and J. Daengdej, "A Test Case Prioritization Method with Practical Weight Factors," *Journal of Software Engineering*, vol. 4, no. 3, pp. 193–214, 2010.

[4] A. Dadwal, H. Washizaki, Y. Fukazawa, T. Iida, M. Mizoguchi, and K. Yoshimura, "Prioritization in automotive software testing: Systematic literature review," in *CEUR Workshop Proceedings*, vol. 2273, pp. 52–58, 2018.

[5] D. Sundmark, K. Petersen, and S. Larsson, "An exploratory case study of testing in an automotive electrical system release process," in *SIES 2011 - 6th IEEE International Symposium on Industrial Embedded Systems, Conference Proceedings*, pp. 166–175, 2011.

[6] N. Juristo, A. M. Moreno, and S. Vegas, "Reviewing 25 Years of Testing Technique Experiments," *Empirical Software Engineering*, vol. 9, no. 1, pp. 7–44, 2004.

[7] R. Hamlet, "Theoretical Comparison of testing methods," *ACM SIGSOFT Software Engineering Notes*, vol. 14, no. 8, pp. 28–37, 2004.

[8] S. Eldh, H. Hansson, S. Punnekkat, A. Pettersson, and D. Sundmark, "A framework for comparing efficiency, effectiveness and applicability of software testing techniques," *Proceedings - Testing: Academic and Industrial Conference - Practice and Research Techniques, TAIC PART 2006*, pp. 159–170, 2006.

[9] T. E. Vos, B. Marínt, M. J. Escalona, and A. Marchetto, "A methodological framework for evaluating software testing techniques and tools," *Proceedings - International Conference on Quality Software*, pp. 230–239, 2012.

[10] A. Pretschner, M. Broy, I. H. Krüger, and T. Stauner, "Software engineering for automotive systems: A roadmap," *FoSE 2007: Future of Software Engineering*, pp. 55–71, 2007.

[11] A. Michailidis, U. Spieth, T. Ringler, B. Hedenetz, and S. Kowalewski, "Test front loading in early stages of automotive software development based on AUTOSAR," *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pp. 435–440, 2013.

[12] D. Marijan, A. Gotlieb, and S. Sen, "Test Case Prioritization for Continuous Regression Testing: An Industrial Case Study," in *IEEE International Conference on Software Maintenance, ICSM*, pp. 540–543, IEEE, 2013.

[13] T. Muthusamy and D. K. Seetharaman, "A Test Case Prioritization Method with Weight Factors in Regression Testing Based on Measurement Metrics," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 12, pp. 390–396, 2013.

[14] S. Elbaum, G. Rothermel, and M. J. Harrold, "Prioritizing Test Cases for Regression Testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[15] X. Wang and H. Zeng, "Dynamic Test Case Prioritization based on Multi-objective," in *2014 IEEE/ACIS 15th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2014 - Proceedings*, pp. 1–6, IEEE, 2014.

[16] S. Yoo, R. Nilsson, and M. Harman, "Faster Fault Finding at Google Using Multi Objective Regression Test Optimisation," in *FSE'11*, pp. 1–4, 2011.

[17] C. Malz and P. Göhner, "Agent-Based Test Case Prioritization," in *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011*, pp. 149–152, 2011.

[18] T. Noguchi, H. Washizaki, Y. Fukazawa, A. Sato, and K. Ota, "History-Based Test Case Prioritization for Black Box Testing Using Ant Colony Optimization," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 - Proceedings*, pp. 1–2, IEEE, 2015.

[19] Y. Fazlalizadeh, A. Khalilian, M. Abdollahi Azgomi, and S. Parsa, "Prioritizing Test Cases for Resource Constraint Environments Using Historical Test Case Performance Data," in *Proceedings - 2009 2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2009*, pp. 190–195, IEEE, 2009.

[20] R. Lachmann, M. Nieke, C. Seidl, I. Schaefer, and S. Schulze, "System-level test case prioritization using machine learning," in *Proceedings - 2016 15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016*, pp. 361–368, 2017.

[21] B. Qu, C. Nie, B. Xu, and X. Zhang, "Test Case Prioritization for Black Box Testing," in *Proceedings - International Computer Software and Applications Conference*, vol. 1, pp. 465–472, IEEE, 2007.

[22] V. Bachmann and R. Messnarz, "Improving the software development for multiple projects by applying a platform strategy for mechatronic systems," *Journal of software: Evolution and Process*, vol. 24, pp. 541–549, 2012.

[23] D. Cotroneo, R. Pietrantuono, and S. Russo, "Testing techniques selection based on ODC fault types and software metrics," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1613–1637, 2013.

[24] K. Grimm, "Software technology in an automotive company - major challenges," *25th International Conference on Software Engineering, 2003. Proceedings.*, vol. 6, pp. 498–503, 2004.

[25] C. Malz, N. Jazdi, and P. Gohner, "Prioritization of Test Cases using Software Agents and Fuzzy Logic," in *Proceedings - IEEE 5th International Confer-*

ence on Software Testing, Verification and Validation, ICST 2012, pp. 483–486, 2012.

[26] R. Lachmann and I. Schaefer, "Towards Efficient and Effective Testing in Automotive Software Development," in *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)*, vol. P-232, pp. 2181–2192, Technische Universität Braunschweig, 2014.

[27] M. Khatibsyarbini, M. A. Isa, D. N. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," 2018.

[28] R. Lachmann, *Black-Box Test Case Selection and Prioritization for Software Variants and Versions*. PhD thesis, Braunschweig University of Technology, Germany, 2017.

[29] M. Harman, "Making the case for MORTO: Multi objective regression test optimization," in *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011*, pp. 111–114, IEEE, 2011.

[30] R. Patton, *Software Testing*. Sams Publishing, 2001.

[31] P. M. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Signature Series, Pearson Education, 2007.

[32] R. Mukherjee and K. S. Patnaik, "A survey on different approaches for software test case prioritization," *Journal of King Saud University - Computer and Information Sciences*, 2018.

[33] A. G. Malishevsky, J. R. Ruthruff, G. Rothermel, and S. Elbaum, "Cost-cognizant Test Case Prioritization," tech. rep., University of Nebraska-Lincoln, Lincoln, Nebraska, U.S.A., 2006.

[34] H. Park, H. Ryu, and J. Baik, "Historical Value-Based Approach for Cost-cognizant Test Case Prioritization to Improve the Effectiveness of Regression Testing," in *Proceedings - The 2nd IEEE International Conference on Secure System Integration and Reliability Improvement, SSIRI 2008*, pp. 39–46, IEEE, 2008.

[35] E. D. Ekelund, *Test Selection Based on Historical Test Data*. PhD thesis, Lund University, 2015.

[36] D. Leffingwell and D. Widrig, *Managing Software Requirements: A Use Case Approach*. Pearson Education, 2 ed., 2003.

[37] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[38] A. Srivastava and J. Thiagarajan, "Effectively Prioritizing Tests in Development Environment," 2002.

[39] J. Zheng, B. Robinson, L. Williams, and K. Smiley, "A lightweight process for change identification and regression test selection in using COTS components," *Proceedings - Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems*, vol. 2006, pp. 137–143, 2006.

[40] S. Biswas, R. Mall, M. Satpathy, and S. Sukumaran, "Regression test selection techniques: A survey," *Informatica (Ljubljana)*, vol. 35, no. 3, pp. 289–321, 2011.

[41] E. Bringmann and A. Krämer, "Model-based testing of automotive systems," *Proceedings of the 1st International Conference on Software Testing, Verification and Validation, ICST 2008*, pp. 485–493, 2008.

[42] P. Koopman and M. Wagner, "Challenges in Autonomous Vehicle Testing and Validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 2016–01–0128, 2016.

[43] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars," 2017.

[44] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors (Switzerland)*, vol. 19, no. 3, 2019.

[45] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[46] R. A. Krueger, "Designing and Conducting Focus Group Interviews," tech. rep., University of Minnesota, 2002.

[47] W. Lam, D. Jalali, K. Muşlu, D. Notkin, M. D. Ernst, J. Wuttke, and S. Zhang, "Empirically Revisiting the Test Independence Assumption," *ISSTA 2014 Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 385–396, 2014.

A

Appendix 1

A.1 Experiment Based on Full Regression History

A.1.1 APFDc Graphs

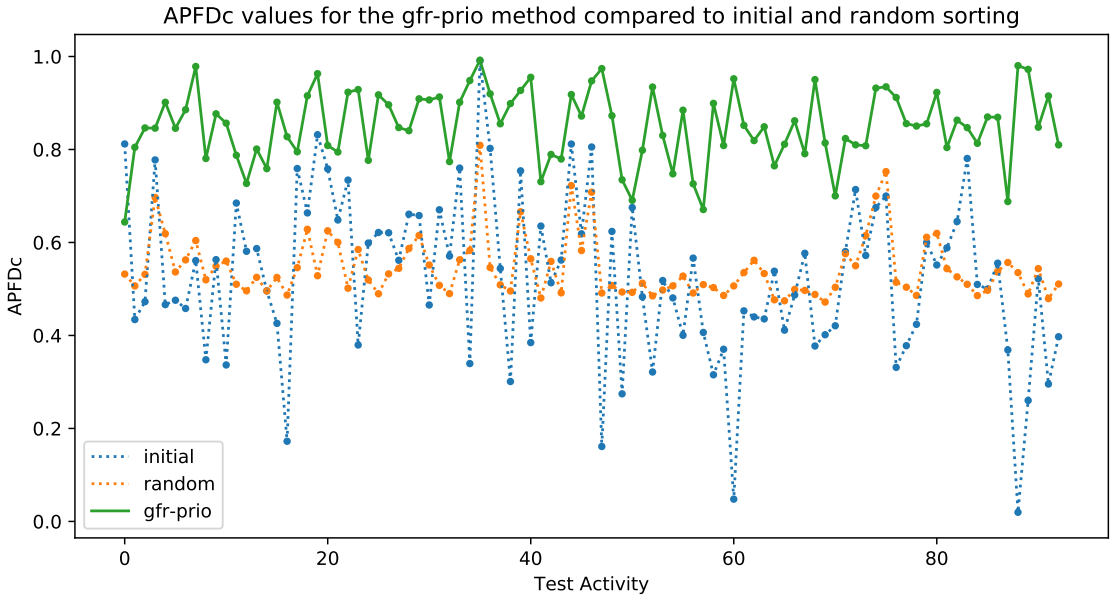

Figure A.1: Graph showing the APFD_C values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the `gfr-prio` method, the dashed blue line corresponds to the `initial` method, and the dashed orange line corresponds to the `random` method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

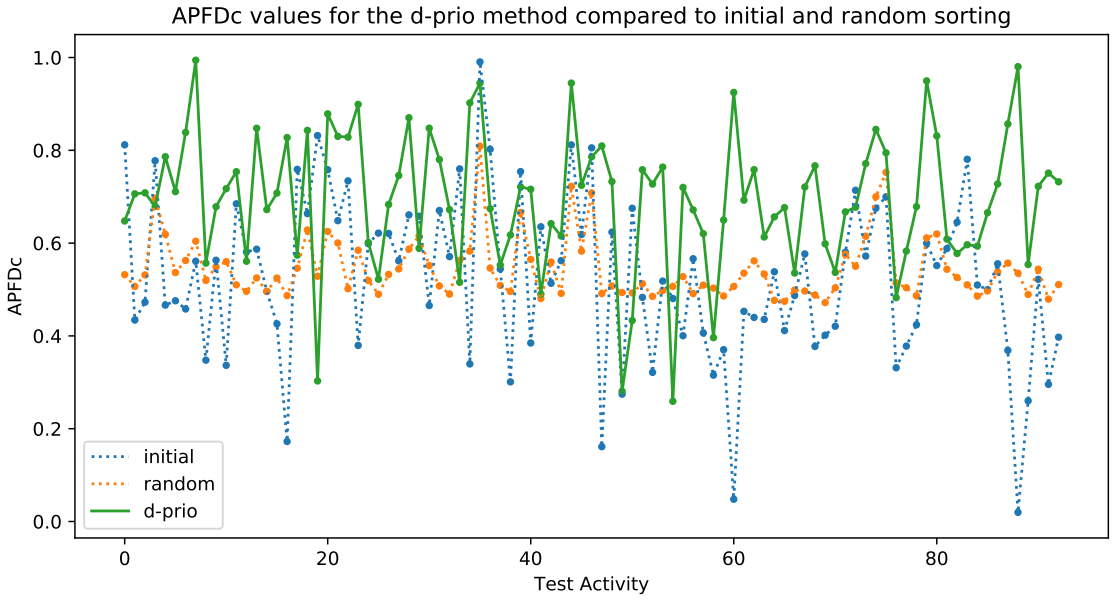

Figure A.2: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the `d-prio` method, the dashed blue line corresponds to the `initial` method, and the dashed orange line corresponds to the `random` method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

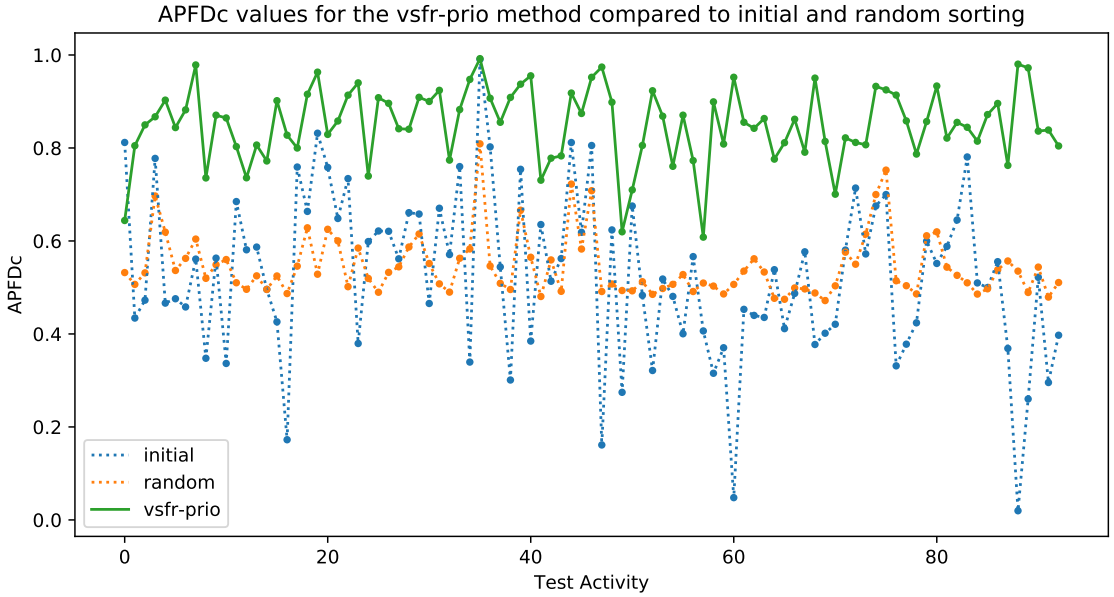

Figure A.3: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the `vsfr-prio` method, the dashed blue line corresponds to the `initial` method, and the dashed orange line corresponds to the `random` method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

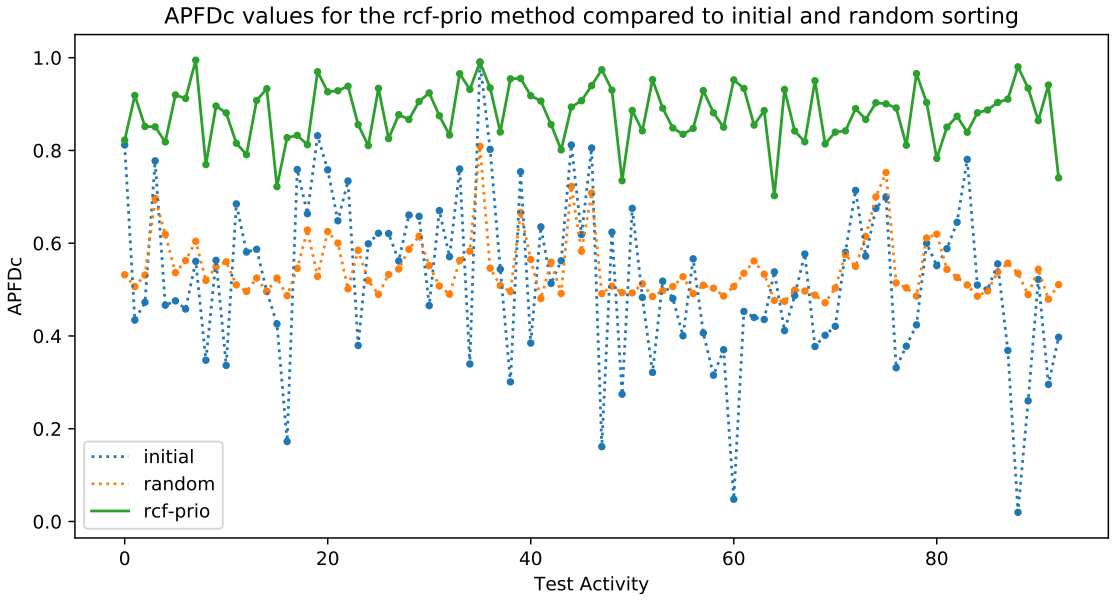

Figure A.4: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the `rcf-prio` method, the dashed blue line corresponds to the `initial` method, and the dashed orange line corresponds to the `random` method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

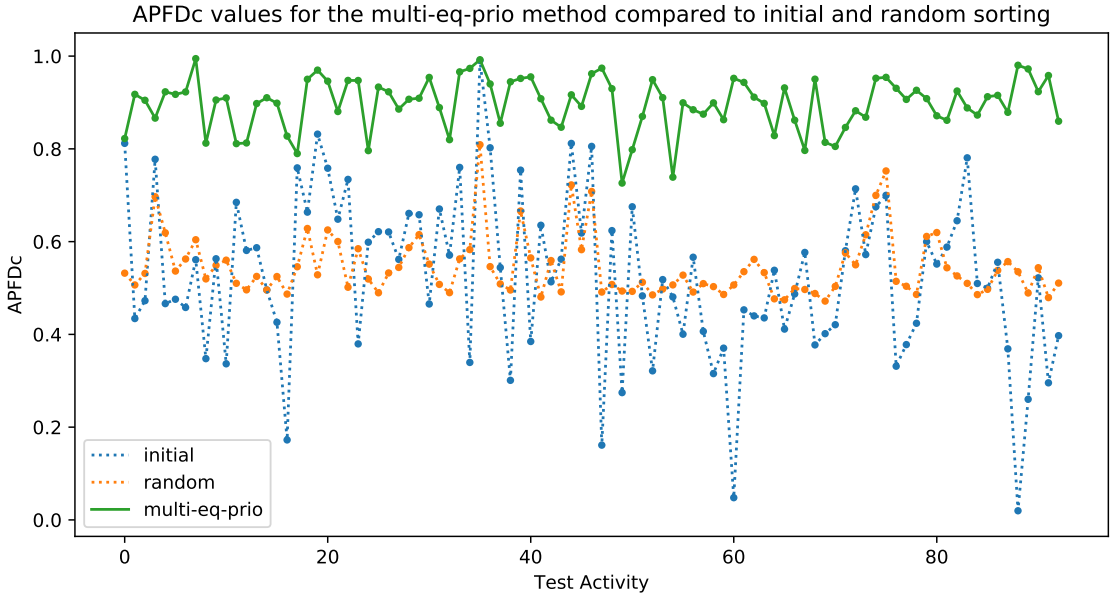

Figure A.5: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the `multi-eq-prio` method, the dashed blue line corresponds to the `initial` method, and the dashed orange line corresponds to the `random` method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

A.1.2 APFDc Values

	Baseline	Version	Activity	Tests	initial	random	gfr-prio	d-prio	vsfr-prio	rcf-prio	multi-prio	multi-eq-prio
0	#2	#2505	#1	8	0.8119	0.5320	0.6440	0.6477	0.6440	0.8223	0.8223	0.8223
1	#2	#2505	#2	113	0.4342	0.5066	0.8048	0.7062	0.8048	0.9185	0.9424	0.9176
2	#3	#274	#4	27	0.4727	0.5314	0.8463	0.7083	0.8497	0.8517	0.9128	0.9048
3	#4	#39	#8	36	0.7776	0.6967	0.8459	0.6793	0.8671	0.8509	0.8051	0.8663
4	#5	#77	#9	76	0.4663	0.6190	0.9015	0.7863	0.9029	0.8183	0.9234	0.9230
5	#6	#75	#10	33	0.4757	0.5365	0.8456	0.7110	0.8439	0.9197	0.9487	0.9175
6	#6	#75	#11	50	0.4580	0.5625	0.8855	0.8385	0.8820	0.9120	0.9517	0.9228
7	#6	#75	#12	21	0.5610	0.6041	0.9786	0.9942	0.9786	0.9945	0.9945	0.9945
8	#6	#75	#13	17	0.3477	0.5196	0.7806	0.5569	0.7356	0.7694	0.8231	0.8123
9	#7	#86	#14	49	0.5631	0.5489	0.8767	0.6784	0.8705	0.8959	0.8969	0.9054
10	#7	#86	#15	46	0.3366	0.5600	0.8565	0.7173	0.8645	0.8809	0.9366	0.9098
11	#7	#86	#17	11	0.6847	0.5101	0.7874	0.7539	0.8029	0.8155	0.8039	0.8113
12	#8	#783	#19	49	0.5810	0.4960	0.7270	0.5609	0.7359	0.7909	0.8463	0.8128
13	#8	#783	#20	45	0.5869	0.5250	0.8010	0.8476	0.8063	0.9078	0.9336	0.8977
14	#8	#783	#21	49	0.4959	0.4958	0.7588	0.6722	0.7721	0.9332	0.9407	0.9103
15	#9	#115	#22	19	0.4261	0.5247	0.9017	0.7078	0.9017	0.7221	0.7763	0.8985
16	#9	#115	#23	2	0.1725	0.4869	0.8275	0.8275	0.8275	0.8275	0.8275	0.8275
17	#9	#115	#24	32	0.7590	0.5458	0.7951	0.5740	0.7999	0.8325	0.7807	0.7898
18	#10	#86	#25	40	0.6636	0.6283	0.9156	0.8430	0.9156	0.8124	0.9294	0.9501
19	#10	#86	#26	25	0.8318	0.5285	0.9631	0.3029	0.9631	0.9698	0.9698	0.9698
20	#11	#146	#28	46	0.7583	0.6251	0.8086	0.8786	0.8292	0.9264	0.9775	0.9460
21	#11	#146	#29	9	0.6484	0.6003	0.7945	0.8301	0.8581	0.9287	0.9287	0.8805
22	#12	#70	#31	34	0.7342	0.5016	0.9232	0.8284	0.9136	0.9386	0.9653	0.9475
23	#13	#139	#32	35	0.3795	0.5847	0.9289	0.8991	0.9400	0.8555	0.9600	0.9474
24	#15	#155	#34	25	0.5987	0.5194	0.7767	0.6014	0.7396	0.8105	0.8226	0.7962
25	#17	#173	#36	67	0.6215	0.4897	0.9175	0.5221	0.9082	0.9339	0.9460	0.9333
26	#17	#173	#37	27	0.6208	0.5324	0.8961	0.6834	0.8961	0.8255	0.9293	0.9230
27	#17	#173	#39	73	0.5616	0.5444	0.8471	0.7457	0.8413	0.8772	0.9045	0.8858
28	#18	#76	#42	19	0.6608	0.5870	0.8403	0.8702	0.8403	0.8667	0.9246	0.9070
29	#18	#76	#43	44	0.6579	0.6151	0.9090	0.5888	0.9090	0.9055	0.8904	0.9092
30	#19	#128	#45	44	0.4655	0.5515	0.9066	0.8477	0.9000	0.9241	0.9698	0.9540
31	#20	#109	#49	18	0.6704	0.5077	0.9129	0.7800	0.9240	0.8748	0.9056	0.8890
32	#21	#99	#50	23	0.5708	0.4900	0.7739	0.6722	0.7739	0.8331	0.8626	0.8198
33	#21	#99	#51	46	0.7600	0.5627	0.9015	0.5156	0.8826	0.9654	0.9518	0.9661
34	#22	#112	#55	25	0.3395	0.5828	0.9482	0.9022	0.9474	0.9316	0.9745	0.9735
35	#23	#38	#56	36	0.9906	0.8089	0.9921	0.9448	0.9921	0.9906	0.9784	0.9921
36	#24	#764	#59	42	0.8024	0.5461	0.9198	0.6742	0.9070	0.9349	0.9469	0.9399
37	#24	#764	#60	13	0.5441	0.5088	0.8552	0.5528	0.8552	0.8397	0.8847	0.8552
38	#24	#764	#61	29	0.3009	0.4956	0.8987	0.6177	0.9087	0.9544	0.9725	0.9446
39	#25	#105	#63	81	0.7541	0.6661	0.9270	0.7212	0.9372	0.9554	0.9665	0.9519
40	#25	#105	#64	21	0.3847	0.5649	0.9552	0.7161	0.9552	0.9180	0.9552	0.9552
41	#26	#200	#67	18	0.6353	0.4806	0.7307	0.4901	0.7307	0.9065	0.9120	0.9081
42	#27	#35	#71	38	0.5133	0.5592	0.7893	0.6420	0.7780	0.8559	0.8888	0.8619
43	#27	#35	#72	18	0.5622	0.4916	0.7795	0.6153	0.7830	0.8012	0.8709	0.8466
44	#29	#92	#74	43	0.8117	0.7223	0.9181	0.9449	0.9181	0.8936	0.9508	0.9166
45	#30	#84	#75	38	0.6186	0.5827	0.8718	0.7246	0.8742	0.9072	0.8911	0.8916
46	#30	#84	#78	64	0.8053	0.7081	0.9475	0.7860	0.9520	0.9398	0.9373	0.9620
47	#32	#113	#82	17	0.1612	0.4913	0.9740	0.8094	0.9740	0.9740	0.9740	0.9740
48	#32	#113	#83	23	0.6237	0.5075	0.8726	0.7327	0.8984	0.9300	0.9522	0.9297
49	#33	#76	#84	6	0.2744	0.4934	0.7348	0.2807	0.6197	0.7348	0.7262	0.7262
50	#34	#2555	#85	14	0.6751	0.4927	0.6907	0.4333	0.7098	0.8861	0.8962	0.7982
51	#34	#2555	#86	29	0.4830	0.5124	0.7984	0.7581	0.8054	0.8422	0.9031	0.8700
52	#35	#257	#87	19	0.3214	0.4852	0.9343	0.7272	0.9231	0.9528	0.9641	0.9492
53	#35	#257	#88	43	0.5181	0.4977	0.8302	0.7639	0.8683	0.8911	0.9249	0.9104
54	#35	#257	#90	33	0.4807	0.5067	0.7476	0.2590	0.7605	0.8484	0.7531	0.7390
55	#37	#720	#93	14	0.4004	0.5278	0.8845	0.7200	0.8705	0.8348	0.9001	0.8993
56	#37	#720	#94	40	0.5664	0.4913	0.7261	0.6716	0.7729	0.8475	0.9152	0.8844
57	#38	#788	#95	32	0.4065	0.5095	0.6708	0.6203	0.6082	0.9291	0.9346	0.8748
58	#38	#788	#96	14	0.3154	0.5030	0.8992	0.3963	0.8992	0.8813	0.9106	0.8992
59	#38	#788	#97	17	0.3703	0.4860	0.8085	0.6497	0.8085	0.8501	0.9223	0.8629
60	#39	#2508	#100	7	0.0479	0.5065	0.9521	0.9248	0.9521	0.9521	0.9521	0.9521
61	#40	#264	#101	42	0.4529	0.5352	0.8518	0.6926	0.8553	0.9336	0.9577	0.9433
62	#41	#769	#102	47	0.4399	0.5617	0.8191	0.7583	0.8423	0.8547	0.9222	0.9115
63	#41	#769	#104	70	0.4355	0.5332	0.8489	0.6131	0.8633	0.8860	0.9190	0.8980
64	#41	#769	#105	15	0.5382	0.4766	0.7648	0.6566	0.7760	0.7025	0.8386	0.8286

	Baseline	Version	Activity	Tests	initial	random	gfr-prio	d-prio	vsfr-prio	rcf-prio	multi-prio	multi-eq-prio
65	#42	#715	#106	14	0.4114	0.4746	0.8113	0.6765	0.8113	0.9315	0.9577	0.9315
66	#42	#715	#107	13	0.4871	0.4989	0.8618	0.5354	0.8618	0.8419	0.8914	0.8618
67	#43	#765	#110	11	0.5767	0.4968	0.7909	0.7210	0.7909	0.8187	0.8187	0.7966
68	#44	#2474	#111	9	0.3773	0.4881	0.9503	0.7667	0.9503	0.9503	0.9503	0.9503
69	#44	#2474	#112	3	0.4015	0.4718	0.8141	0.5985	0.8141	0.8141	0.8141	0.8141
70	#44	#2474	#113	14	0.4208	0.5035	0.7002	0.5370	0.7006	0.8395	0.8413	0.8052
71	#45	#246	#114	74	0.5804	0.5759	0.8236	0.6673	0.8217	0.8422	0.8666	0.8462
72	#45	#246	#116	34	0.7138	0.5500	0.8100	0.6774	0.8119	0.8903	0.9057	0.8822
73	#45	#246	#117	37	0.5721	0.6148	0.8081	0.7714	0.8069	0.8668	0.9225	0.8682
74	#46	#56	#118	44	0.6752	0.6998	0.9320	0.8451	0.9325	0.9029	0.9563	0.9523
75	#46	#56	#119	43	0.6998	0.7524	0.9348	0.7946	0.9249	0.9004	0.9509	0.9542
76	#47	#773	#120	47	0.3314	0.5144	0.9118	0.4825	0.9138	0.8916	0.9181	0.9306
77	#47	#773	#122	39	0.3780	0.5037	0.8556	0.5828	0.8582	0.8111	0.9165	0.9066
78	#48	#171	#125	26	0.4240	0.4861	0.8504	0.6786	0.7869	0.9657	0.9657	0.9263
79	#49	#74	#126	43	0.5998	0.6112	0.8553	0.9497	0.8569	0.9034	0.9471	0.9083
80	#49	#74	#127	43	0.5516	0.6196	0.9228	0.8311	0.9333	0.7828	0.8571	0.8714
81	#49	#74	#128	43	0.5883	0.5436	0.8042	0.6086	0.8212	0.8503	0.8766	0.8616
82	#51	#794	#134	48	0.6450	0.5260	0.8631	0.5776	0.8551	0.8737	0.9075	0.9247
83	#52	#2529	#138	49	0.7807	0.5100	0.8470	0.5965	0.8445	0.8392	0.8848	0.8883
84	#52	#2529	#139	44	0.5097	0.4857	0.8132	0.5935	0.8146	0.8812	0.9036	0.8729
85	#52	#2529	#140	45	0.4997	0.4971	0.8699	0.6655	0.8718	0.8872	0.9266	0.9124
86	#53	#266	#144	41	0.5555	0.5379	0.8693	0.7275	0.8957	0.9035	0.9156	0.9154
87	#53	#266	#145	31	0.3688	0.5569	0.6880	0.8568	0.7625	0.9106	0.9439	0.8788
88	#54	#77	#147	6	0.0197	0.5351	0.9803	0.9803	0.9803	0.9803	0.9803	0.9803
89	#56	#723	#149	22	0.2602	0.4892	0.9723	0.5538	0.9723	0.9342	0.9723	0.9723
90	#56	#723	#150	48	0.5220	0.5438	0.8479	0.7224	0.8363	0.8643	0.9385	0.9233
91	#57	#797	#151	45	0.2956	0.4793	0.9151	0.7509	0.8387	0.9410	0.9660	0.9582
92	#57	#797	#152	35	0.3973	0.5106	0.8099	0.7322	0.8043	0.7409	0.8593	0.8596

Table A.1: The $APFD_C$ values gathered during the experiment for the different prioritization methods. A total of 93 test activities with a total of 3157 test cases were prioritized and evaluated at their corresponding latest baseline version (based on the full test history)

A.2 Experiment Based on Limited Regression History

A.2.1 APFDc Graphs (Limited Regression History)

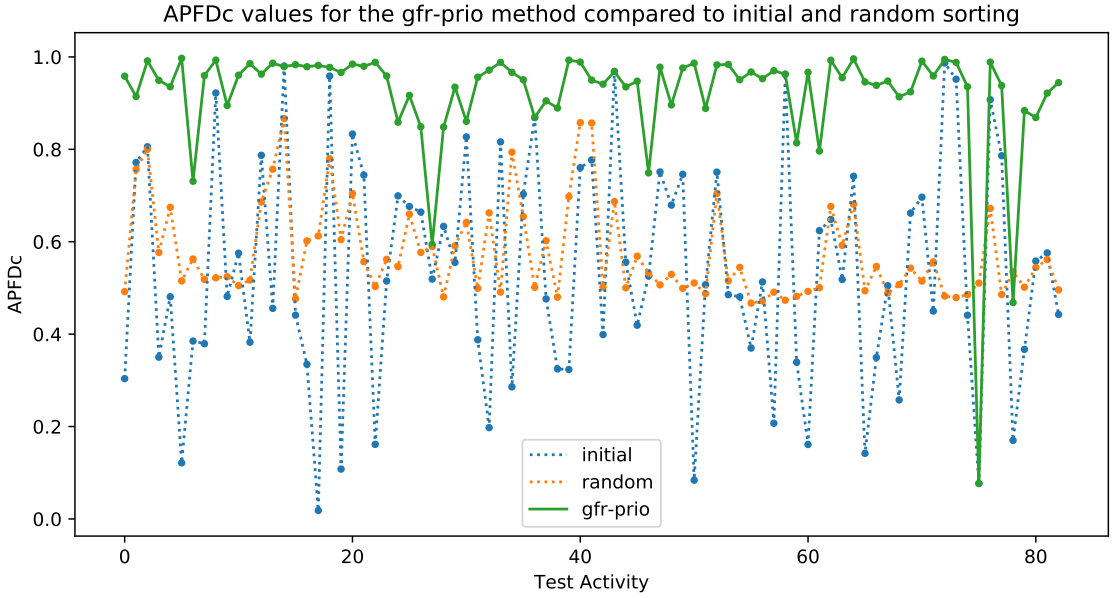

Figure A.6: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the `gfr-prio` method, the dashed blue line corresponds to the `initial` method, and the dashed orange line corresponds to the `random` method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

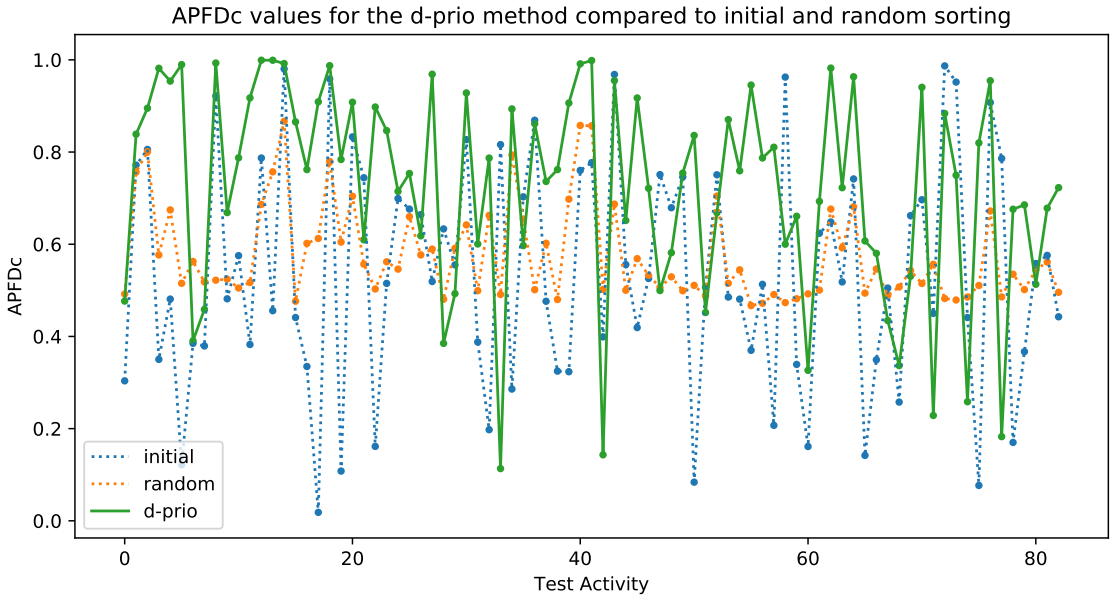

Figure A.7: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the `d-prio` method, the dashed blue line corresponds to the `initial` method, and the dashed orange line corresponds to the `random` method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

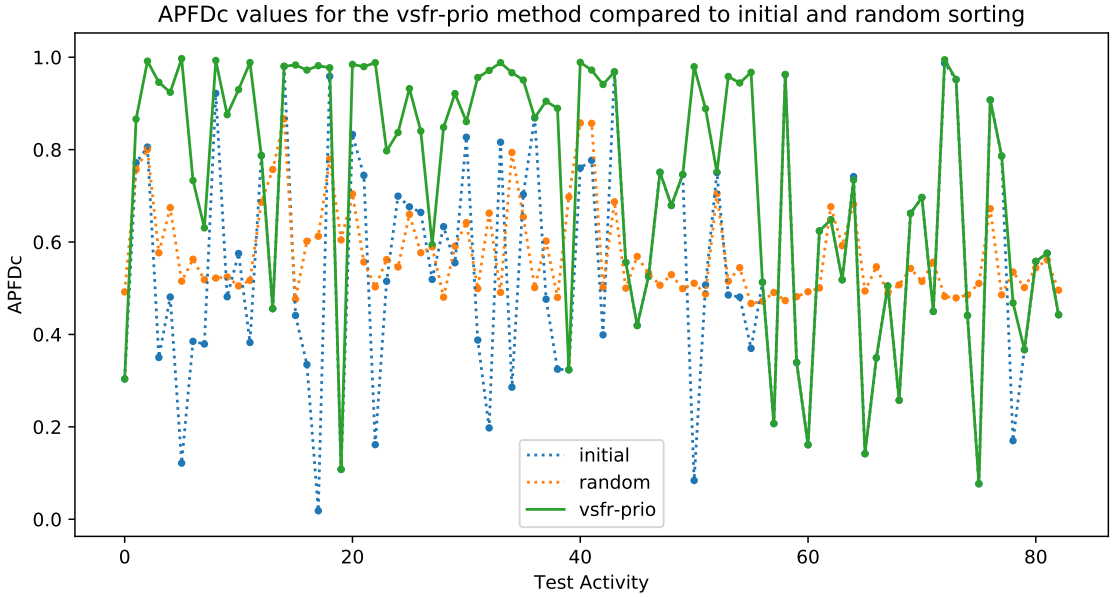

Figure A.8: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the `vsfr-prio` method, the dashed blue line corresponds to the `initial` method, and the dashed orange line corresponds to the `random` method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

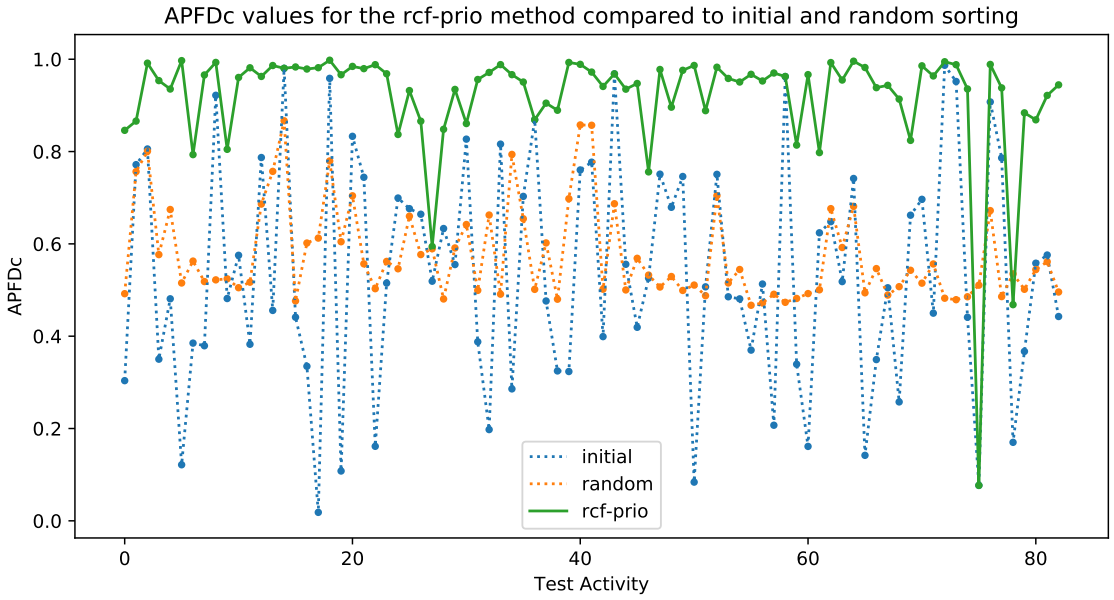

Figure A.9: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the `rcf-prio` method, the dashed blue line corresponds to the `initial` method, and the dashed orange line corresponds to the `random` method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

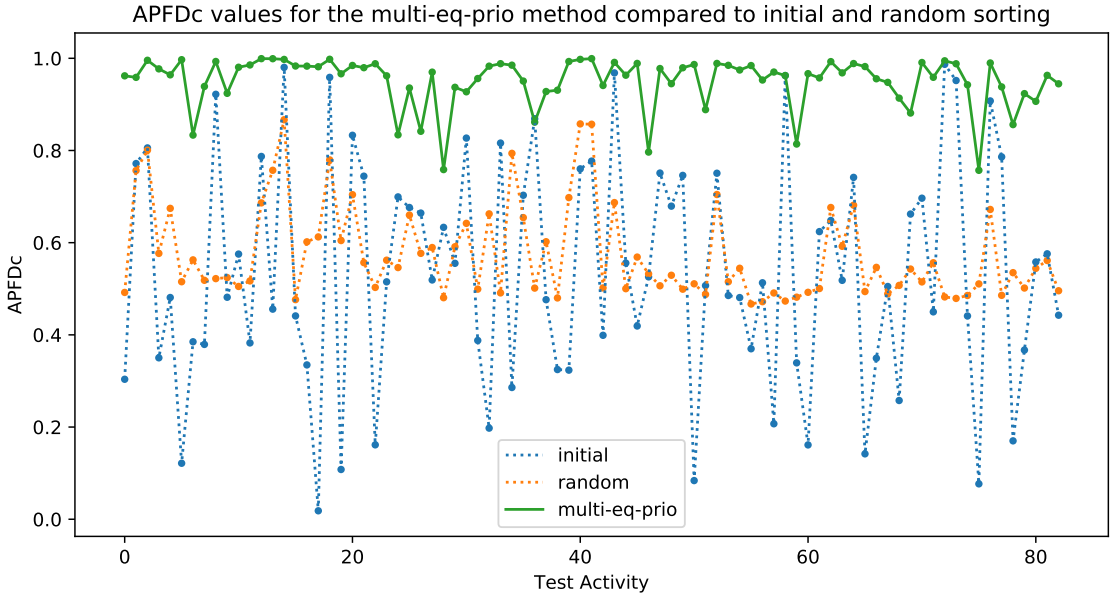

Figure A.10: Graph showing the $APFD_C$ values for each of the 93 prioritized test activities (based on the full test history) for three different prioritization methods. The solid green line corresponds to the `multi-eq-prio` method, the dashed blue line corresponds to the `initial` method, and the dashed orange line corresponds to the `random` method. Each dot corresponds to a prioritized test activity, hence there are three vertical dots for every test activity on the x-axis.

A.2.2 APFDc Values (Limited Regression History)

	Baseline	Version	Activity	Tests	initial	random	gfr-prio	d-prio	vsfr-prio	rcf-prio	multi-prio	multi-eq-prio
0	#2	#6	#2	113	0.3037	0.4921	0.9583	0.4766	0.3037	0.8460	0.8855	0.9620
1	#4	#6	#8	36	0.7716	0.7565	0.9142	0.8388	0.8660	0.8660	0.9466	0.9586
2	#5	#6	#9	76	0.8056	0.8007	0.9915	0.8951	0.9915	0.9915	0.9957	0.9957
3	#6	#6	#10	33	0.3502	0.5767	0.9494	0.9817	0.9457	0.9537	0.9933	0.9770
4	#6	#6	#11	50	0.4810	0.6745	0.9354	0.9539	0.9242	0.9354	0.9745	0.9641
5	#6	#6	#12	21	0.1215	0.5153	0.9970	0.9897	0.9970	0.9970	0.9970	0.9970
6	#6	#6	#13	17	0.3850	0.5627	0.7309	0.3905	0.7332	0.7933	0.8293	0.8334
7	#7	#6	#15	46	0.3793	0.5183	0.9596	0.4587	0.6307	0.9659	0.9087	0.9389
8	#7	#6	#17	11	0.9219	0.5220	0.9931	0.9931	0.9931	0.9931	0.9931	0.9931
9	#8	#6	#19	49	0.4816	0.5243	0.8948	0.6687	0.8757	0.8048	0.9059	0.9240
10	#8	#6	#20	45	0.5752	0.5050	0.9605	0.7875	0.9299	0.9605	0.9806	0.9806
11	#8	#6	#21	49	0.3825	0.5170	0.9855	0.9175	0.9885	0.9815	0.9870	0.9855
12	#9	#6	#22	19	0.7871	0.6863	0.9627	0.9992	0.7871	0.9627	0.9992	0.9992
13	#9	#6	#24	32	0.4558	0.7570	0.9864	0.9991	0.4558	0.9864	0.9991	0.9991
14	#10	#6	#25	40	0.9804	0.8666	0.9791	0.9920	0.9804	0.9804	0.9976	0.9976
15	#10	#6	#26	25	0.4411	0.4764	0.9832	0.8655	0.9832	0.9832	0.9832	0.9832
16	#11	#6	#28	46	0.3348	0.6018	0.9788	0.7619	0.9722	0.9788	0.9886	0.9830
17	#11	#6	#29	9	0.0183	0.6125	0.9817	0.9090	0.9817	0.9817	0.9817	0.9817
18	#12	#6	#31	34	0.9586	0.7796	0.9773	0.9874	0.9773	0.9979	0.9979	0.9979
19	#13	#6	#32	35	0.1079	0.6046	0.9664	0.7839	0.1079	0.9664	0.9664	0.9664
20	#15	#6	#34	25	0.8330	0.7041	0.9845	0.9081	0.9845	0.9845	0.9845	0.9845
21	#17	#6	#36	67	0.7443	0.5567	0.9796	0.6096	0.9796	0.9796	0.9777	0.9796
22	#17	#6	#37	27	0.1614	0.5029	0.9882	0.8977	0.9882	0.9882	0.9882	0.9882
23	#17	#6	#39	73	0.5151	0.5620	0.9586	0.8465	0.7973	0.9685	0.9763	0.9620
24	#18	#6	#42	19	0.6993	0.5460	0.8587	0.7147	0.8370	0.8370	0.8602	0.8340
25	#18	#6	#43	44	0.6763	0.6599	0.9167	0.7535	0.9321	0.9321	0.9418	0.9356
26	#19	#6	#45	44	0.6642	0.5768	0.8488	0.6183	0.8402	0.8658	0.8552	0.8417
27	#20	#6	#49	18	0.5191	0.5895	0.5942	0.9688	0.5942	0.5942	0.9700	0.9700
28	#21	#6	#50	23	0.6333	0.4806	0.8482	0.3849	0.8482	0.8482	0.7722	0.7585
29	#21	#6	#51	46	0.5552	0.5912	0.9346	0.4928	0.9213	0.9346	0.9462	0.9371
30	#23	#6	#56	36	0.8269	0.6419	0.8606	0.9282	0.8606	0.8606	0.9273	0.9273
31	#24	#6	#60	13	0.3878	0.4992	0.9561	0.6005	0.9561	0.9561	0.9561	0.9561
32	#24	#6	#61	29	0.1977	0.6625	0.9714	0.7870	0.9714	0.9714	0.9830	0.9830
33	#25	#6	#63	81	0.8161	0.4910	0.9883	0.1133	0.9883	0.9883	0.9791	0.9883
34	#25	#6	#64	21	0.2858	0.7938	0.9666	0.8936	0.9666	0.9666	0.9851	0.9851
35	#26	#6	#67	18	0.7029	0.6544	0.9506	0.5971	0.9506	0.9506	0.9506	0.9506
36	#26	#6	#69	16	0.8690	0.5014	0.8690	0.8615	0.8690	0.8690	0.8615	0.8615
37	#27	#6	#71	38	0.4763	0.6021	0.9048	0.7361	0.9048	0.9048	0.9638	0.9279
38	#27	#6	#72	18	0.3247	0.4801	0.8895	0.7619	0.8895	0.8895	0.9308	0.9308
39	#29	#6	#74	43	0.3235	0.6978	0.9930	0.9063	0.3235	0.9930	0.9930	0.9930
40	#30	#6	#75	38	0.7601	0.8577	0.9890	0.9913	0.9890	0.9890	0.9976	0.9976
41	#30	#6	#78	64	0.7770	0.8570	0.9498	0.9985	0.9722	0.9722	0.9993	0.9993
42	#32	#6	#82	17	0.3990	0.5018	0.9410	0.1432	0.9410	0.9410	0.9410	0.9410
43	#32	#6	#83	23	0.9683	0.6870	0.9683	0.9552	0.9683	0.9683	0.9911	0.9911
44	#34	#6	#86	29	0.5558	0.5002	0.9352	0.6516	0.5558	0.9352	0.9633	0.9633
45	#35	#6	#87	19	0.4192	0.5687	0.9475	0.9173	0.4192	0.9475	0.9889	0.9889
46	#35	#6	#88	43	0.5258	0.5324	0.7490	0.7213	0.5258	0.7559	0.7913	0.7966
47	#35	#6	#90	33	0.7510	0.5065	0.9778	0.4996	0.7510	0.9778	0.9778	0.9778
48	#37	#6	#93	14	0.6792	0.5293	0.8961	0.5816	0.6792	0.8961	0.9449	0.9449
49	#37	#6	#94	40	0.7460	0.4991	0.9760	0.7543	0.7460	0.9760	0.9794	0.9794
50	#38	#6	#95	32	0.0839	0.5108	0.9865	0.8360	0.9795	0.9865	0.9865	0.9865
51	#38	#6	#97	17	0.5069	0.4877	0.8886	0.4519	0.8886	0.8886	0.9369	0.8886
52	#40	#6	#101	42	0.7506	0.7045	0.9827	0.6684	0.7506	0.9827	0.9888	0.9888
53	#41	#6	#102	47	0.4854	0.5153	0.9838	0.8703	0.9584	0.9584	0.9805	0.9847
54	#41	#6	#104	70	0.4809	0.5445	0.9506	0.7593	0.9443	0.9506	0.9746	0.9746
55	#41	#6	#105	15	0.3698	0.4670	0.9673	0.9452	0.9673	0.9673	0.9842	0.9842
56	#42	#6	#106	14	0.5130	0.4720	0.9529	0.7871	0.5130	0.9529	0.9529	0.9529
57	#42	#6	#107	13	0.2071	0.4908	0.9702	0.8104	0.2071	0.9702	0.9702	0.9702
58	#42	#6	#108	13	0.9626	0.4733	0.9626	0.6001	0.9626	0.9626	0.9626	0.9626
59	#44	#6	#112	3	0.3393	0.4818	0.8140	0.6607	0.3393	0.8140	0.8140	0.8140
60	#44	#6	#113	14	0.1612	0.4923	0.9667	0.3265	0.1612	0.9667	0.9667	0.9667
61	#45	#6	#114	74	0.6240	0.5005	0.7963	0.6931	0.6240	0.7977	0.9574	0.9574
62	#45	#6	#116	34	0.6481	0.6763	0.9928	0.9821	0.6481	0.9928	0.9928	0.9928
63	#45	#6	#117	37	0.5183	0.5921	0.9549	0.7226	0.5183	0.9549	0.9684	0.9684
64	#46	#6	#118	44	0.7417	0.6813	0.9958	0.9633	0.7345	0.9958	0.9958	0.9886

	Baseline	Version	Activity	Tests	initial	random	gfr-prio	d-prio	vsfr-prio	rcf-prio	multi-prio	multi-eq-prio
65	#46	#6	#119	43	0.1419	0.4940	0.9458	0.6071	0.1419	0.9821	0.9480	0.9821
66	#47	#6	#120	47	0.3493	0.5464	0.9384	0.5802	0.3493	0.9384	0.9323	0.9556
67	#47	#6	#122	39	0.5049	0.4890	0.9478	0.4344	0.5049	0.9435	0.9378	0.9478
68	#48	#6	#125	26	0.2576	0.5073	0.9136	0.3367	0.2576	0.9136	0.9136	0.9136
69	#49	#6	#126	43	0.6621	0.5428	0.9245	0.5307	0.6621	0.8242	0.7468	0.8812
70	#49	#6	#127	43	0.6965	0.5149	0.9909	0.9405	0.6965	0.9857	0.9924	0.9909
71	#49	#6	#128	43	0.4499	0.5564	0.9587	0.2285	0.4499	0.9638	0.8396	0.9587
72	#51	#6	#134	48	0.9869	0.4823	0.9946	0.8837	0.9946	0.9946	0.9946	0.9946
73	#52	#6	#138	49	0.9518	0.4791	0.9881	0.7496	0.9518	0.9881	0.9881	0.9881
74	#52	#6	#139	44	0.4410	0.4854	0.9356	0.2583	0.4410	0.9356	0.9404	0.9425
75	#52	#6	#140	45	0.0768	0.5105	0.0768	0.8197	0.0768	0.0768	0.7819	0.7570
76	#53	#6	#144	41	0.9075	0.6721	0.9887	0.9547	0.9075	0.9887	0.9898	0.9898
77	#53	#6	#145	31	0.7861	0.4857	0.9380	0.1826	0.7861	0.9380	0.9380	0.9380
78	#54	#6	#147	6	0.1701	0.5350	0.4683	0.6759	0.4683	0.4683	0.8561	0.8561
79	#56	#6	#149	22	0.3670	0.5016	0.8839	0.6854	0.3670	0.8839	0.9233	0.9233
80	#56	#6	#150	48	0.5581	0.5443	0.8687	0.5131	0.5581	0.8687	0.8951	0.9066
81	#57	#6	#151	45	0.5758	0.5613	0.9216	0.6781	0.5758	0.9216	0.9630	0.9630
82	#57	#6	#152	35	0.4426	0.4957	0.9445	0.7228	0.4426	0.9445	0.9447	0.9447

Table A.2: The $APFD_C$ values gathered during the experiment for the different prioritization methods. A total of 83 test activities with a total of 2972 test cases were prioritized and evaluated at their corresponding sixth baseline version (i.e. using a limited test history).