CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Bayesian modeling approach to Test Case Prioritization

An Action Research study investigating the requirements and evaluating Bayesian modeling for test case prioritization in a small to medium-sized company

Master's thesis in Computer science and engineering

Adrian Eliasson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Bayesian modeling approach to Test Case Prioritization

An Action Research study investigating the requirements and evaluating Bayesian modeling for test case prioritization in a small to medium-sized company

Adrian Eliasson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Bayesian modeling approach to Test Case Prioritization

An Action Research study investigating the requirements and evaluating Bayesian modeling for test case prioritization in a small to medium-sized company
Adrian Eliasson

Supervisor: Robert Feldt, Department of Computer Science and Engineering
Advisor: Robert Carlsson and Lisa Lönroth, Plejd AB
Examiner: Jennifer Horkoff, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2023

Bayesian modeling approach to Test Case Prioritization

An Action Research study investigating the requirements and evaluating Bayesian modeling for test case prioritization in a small to medium-sized company

Adrian Eliasson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Regression testing taking up valuable time and computational resources is a prevalent issue in software engineering. Lowering the time consumed by the execution of a regression suite enables timely feedback to developers and faster software verification. One technique to reduce the cost of regression testing is called test case prioritization. This project utilized an action research methodology to investigate a case company's requirements specific to test case prioritization and evaluated a novel Bayesian modeling approach against a Heuristic approach. In addition, the results were discussed in relation to Machine Learning-based approaches. The different approaches were evaluated based on several elicited requirements on one open-source dataset and the case company dataset. The results showed that Bayesian modeling performed similarly to heuristic models and machine learning models in terms of early fault detection, even with limited amounts of training data. Furthermore, Bayesian models showed a higher average percentage fault detection than heuristic models on the open-source dataset. From the perspective of small to medium-sized companies (SMEs), common test case prioritization techniques may improve early fault detection, however, additional work may be needed to meet the requirements and demands of the companies' testing and verification practices. A pragmatic approach to test case prioritization for SMEs could use a combination of Bayesian modeling and rule-based or heuristic prioritization.

Keywords: Computer, science, computer science, engineering, project, thesis.

Acknowledgements

First, I would like to express my gratitude to my supervisor Robert Feldt for his great advice and patience throughout the project. I would like to thank my examiner Jennifer Horkoff for her great input on the report. Thank you to Plejd AB for the research opportunity and especially to my supervisors Robert Carlsson and Lisa Lönroth for their time and invaluable support and openness during the project's entirety. A final thank you to my friends and family for their support during these trying times. Without your help, none of this would be possible.

Adrian Eliasson, Gothenburg, January 2023

Contents

List of Figures

List of Tables

1

Introduction

It is not uncommon for single executions of a full regression test suite to take hours or even days to complete. A technique for lowering the time and cost of such test suite executions is Test Case Prioritization (TCP). TCP works by analyzing a variety of software-related artifacts and metrics to prioritize the order of execution for tests in the regression test suite so that faults are detected earlier rather than later. In this way, the number of test case executions needed to find most faults in the code can be limited. Many different approaches to TCP have been presented and analyzed previously, with the use of a wide variety of data metrics. This thesis presents a novel approach to TCP using Bayesian Modeling that uses historical execution data to perform TCP and compares its performance with a Heuristic model. A comparison of the Bayesian models are compared with previous research on ML-based approaches to TCP.

1.1 Statement of the problem

As a software system grows, so do the cost and scope of its related testing practices. The regular use of a regression test suite to validate the system's containing functionality is one such practice. Different techniques have been proposed to combat the growing cost of executing the complete regression suite. One such technique is called TCP and is based on the idea of prioritizing the test cases for execution to find faults earlier rather than later. Proposed solutions to the TCP problem have been studied extensively and in recent years Machine Learning (ML) approaches have been prevalent [1]. Current approaches in TCP can generally be divided into two categories: ML-based approaches and Heuristics-based approaches [2]. There are various forms of ML techniques in use, but the most common ones work by identifying measures of data (features), which are then used to train a computational network to – as accurately as possible – provide an optimal test prioritization. In general, these computational networks used in ML require large amounts of data to be efficient. Collecting enough data to train such a network typically takes several months or even years to accumulate. This data requirement can be detrimental, especially for small and medium-sized companies (SMEs) that do not have this amount of data readily available. Heuristics-based approaches in TCP, on the other hand, are generally rule-based and developed on an ad-hoc basis, and therefore require little to no historical data to perform. While this makes them better suited for

SMEs they are less efficient than ML-based approaches [2]. Therefore, this thesis aims to investigate the possibility of using Bayesian statistics to provide performant test case prioritization with a reduced need for large amounts of data.

1.2 Purpose

This work intends to bridge the gap between these two presented approaches by investigating the use of Bayesian modeling for the problem of TCP. Bayesian modeling originates from the field of Bayesian statistics, a field of statistics which, in contrast to classical, frequentist statistics, makes use of prior distributions and observed data to compute posterior distributions through the use of Bayes theorem [3]. These prior distributions are fed with our previous knowledge about the problem and then the posterior distribution can be computed using our priors and data observations using algorithms such as the Markov-Chain-Monte-Carlo (MCMC) [4]. The posterior distribution can then be used to draw predictions about future observations. The use of Bayesian modeling has been growing increasingly popular in later years due to its more intuitive way of using counts of data and probabilities as well as more comprehensible output information instead of the counterintuitive nature of hypothesis testing. Moreover, the use of Bayesian modeling has been made more accessible through the increased computational ability of today's computers, since MCMC-like methods get increasingly resource-intensive as the number of parameters grows. The field of Machine Learning has drawn inspiration from Bayesian statistics, for example through the use of graphical modeling in Bayesian Networks and Neural networks [5].

By sacrificing the use of large amounts of parameters (where the number of parameters estimated by common neural networks can reach millions), it is possible that Bayesian modeling can reduce the amount of data needed to draw accurate predictions. This lowered amount of data needed can make Bayesian modeling a viable method for TCP, especially for SMEs, where large amounts of data might not be available. In that regard, this work aims to make three distinct contributions: A study of the requirements for TCP in a case SME, an investigation of the performance of Bayesian modeling for TCP in comparison to a heuristic model, an analysis of the impact that differing amounts of data have on the performance of Bayesian modeling in TCP, and a comparison of the results found in this study with previous research on ML-based approaches to TCP. A research research methodology called Action Research will be utilized to conduct this investigation [6].

1.2.1 Research questions

The overarching goal of this thesis is to make it possible for not only large-size companies with an established test data collection process to perform TCP, but also for SMEs and similar organizations with limited to no historical test data available. This goal will be pursued in two parts. The first part is the investigation of a case SME's requirements for TCP in practice, and the development of an initial data collection method for the introduction of TCP. The second part is the development

and evaluation of a Bayesian model for TCP in comparison to a Heuristics-based model, as well as a comparison with results found in previous research on ML-based TCP models. Four research questions will be answered by this work:

1. What are the requirements of an SME in the introduction of TCP and its performance and related practices (data collection, selection of metrics, and automation)?
2. What is the difference in performance between Bayesian models and a Heuristic model in terms of fault detection capability and company specific requirements?
3. How does the amount of available data impact the performance of Bayesian models in TCP?
4. How can the performance in terms of fault detection capability and data size requirement of Bayesian models be interpreted in relation to ML-based TCP models?

1.3 Thesis Outline

Chapter 2 presents related research in TCP and the theoretical background which guides the techniques used for TCP in this thesis.

Chapter 3 covers the scientific methodology that was chosen for this thesis, namely action research, and the different phases of the action research that were performed. Moreover, it will go into some detail regarding the implementation of the TCP at the company.

Chapter 4 presents the results from the project in terms of the requirements that were elicited concerning TCP solutions for the case company and results regarding the developed TCP solutions in terms of these requirements.

Chapter 5 presents a discussion of the findings and provides insights gained from the results and activities of the project and the conclusion that can be drawn within the area of TCP. In addition, this chapter includes a discussion of the main threats to validity and what steps were taken to mitigate these threats.

Chapter 6 contains concluding remarks regarding the study, the main results as well as what conclusions can be drawn from this the results.

2

Theory

This chapter will go into depth regarding the background for this thesis, as well as review related research and concepts to the area of TCP. It will describe Regression Testing and the research investigating different solutions to the problem of costly executions. Furthermore, it will present the research area of TCP and the different previously investigated TCP techniques in the literature. Similar techniques to TCP that have been attempted to combat the increasing cost of regression testing will also be reviewed. It will also present Bayesian statistics and modeling. Lastly, how the adoption of agile and practices regarding continuous integration has impacted regression testing and TCP, as well as the common machine learning techniques from the literature, will be reviewed.

2.1 Background

Software that is being used in practice and that evolves must change according to changing requirements [7]. Evidence of this has been described extensively in Software Evolution and the related subjects of change impact analysis, and software maintenance. As change is made to the software faults and bugs being introduced in the code is inevitable. To not release faulty or buggy code, techniques for finding and dealing with such faults before the software is released are therefore necessary. Many models of software maintenance describe a three-phase process which includes program understanding, program change, and program re-validation [8]. The re-validation of a program after a change has been made is one such technique to find change-induced faults. A method for performing such re-validation in practice that has seen wide adoption is a form of software testing called regression testing [9]. Due to its extensive use as program re-validation, regression testing has been shown to account for much of the cost related to software maintenance [10].

The cost of executing a regression test suite has also been studied in detail. Notably, the size of a regression suite has been found to have a linear relationship with code submission rate, further increasing the cost related to its execution [9]. The growing adoption of Continuous Integration related practices further necessitates the mitigation of regression testing costs. Research has therefore been focused on finding ways to lower the cost of regression testing, by ways of removing redundant or old test cases, selecting only a subset of the total tests for execution, or prioritizing the

order of execution of the test cases, to increase the rate of fault detection [11]. The reason to pursue an increase in the rate of fault detection is many-fold, it will allow faster feedback on the program under test and allow developers to start bug-fixing earlier than otherwise might have been possible [12].

2.2 Regression Testing

Regression testing is the act of retesting old functionality upon the introduction of new functionality. Its goal is to verify that the new changes have not adversely affected the existing functionality, i.e., that the system has not regressed. Regression testing is performed by executing a complete (retest-all strategy) or partial collection of the total regression test suite [11]. As the software evolves and new functionality to the software is continually added, the regression suite grows as well. Due to the already high cost of software maintenance related to the continual testing and retesting of the software, the expansion of the regression suite can be problematic. This problem has been extensively researched, and three main types of solutions to the cost incurred by a constantly growing regression suite have been prevalent. These are Test Suite Minimization (TSM), Test Case Selection (TCS) and Test Case Prioritization (TCP) [11] which will be described in depth in the following sections.

2.2.1 Test Suite Minimization

Test Suite Minimization is a technique that aims to decrease the size of the test suite by removing redundant test cases while retaining a high fault detection capability. Reducing the number of total test cases in the suite, it will also reduce the cost related to executing the complete suite. The main approach to minimization is by determining a subset of tests from the total suite which is regarded as redundant. Early efforts of suite minimization included versions of integer linear programming and the use of data flow graphs, such as decision-to-decision-graphs (dd-graphs) to find redundant test cases [13][14]. Instead of trying to find redundant test cases, Chen and Lau presented the idea of trying to find essential test cases. These are test cases that are essential to finding certain faults: faults that cannot be found with other test cases. They used versions of the greedy algorithm as heuristics to determine these test cases [15]. Offut et al. expanded on the use of greedy algorithms but with the addition of mutation testing to minimize test sets [16]. In later years, Lin et al. developed a tool called *Nemo*, capable of using multiple criteria for minimization and can show both reductions in suite size and increased fault detection capability after minimization compared to previous work. There have been conflicting results from studies investigating the performance of TSM. Gregg et al. showed that the fault detection capability of the minimized suites was significantly reduced [17], whereas other studies have shown only minor decreases in fault detection capability after TSM [18][19][20]. Some possible reasons for the conflicting results were presented together with the findings, the majority of which were attributed to differences in the SUT or differences in the test suite [11]. Due to the reduction in fault detection capability, as well as the difficulty of minimizing a test suite, especially for complex builds, TSM is not widely used in practice [21].

2.2.2 Test Case Selection

Test case selection is similar to TSM in the sense that they both attempt to select a subset of test cases for execution. The main difference lies in what sources of data and what criteria for selection are used. While suite minimization uses many coverage-related metrics for choosing an optimal subset of tests for execution, TCS focuses more on choosing subsets based on changes in the SUT from version to version. A common definition of the TCS problem is stated as:

Given: Program P, a new version of P, P' and test suite T.

Problem: Find a subset of the test suite T, T' to test P' with.

Changes in the SUT can be found by investigating the difference between P and P'. Different approaches to this have been proposed in the literature. One is Dynamic Slicing. Dynamic Slicing is based on the ideas of execution slices, relevant slices, and approximate relevant slices. An execution slice is the set of statements executed by a test case. A dynamic slice is a subset computed from an execution slice based on output variables and changes made to the execution slice from the previous version of the code. The dynamic slice only contains the statements which impact the output from the test case. Thus, a test case is only selected to be run if the dynamic slice of a test case has had modifications. Relevant slices are the set of instructions contained in the dynamic slice but with the addition of an account predicate statement causing the execution of the statements in the dynamic slice. The data dependency of this predicate could then be added to the slice and if it receives a modification, the test case should be re-run. An approximate relevant slice is an even more conservative choice of statements which further includes the addition of all predicate statements which incurred the execution of the relevant slice[22].

Another approach is called Graph Walking. Graph Walking differs from code coverage techniques in that it not only shows the difference in changed code and possible fault-revealing tests for those parts of the code but also takes the control flow of the application into account, for example by use of so-called Control Dependence Graphs (CDG) Similar work uses Program Dependence Graphs (PDG) and System Dependence Graphs (SDG) [23]. Later, graph walking based on Control Flow Graphs (CFG) was also used [24]. Another is identifying textual differences in files using tools like diff in UNIX or by extracting commit logs from the VCS, using one of the many tools available to do this [25].

2.3 Test Case Prioritization

Test case prioritization is another technique used for dealing with the increasing cost of regression testing. Differing from test case minimization and test case selection, prioritization does not remove any tests nor create subsets of the test suite, it simply prioritizes the test suite. TCP aims to create an ordering of the test suite so that when executed, it will pursue some objective by the testers. Many possible objectives can be pursued by testers for regression testing. For example, testers can aim to

maximize early code coverage or maximize coverage of frequently used features. Objectives can also be combined, as shown by Per Erik Strandberg et al. [26]. To evaluate the efficiency of an algorithm for TCP according to the current objective pursued by the testers, Rothermel et al. established the use of something called an "objective function" [27]. The idea of an objective function is that given the test suite, its ordering, and, if necessary, any relevant outcome information from the suite execution, it will output the degree to which the algorithm was able to pursue the current prioritization objective. An objective function is useful not only to evaluate the applied TCP algorithm but also to compare the efficiency of one TCP algorithm to another. Formally, the objective function can be described (from Siavash Mirarab) [28] as:

Definition 1. Test Case Prioritization Problem:

Given: a program P_v, its maintained test suite T_v^m, a new version P_{v+1}, and a function $f(T)$ from a test suite T to a real number,

Problem: to find an ordered test suite $T_v^o \in Perm(T_v^m)$ such that $\forall T \in Perm(T_v^m)$, $f(T) \leq f(T_v^o)$.

Here, $Perm(T)$ is used to denote the set of possible permutations of the tests in the suite T, such that it includes the same tests but in a different order. in other words, the TCP problem aims to find an optimal ordering of the test suite concerning the objective function $f(T)$.

The most common objective for prioritization in the literature is to maximize the rate of fault detection. The rate of fault detection can be measured from a given test suite ordering and its verdicts by using an objective function called "Average Percentage of Fault Detection" (APFD). The APFD measure was first introduced by Rothermel et al. [27] and has since then been extensively used to evaluate the efficiency of TCP algorithms.

The definition for APFD (from Yoo and Harman [11]) can be written as follows:

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

where TF_i is the index of the fault, n is the total number of test cases in the test suite, and m is the total number of faults detected.

The value gained from increasing the rate of fault detection is two-fold. First, finding faults earlier in the execution could allow the developers to start debugging earlier than what might otherwise have been possible [27]. Second, if there is a limited time window to run the tests a subset can be selected from the prioritized suite so that the total fault detection is improved over such a subset of a non-prioritized test suite.

Other examples of objective functions are measures of the rate of code coverage,

frequency of feature usage and historical fault-proneness [12]. Parejo et al. also define seven different objective functions, including measures of dissimilarity between test cases, cyclomatic complexity, and the historical number of changes in the test case [29]. These functions provide different values to the developers but all of them aim to lower the cost of the regression testing by gaining value earlier in the execution cycle rather than later.

Many TCP method categories were presented by Yoo et al. [11]. These include, among others, Requirement-based approaches [30], model-based approaches [31][32] and probabilistic approaches [33][34]. Yaraghi et al. instead distinguish between ML-approaches and Heuristics-approaches [2].

2.4 Bayesian statistic and modeling

Bayesian statistics is a branch of statistics based around using counting to represent probabilities of events. This approach to statistics is viewed to be more intuitive than frequentist statistics. Bayesian modeling is a way to model systems and is later used to predict the outcomes of these systems. Bayesian statistics is based on the mathematical theorem called the Bayes Theorem, which states that knowledge of the outcomes of a variable can be described in terms of our previous knowledge of the system and the knowledge of the system in general. Mathematically this can be stated as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.2}$$

with P(A|B) being the probability of A occurring given that B is true, P(B|A) the probability of B occurring given that A is true, P(A) and P(B) the probabilities of A and B occurring without any given conditions. In other words, given the probability of the events A and B and the probability of B given A, the probability of A given B can be calculated. This theorem is further used for Bayesian modeling to incorporate previous knowledge of a system to predict outcomes. In this thesis, a common model notation is used to present the different Bayesian models used in the project. Examples of this notation are provided in section 4.3.2.

2.5 Continuous Integration and Test Case Prioritization

With the advent of an agile approach to software development, concepts such as Continuous Integration (CI) and Continuous Delivery (CDE) have become ubiquitous. Thus, in later years, researchers have investigated the use of TCP in a CI environment, where builds are run very often to provide fast feedback to developers – something that TCP also aims to do. However, in a CI environment, the time used for executing test suites is at a premium which means that traditional TCP methods

often need to adapt to new constraints. Especially expensive static code analysis and coverage approaches may be problematic when required time and processing resources are not available [35]. For example, Marijan et. al. developed a TCP algorithm specifically produced for a CI environment that attempts to maximize the number of tests run within given testing time constraints [36]. Facing scalability issues with an increased amount of available historical data, this algorithm was later replaced by a deep learning model more suited to the large-size dataset. A deep learning model with few hidden layers was chosen to reduce training time, further emphasizing the time limitations for a CI environment [37]. Despite the implied limitations of a CI environment for TCP, further research into the use of advanced machine learning models has still been prevalent, as demonstrated by Pan et al [1]. Solutions have instead focused on trying to combat the limitations of the environment by adapting the models often by decreasing the size of the total dataset by using a smaller subset for prediction [1].

2.6 Information Sources

Earlier research on TCP has varied in the sense of which information sources are used for prioritization. Earlier research in ML-based approaches gives us clues into which data sources are preferable to use for TCP, due either to its cost of collection or impact on performance. Test execution history seems to have the highest impact on prediction accuracy for faults [2]. Earlier research suggests that using the complete set of historical outcomes could lead to a performance reduction [38].

The different information sources in terms of the TePIA taxonomy can be categorized as follows:

- test case execution, execution time, exec, simple, dynamic, num, 1:1
- test case report, failure frequency, exec, simple, dynamic, num, 1:N
- test case history, historical fails, exec, simple, dynamic, int, 1:N
- test case history, historical effectiveness, exec, derived, dynamic, num, 1:N

Ramírez et. al. proposed a taxonomy for ordering the different types of data that can be collected, named the TePIA Taxonomy [39]. TePIA is a comprehensive taxonomy for data collection in TCP.

2.7 Machine Learning approach

Recent years have seen extensive research into the use of machine learning techniques in many areas of software development. In TCP, the use of these techniques has also been studied. For example, Pan et. al. reviewed 29 primary studies between 2006 and 2020, all using some form of machine learning to perform test case selection or test case prioritization or both [1]. They classify the ML techniques into four groups based on their implementation details: Supervised learning, Unsupervised learning, Reinforcement learning (RL), and Natural Language Processing

models (NLP). Reinforcement learning seems to have the greatest benefit in regression testing, especially in a CI context, due to the ability to feed new information into an already trained neural network. This reduces the need for retraining the network, which is one of the most time-consuming activities when using Machine Learning models. Spieker et. al. developed an implementation of an RL model called RETECS and evaluated its TCP performance on an open-source data set. Their evaluation will be used in later chapters to compare the performance between Machine Learning models and Bayesian models [38]. As previously mentioned, a Deep Learning model called DeepOrder was developed by Marijan et. al. to improve upon an earlier TCP algorithm to prioritize the order of test cases based on historical information of regression suite executions. This was also evaluated on the same open-source dataset as RETECS and will be used as a comparison with the Bayesian models in the later chapters. DeepOrder showed only some improvement on the previous model, especially on a larger dataset. An average APFD of 0.76 was measured on the open-source dataset [37].

2.8 Related Work

The related work in TCP can be divided into three larger areas of TCS, TSM, and TCP. Initially, most research trying to solve the problem of increasing cost concerning regression testing attempted to use TSM. It would include removing test cases from the test suite or modifying them to keep code coverage while lowering execution time. TCS could be viewed as a subfield of TSM because then you would also try to select a subset of tests for regression suite execution. However, TCS techniques tend to be modification aware, meaning the selection process will consider the latest changes in the SUT. TCP does not select or modify the test suite, but rather focuses on prioritizing the test cases within the suite. In the area of TCP, many different approaches have been researched. One approach has been to use statistical methods for prioritizing test cases. Lately, most of these have used different Machine Learning techniques to prioritize test cases. However, there has not yet been research investigating the use of Bayesian statistical modeling for prioritization.

3

Methods

To answer the research questions stated in the introduction this project has used Action research (AR). AR is a scientific method originally developed for use in social science research to investigate the effect of some action in an organization in 5 steps or phases [40]. In later years this method has found use in the software development research community. The book Action Research in Software Engineering by Miroslav Staron [6] is used extensively to guide the methods used in this project.

3.1 Research Context

According to the European Commission's definition of small and medium-sized enterprises (SMEs), the case company is considered an SME. The definition states that if a company has a total staff headcount of between 10 and 250 and its turnover is between 2m euro and 50m euro or a balance sheet total of between 2m euro and 43m euro it is considered an SME [41]. The reason for the particular focus on SMEs in this thesis is not necessarily due to their economic situation or employee count, but rather the maturity and sophistication of their regression testing practices.

3.1.1 The Case Company

The case company is a Swedish IoT and smart lighting company that provides wireless dim and relay control of products for home use. As a company that develops both hardware and software, it has many technological responsibilities, including production and quality control, firmware development, software development, and cloud development. Each of these areas has its own set of testing requirements and techniques. For example, hardware testing must take into account compliance policies, as the products will be installed in homes to control power supply. The company's developed application for wireless control of products has different testing requirements, as errors are not as detrimental, and the application is not as regulated as hardware testing.

In terms of it's regression testing practices, the case company has been performing nightly builds and has had to opt out of running certain tests to meet the time deadline. Previously, the company did not prioritize test cases for execution, and they were run in alphabetical order. Exploring state-of-the-art TCP alternatives

could help the company improve its regression testing practice and provide better test suite coverage through prioritization. This project was conducted in collaboration with the company's integration test team ("system test"), which consisted of four to five team members, and performed testing on mainly the software part of the case company's product portfolio. In this sense, I would say that the case company represents a typical SME in terms of testing practices. The company, which had around 90 employees and was growing, agreed to provide access to both the system-test team members for interviews and workshops and to the key systems for this project.

3.2 Action Research

This project is conceptualized in a cycle of five phases according to the practice of action research [6]. It is more unstructured than for example design science research or an experiment, where the problem and method are clearly defined beforehand. However, it is like design science-oriented towards deliverables, both in the results of the research but also for milestones during the action taking. Action research is aimed at contributing to the company's way of working and knowledge by performing the research in intimate conjunction with the practitioners. The closeness to practitioners is a property of action research that allows it to be a bit more unstructured methodology [6]. By:

1. investigating of the problem space (Diagnosing Phase),
2. research planning together with the practitioners (Planning Phase),
3. joint establishment of evaluation metrics and action taking that is equally enforced by practitioners and researcher (Action Taking Phase),
4. evaluation of the actions that were taken (Evaluation Phase) and,
5. drawing conclusions and learnings from the project (Learning Phase),

all in consecutive order, the entire research project can be performed in a more agile matter and be tailored specifically to the case company. Using Action research will help us find the actual problems at hand at the industry level and the requirements which are key to the case company for TCP solutions.

Because Action Research is aimed at integrating the researchers into the development teams, in order to really understand the problem at hand and the symptoms that the practitioners face daily, much focus in this project was placed on integrating activities. In this spirit, and due to the free access to the team members given by the case company, the researcher participated in daily status meetings, office space interactions, and various team activities. Moreover, the interaction was spread out between e-mails, in-person discussions at the office, as well as using messages over a common communication network (Slack). This kind of daily and personal interaction is in line with the Action Research methodology, as it improves the ability for the researcher to understand the problem in the diagnosing phase, as well as enhances it's ability to devise proper solutions and evaluation frameworks to test the developed solutions.

An entire action research project is normally not limited to a single cycle but instead conducted as a succession of two or more cycles, each with its five phases. This succession and iteration of cycles are conceptually displayed in figure 3.1. This project, however, was limited to a single cycle due to time constraints for the entire project, where the implementation of more than one cycle would only result in an overly rushed project and the introduction of risks to the quality of the research. By carefully and systematically conducting the single cycle of action research, I found the results from the project sufficient to answer the research questions and purpose of the study. The five different phases performed in a cycle of action research all have their utility. In the following section, I will outline the contents of the five phases according to Staron [6], while the next chapters present in further detail how these phases were carried out within this study.

The first phase is the diagnosing phase and its central aim is to develop an understanding of the problem, both for the practitioners and the researchers. The main approach to do this is by interviewing the stakeholders or having group workshops to elicit the practitioners' view of the problem and possible solution space. In other words, the diagnosing phase is where the symptoms of the problem at hand are identified. For the researcher, a significant portion of phase one also includes an investigation of the literature related to the problem and the symptoms identified.

The second phase is the action planning phase, which includes translating the problem defined in the diagnosing phase into concrete goals and a potential solution as well as our hypothesis of the outcome and how it might affect the case company. To approach the goals of the study, an action plan needs to be created. Beyond the inclusion of the action broken down into manageable pieces and a discussion of what the researchers will do and what the practitioners will do, the action planning also needs to make sure that the researchers have access to both relevant competencies at the company and the relevant systems and infrastructure. Due to its deliverable-oriented methodology, the action plan also needs to include milestones and attached deliverables for the action-taking. A plan of what meetings should be had, with whom, and when is also important to include as it is important to discuss intermediate results and present milestone deliverables to the stakeholders and practitioners.

The third phase is the action-taking phase, which includes setting up an evaluation framework and taking the action. The evaluation framework is one of the most important steps in action research as it helps us answer the question if the action had the intended results. Depending on the type of action to be undertaken the evaluation framework may differ, e.g., for a factorial experiment you would most likely use quantitative factor analysis, whereas when investigating the effects of a change in stand-up meeting time on employee satisfaction you could instead use a thematic analysis for evaluation. There is also a distinction between direct action taking and indirect action taking, where the former means to introduce a change directly and evaluate its effect and the practitioners' view of their changed way of working, and the latter implies a form of presentation of results or an analysis of the effects of the action taking to influence the practitioners to introduce the change by

themselves.

The fourth phase is the evaluation phase where the effect of the action is evaluated. This is the phase where the evaluation framework that was developed in the previous phase shows its usefulness. The evaluation phase is important because it lays the foundation for what kind of learning can be drawn from the project in the next and final phase of the action research cycle. Using the correct set of result measurements is among the most vital steps taken in any research as this is the source of information from which conclusions about causal relationships can be drawn. It is also the focal point of attention regarding questions and criticisms of validity and reliability in the study. What we must ask ourselves is, therefore, if the evaluation framework answers the research questions stated in the introductory chapter correctly and consistently, i.e., "are the results reliable?", as well as if it answers them accurately, i.e., "are the results valid?".

The fifth and final phase is the learning phase in which the things that were learned from evaluating the action are documented. It is targeted mainly in two directions, to the practitioners and the broader research community. Specifying the learning gives the company the ability to apply this new knowledge for new solutions and to communicate the knowledge to others in their own company or other companies. Specifying learning to the broader research community is important in the sense that it allows future research to build upon it, and for the researchers to contribute to further knowledge in the state of the art. In the case of successive action research cycles, the learning phase also plays a big role in providing information for the first phase of the next cycle, the diagnosing phase. Because the diagnosing phase aims to understand the problem at hand and the related symptoms of the issue, previous knowledge of the context and results and discussions might prove invaluable.

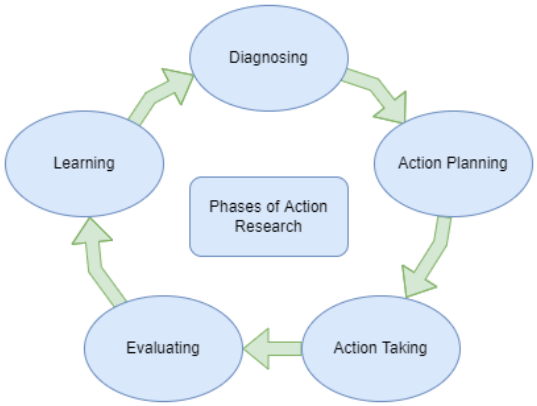

Figure 3.1: The phases of Action Research

3.2.1 Workshops

Throughout the research project, several workshops were held together with the company supervisors. Every conducted workshop had the aim of being finished within 45 minutes, but for a couple of them, another meeting was arranged because

the participants did not feel satisfied with the discussions during the first 45 minutes. The length of the second meeting was decided depending on how much time that felt needed to conclude the earlier discussions. An initial itinerary was constructed before each workshop stating the goal and discussion points of the workshop to keep the discussion focused on the required topic and to estimate how much time would be required. I generally led the discussion according to the itinerary but opened for free discussion around the points and let the company supervisors lead the discussion in a direction they felt was important. Giving the company supervisors significant control of the discussion led to insights regarding what the company felt was important regarding TCP solutions. Furthermore, bi-weekly meetings were utilized as status report meetings and during the action-taking phase used to present milestones and deliverables.

3.2.2 Diagnosing Phase

For this thesis, the diagnosing phase comprised investigating the previous research within regression testing, test case prioritization, and related research. It also comprised joint problem formulation between the researcher and the company stakeholder representatives (the company supervisors). The first workshop was held during the diagnosing phase. The goal of this workshop was to create a problem formulation and potential solution ideas and to elicit the supervisors' view of the problem and symptoms related to the company's regression testing practices and test prioritization. A set of initial requirements that were important to the company for a TCP solution was also gathered during this workshop. The selection of metrics to use for TCP was an integral part of this phase. A list of possible metrics was produced from reviewing literature and cross-checking with the company context. An initial set of metrics that could be collected and regarded as relevant according to previous research [1] were produced during the workshop, such as historical verdicts and duration of previous regression test suite executions, which were referenced together with the metrics from the literature review to select which metrics would be used for the TCP solution.

3.2.3 Action Planning phase

Phase 2 included a translation from the information gathered from the diagnosing phase in terms of workshop results, literature review, and requirements elicitation into a 2-part actionable plan. The goals of the action, the intended solution, the expected outcome, and the expected impact on the company were all part of the first part of the plan. The second part of the plan included a list of actionable tasks, a concoction of which people at the company to be used, a list of which tasks to be regarded as milestones and their connected deliverables, and some ideas of when status and planning meetings, presentations and documentation were to occur. The second part can be considered as the action that was later undertaken and evaluated during the project and was constructed using the resulting documentation from phase 1. The plan was reviewed, refined, and agreed upon together with the company supervisors during a short following meeting.

3.2.4 Action Taking Phase

This phase included establishing an evaluation framework for the impact of TCP, and also performing the action of introducing the different parts of TCP, e.g., data collection and modeling, in the case company. The established evaluation frameworks will be presented in the next chapter. The action that was conducted during this project can be seen as an indirect action taking, where the ways of working for the practitioners did not directly change, but a new implementation of a test case prioritization was created, and an analysis of the results was provided to influence the practitioners' choice of using this implementation of test case prioritization in the current ways of working. The evaluation framework was developed together with the company supervisor as an adapted version of a GQM framework [42], where the focus did not lay on a strict following of GQM, but rather focused on the requirements that were elicited during diagnosing phase and how these can be measured from the results. During a workshop with the company supervisors, the requirements established earlier were used to devise a set of measures that would satisfactorily answer to what degree the requirements were met by the developed test prioritization solution. Furthermore, the action was undertaken according to the steps outlined in the action plan from the previous phase, and milestones and deliverables were presented during status report meetings.

Two main activities were pursued during this phase: data collection and the TCP algorithms/models. The data collection was constructed according to how the company currently was doing nightly builds, and readily available metrics were extracted directly from the build system interface by parsing execution result output files. The tool to perform this data collection was developed using the Python language [43]. The TCP models were developed using a mix of Python and the R programming language [44]. The heuristic model and some data preprocessing were developed in Python while Bayesian models and the majority of preprocessing of the data were done in R. The model development was done incrementally according to the guidelines provided by C. Furia et al [45].

3.2.5 Evaluation Phase

In this thesis, the evaluation phase included the construction of a set of measurements, partly produced together with the company supervisors, and partly drawn from the TCP literature. Most of the measurements constructed for this study were directly inferred from how the company representatives felt the measure would accurately depict whether the produced TCP solution met the company's requirements. These measures were presented and interpreted mostly by visual investigation, such as colored plots of faulting tests and bar charts showing the criticality of test cases. Other measurements, e.g., the APFD measure, were used as an objective measure of the efficiency of early fault detection used in the wider research community for reporting the results of similar TCP algorithms. The collection of viewpoints, opinions, and results regarding the measures was extracted during workshops and status report meetings for the current milestone deliverable.

3.2.6 Learning Phase

The learning phase prepares the results and evaluation from the previous phases to provide a basis for learning both within the company and the scientific community. For this thesis, this included preparing a presentation of the produced programs and an explanation and discussion of the conclusion that can be drawn from the project. Furthermore, the documentation collected during the entire action research project was compiled to present it as part of the thesis.

3.3 Comparing Results with Machine Learning Models

In order to answer whether the Bayesian models can perform better or worse than machine learning models there needs to be a way to make a trustworthy comparison. Because no Machine Learning models were implemented during this project, the comparison can not be made on the company specific dataset. The existence of open-source datasets opens up the possibility of applying and evaluating the Bayesian models on the same open-source dataset that other researchers have implemented Machine Learning models. Because the models depend only on their implementation and the input dataset, and since the dataset is the same, this comparison of results should be possible.

Research questions 4 depends on this comparison between Machine Learning models and Bayesian models, and there are two ways this comparison should be made: the performance of the models in terms of APFD and the extent to which the dataset size impacts this performance should be investigated through this comparison. The Machine Learning part of these questions can be found in two other studies, which have both implemented and evaluated machine learning models on the same open-source dataset as the one chosen in this project (the choice of open-source dataset will be discussed further in the next section). Spieker et. al. have implemented and evaluated a reinforcement model called RETECS and Marijian et. al. have implemented and evaluated a deep learning model called DeepOrder on this dataset [38][37]. How these studies help answer this research questions together with the results from the Bayesian models will be presented in the discussion chapter.

3.4 The Datasets Used

Some of the models presented in this thesis were tested using an open-source dataset. The dataset used is called "ABB Paint Control" and represents 352 build cycles spread over about a year's time. This data was collected and published by ABB Robotics Norway in 2016, and it includes historical information about test case executions and their recorded results (verdicts/ outcomes). It has 114 test cases, and a total of 25,594 datapoints, of which 19.36% represent a failed verdict. This can be compared with the dataset generated during this project form the company, which consisted of about 120 build cycles (four months of data collection) of 244 test cases

which resulted in about 30 000 data points. The open-source dataset is commonly used in research aiming to test models performing test case prioritization or test case selection. As mentioned briefly in section 2.7, several other researchers have recently used this dataset to this end [37][38][46]. There were three main reasons this open-source dataset was selected for use in this project. The first was that it is a common dataset that has seen use among different researchers historically, and would thus allow a way of comparing the results between this project and other research projects. Second, because it was one of the goals of this project to compare the developed models with other implemented ML-based models, and this dataset had been used to evaluate other machine learning models, such as RETECS [38] and DeepOrder [37], it was a good candidate to allow this comparison. The third reason was that because the ABB Paint Control dataset and the dataset generated at the case company was similar in the number of cycles, test cases and data points, the comparison between them could improve the generalizability of the results gathered during this project.

4

Results

The results will be presented in order of the action research phases. First, the outcome of the initial workshop in the diagnosing phase will be shown. The action planning phase subsection will provide an overview of the items in the action plan, what tasks were undertaken and what the expected outcome of the action would be. In the action taking part, the evaluation frameworks will be considered in detail, along with the developed models and the metrics used. The evaluation of the action will be examined in the fourth section, especially focusing on the model's effectiveness both on the company's data and on openly available datasets. The "learnings" section is framed as a discussion where attempts to draw conclusions are made and is therefore not kept as a part of the results chapter but instead left to the discussion chapter.

4.1 Diagnosis results

Most of the results presented in this section are related to the workshop conducted in the diagnosis phase. Both the company requirements elicited, and the available forms of data will be discussed.

4.1.1 Company requirements (RQ1)

From the initial requirements elicitation in phase 1 some requirements were deemed key for the action's outcome. These were requirements that were both related to the fault detection capability of the prioritization model and other qualities of the prioritization. The different requirements were also given a priority rating between them, such that prioritizing the tests over their fault probability is more important than prioritizing tests that have not been run in a while. The main goals explained by the company regarding the TCP model were to increase the efficiency of the regression testing and to find problems with the SUT. From these two goals, the following requirements were presented as attempting to reach these goals or attempting to cover a standalone requirement like usability.

Prioritize Critical Functionality

The case company had a list of criticality levels for each test case in the suite. These were applied on a group basis, meaning all tests belonging to the same category were marked with the same criticality level and were actively and manually maintained. The criticality levels ranged from low to critical, with four different levels named "Low", "Medium", "High" and "Critical". One of the requirements from the company when prioritizing the suite was to take criticality into account so that highly critical test cases are prioritized higher than test cases with low criticality.

Prioritize High Fault Probability Tests

Because one of the main goals of the TCP model for the company was to increase the efficiency of the regression testing practice, the case company wanted to prioritize the test cases for the highest chance of detecting faults. This ties into the second main goal that the company had, which was to find problems in the SUT. The requirement for prioritizing high fault probability tests originated from these goals, and this is also aligned with the scientific literature regarding TCP. Prioritizing tests according to their fault detection capability is the main goal of the common TCP models and is also the metric that is used to evaluate them. The fault detection capability can be measured using APFD, which is described in further detail in chapter 2.3.

Automatic Collection and modeling

The case company wanted the whole prioritization process to be automatic, or at least with minimal manual work needed to maintain the prioritization algorithm. The company accepted some manual work for the prioritization but as little as possible was preferable. This tied in well with usability as a strong criterion from the company because fewer manual steps required to keep the model up-to-date and functional would increase the usability of the solution.

Consistent Test Execution

Another key aspect of the case company's requirements was that they wanted test cases to be consistently executed. The specific case to be avoided was the model giving a test case such a low priority that, in case of added test case selection (for example by imposing a time limit for the regression test suite execution), it is very rarely executed. The time requirement for how often a test case should be executed was argued to be dependent on the size of the suite, but for the current suite size, about 10 days was the longest time allowed before it had to be re-executed.

4.1.2 Data Sources

An automation server (Jenkins) was used to build and execute the regression suite daily at the case company. This automation server kept records of test case metadata from several previous executions. A python script was developed to parse and save important features from the meta-data for later modeling and analysis. The features

accessible from the automation server were **Test Case Name**, **Test Case Group**, **Duration of execution** and **Outcome/Verdict** of the test case execution. The test case group, execution time (duration), and verdict were the main information attributes used for test case prioritization in this project.

A. Ramírez et al. presented a taxonomy based on 3 different kinds of attributes, Relational, SUT, and Testing/test-case [39]. The first attribute, Relational, aims to capture the relationships between the test cases and the SUT codebase. The second attribute, SUT, aims to capture the properties which depend only on the SUT codebase. The third attribute, Testing, focuses on the deployed testing framework and meta-data regarding the tests both in a historical view and regarding their innate properties. Because the test suite at the case company only interacts with the SUT through a graphical user interface, the information attributes (metrics) in the first two attributes were disregarded and focus was placed on attributes in the Testing category. The Dependency and Similarity attribute groups of the Testing category require the development of additional programs to derive the information. Therefore, they were disregarded in favor of the metrics with a higher availability for the case company. Both the specific investigation into SMEs and the company requirements regarding automation and limiting manual labor contributed to this decision. The remaining attribute groups: Execution, History, Property, and Report were pursued instead, due to their ease of collection and impact on fault prediction [1].

Execution attributes: Allocation time, cost and resource utilization were all unavailable for this project, since these would all require manual development to produce, and providing an allocation time was not a requirement for TCP in this company. However, Execution time and total time were both available in this company and could be readily extracted.

History attributes: All the history attributes were available and could be reasonably easily extracted. Historical verdicts were used for the thesis's TCP models.

Property attributes: The age was not available for the test cases but could be counted from the day the test case prioritization started. The test case code was available and information regarding the size of the test case code could have been used in this project but was left in favor of more common history-based metrics which could be extracted directly from the nightly build system. The Resources attribute was not available at the company, since tests did not have a specified list of resources needed for execution. Estimated-fault-detection and textual-description would require manual labor to generate the relevant data, which was one of the company's requirements to avoid if possible, so these attributes were therefore not collected. The criticality metric could be translated to be a static priority for the test cases, but would not serve the intended purpose of the metric as a fault detection information source. A version of status was discussed during the initial workshops to include age or status with regards to if it was changed recently or not. This metric was not included in the final TCP models. Type-of-test was also not included, since no such specification existed.

Report attributes: Both test case effectiveness and failure frequency were utilized in this thesis's TCP models.

4.1.3 Selected metrics

The three selected information attributes can be interpreted in terms of this taxonomy. In total what was used: Historical verdicts, execution time, effectiveness, and failure frequency. **Historical verdicts** are the outcome/verdict of a particular test case from a number (called n) of previous executions. Several different values for n were selected and evaluated during the project, and for the final Bayesian model values of 7 and 21 were used. Additionally, the verdict had 4 distinct possible values, The outcome result had several different possible values including "Pass", "Fail", "Not Executed", and "Not Run", which allowed for the retrieval of metrics such as time since last execution. The specific value of "Not Run" was only assigned to a test case when it was manually selected not to be executed during that nightly build. The **Execution time** was recorded from previous executions and input to the Bayesian model for predicting verdicts of test cases. **Group belonging** of the test cases could be viewed as a sort of code similarity, and this information was available due to the test cases belonging to a class that tests the same type of functionality. Table 4.1 adapted from Ramirez et. al. describes the used attributes in terms of the testing information [39].

Attribute name	Source	(Collection)	Possible Values
Historical verdicts	History	Simple	Integer
Execution time	Report	Simple	Positive Double
Group (code-similarity)	Similarity	Derived	Factor variable

Table 4.1: Simplified version of the TEPIA taxonomy attribute classification based on testing information for the chosen information sources in this project.

4.2 Planning results

As described in the method section, the action plan was divided into two parts, the first detailing the solution idea and expected impact for the company, and the second presenting a task list with connected milestones and deliverables. Therefore, this section will also be similarly presented in two parts.

4.2.1 Solution idea

The produced solution idea was to use state-of-the-art TCP techniques to increase early fault detection capability in the regression testing suite and meet the requirements from diagnosing phase. The specific TCP technique to be used was Bayesian statistics. A heuristic model was also to be developed and used as a basis for comparison against the Bayesian model.

4.2.2 Expected impact

The expected outcome from the action was the development of automatic data collection according to the established metrics to be used from diagnosing phase, as well as models for test case prioritization, both Bayesian and heuristic. The expectation was that one or a combination of the developed models could be used to meet the collection of requirements placed on the solution. A reduction of manual labor from selecting and prioritizing tests for upcoming nightly builds and lowering the cost of increasing the size of the nightly test suite was also an expected impact for the company.

4.2.3 Tasks

The different tasks produced in the action plan were the following:

- **(Milestone:)** Introduce data collection of the decided metrics and save these to some persistent storage.
- Develop a baseline heuristics model for TSP.
- Develop a primitive Bayesian model for TSP.
- Hook the models up to work on our regression suite collected data.
- **(Milestone:)** Calculate $APFD$ or $APFD_c$ and present as intermediate results.
- **(Milestone:)** Intermediate evaluation using established frameworks.
- Evolve the Bayesian model for increased performance.
- Compare an ML model with the developed Bayesian on an openly available dataset.
- **(Milestone:)** Evaluate performance: Heuristics vs. Bayesian vs. ML and present as final results.
- **(Milestone:)** Evaluate with established evaluation frameworks.
- Learning.

Among these tasks, there were five milestones, each representing a key step in the action of implementing TCP. The first milestone included the introduction of the entire data collection process for the data sources. The second milestone included several previous tasks, namely the development of prioritization models. The third milestone included assessing the models with the company requirements. The fourth and fifth milestone was the final evaluation of the developed models using all the established evaluation frameworks and on two different datasets.

4.2.4 Deliverables

The deliverables were all connected to the tasks tagged as milestones and were presented during status report meetings. The deliverable for the first milestone was a program that, daily, fetched the nightly build execution results directly from the automatic build system used at the company (Jenkins). For the second milestone, the initial models were evaluated using APFD. This was to provide a basis for further development of the models and initial opinions from the company supervisors about

what felt missing or lacking and where the development should focus its efforts. The third deliverable was to provide an initial view of the measurements of the results, and thus feedback was given about how clear the visualizations were. The fourth deliverable provided results from the final evaluation using the different models on the company's data. The fifth deliverable was a presentation of the results with regard to the requirements placed by the company on a TCP solution.

4.3 Action taking results

The results from the action-taking will be presented in parts. First, the evaluation frameworks that were developed will be presented, then the different developed models will be presented.

4.3.1 Evaluation Framework

Five different evaluation frameworks were developed for this project. For each of the requirements found during the diagnosis phase, an evaluation framework was developed to measure the degree to which the developed models met the requirements.

4.3.1.1 Prioritize Critical Functionality

To measure how well the developed models prioritized over criticality of functionality, a visualization was introduced. The test cases will be in the order of their execution on the X-axis. On the Y-axis, it shows the execution verdict. The colors will show criticality (Levels of green-red as Low-Critical). An ocular observation made by the company supervisors would be used to assess the measurement outcome. Because the criticality includes several levels, a metric to measure how prioritized they are (with a high degree of similarity to the APFD metric) would be somewhat more complicated than developing an APFD measure, and thus the assessment of the outcome was limited to be an ocular observation of the visualization.

4.3.1.2 Prioritize High Fault Probability Tests

Prioritizing the test cases to maximize the probability of detecting faults was the main objective of the developed models. The measure of APFD (presented in chapter 2.1) is repeated here:

$$APFD = 1 - \frac{TF_1 + ... + TF_m}{nm} + \frac{1}{2n} \qquad \text{(2.1 revisited)}$$

was used to measure the early fault detection capability of a model. Because this evaluation framework is ubiquitous in the research investigating the efficiency of test case prioritization, it was decided early in the project that this would be used to measure our models as well. To provide an efficient and clear way of interpreting the fault-detection capability of the models, a visualization was also developed for this evaluation framework. This visualization orders the test cases in order of execution on the X-axis and its verdict on the Y-axis. Thus, company supervisors were able

to interpret the APFD outcome as both an objective number (APFD) and as a visualization just as with the criticality metric.

4.3.1.3 Automatic Collection and modeling

Measuring the degree to which the collection and modeling were made to be automatic was decided to be a subjective measure by the action team and company together. It was an estimation by the action team as to how much manual labor would be needed and was measured in minutes per day or week.

4.3.1.4 Consistent Test Execution

Another visualization was developed for measuring how long since some tests were executed. This differed somewhat from the earlier visualizations of criticality and verdicts in the sense that this did not show any outcomes of the execution. Instead, it showed all the test cases for a given date on the X-axis and the number of days for which these test cases had not been an execution in the nightly regression test run on the Y-axis.

4.3.2 Bayesian Models

Five different Bayesian models were developed. Each in increasing complexity according to C. Furia et al.[45].

$$\begin{aligned} failed &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= \alpha + \beta^{f7} * f7 \end{aligned} \tag{4.1}$$

$$\begin{aligned} failed &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= \alpha + \beta^{f7} * f7 + \beta^{f21} * f21 \end{aligned} \tag{4.2}$$

$$\begin{aligned} failed &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= \alpha + \beta^{f7} * f7 + \beta^{f21} * f21 + \beta^{g} * group\_id \end{aligned} \tag{4.3}$$

$$\begin{aligned} failed &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= \alpha + \beta^{f7} * f7 + \beta^{f21} * f21 + \beta^{d} * duration \end{aligned} \tag{4.4}$$

$$\begin{aligned} failed &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &\sim \alpha + \beta^{f7} * f7 + \beta^{f21} * f21 + group\_id + \beta^{d} * duration \end{aligned} \tag{4.5}$$

where f7 denotes the precalculated number of failures the specific test case had in the last 7 days, and f21 denotes the number of failures the test case had in within the days 8-21.

4.3.2.1 Heuristic Model

The developed heuristic model in this project was based on a few guiding principles, mainly relying on the experience of the practitioners and an iterative development process. As there are many types of heuristic methods to solving difficult problems, and many classes of heuristics, the main characteristic of any heuristic is that it is a simplified model relying on previous experience or knowledge that aims to reduce cognitive or computational overhead. To that end, a heuristic was developed during this project that depends only on the historical information of the test cases at hand (the same dataset that the Bayesian models use). The heuristic model is a simple calculation that uses the same features as the Bayesian models, *f7* and *f21* numbers to compute some level of priority. These numbers still represent the number of faults that the test cases have produced in the last 7 days, and the last 8-21 days respectively.

The model works by assigning each test case a priority number for the current execution cycle. This priority number is calculated for each test case depending on it's previous failure history. It is similar to another heuristic employed by Porter et al., in the sense that it accounts for historical outcomes with a higher score given to more recent failures [33].

The calculation of producing the priority number of a test case is as follows:

$$priority\_number = k_7 * scale(f7) + k_{21} * scale(f21) \tag{4.6}$$

where k_7 and k_{21} are both manually set scaling factors ($k_7 = 1, k_{21} = 0.3$ was chosen in this project) and scale being a function in the R base package to scale a given statistic to N(0,1). The numbers given to the weights k_7 and k_{21} were chosen partly based on the practitioners' experience, but also through a small series of trial and error. This number of trial and error tests were kept at a minimum to avoid overfitting the model to the particular dataset. The reason different variables were scaled is because they tend to be different in size due to the length of history they account for. The *f7* number can only have the values 0-7, while the *f21* number can have the values 0-14. By scaling them down to a normal distribution N(0,1), weight can be applied (k_7 and k_{21}) without the priority number being weighted by the shape or size of the original features.

Notably, this calculation is similar to that of model 1.2, the difference being that instead of letting the Bayesian model learn the scaling factors from training on the data, the factors are here set manually ad-hoc and from the practitioners' previous experience and knowledge.

4.4 Evaluation results

Phase 4 included the evaluation of several different models developed during Phase 3 (action-taking). Five different Bayesian models were developed, and one heuristic model was developed. This was then evaluated together to compare the results.

The highest-performing Bayesian model will also be presented in this section in comparison with the heuristic model and two baselines: Random Ordering and Alphabetical Order (unchanged). Two parts will follow in this section, the first will present the results relating to APFD performance and a comparison between the models, and the other will present results pertaining specifically to the company requirements for a TCP solution.

4.4.1 Results from Executions

The following section presents the APFD results for the different models on both the company's data and open-source data. The datasets used are described in section 3.4. The models were adapted to be able to operate on both the open-source and the company datasets without many changes to their inner workings, in order to provide as much of an equal chance to perform as possible.

4.4.1.1 Closed source data (RQ1& RQ2)

To measure the performance of the different Bayesian models and the heuristic model on the company dataset, the different Bayesian models were trained on the complete dataset and then compared using the APFD measure, as can be seen in the following figure. The models all performed in a highly similar fashion, with an average APFD around 0.7 as can be observed in figure 4.1.

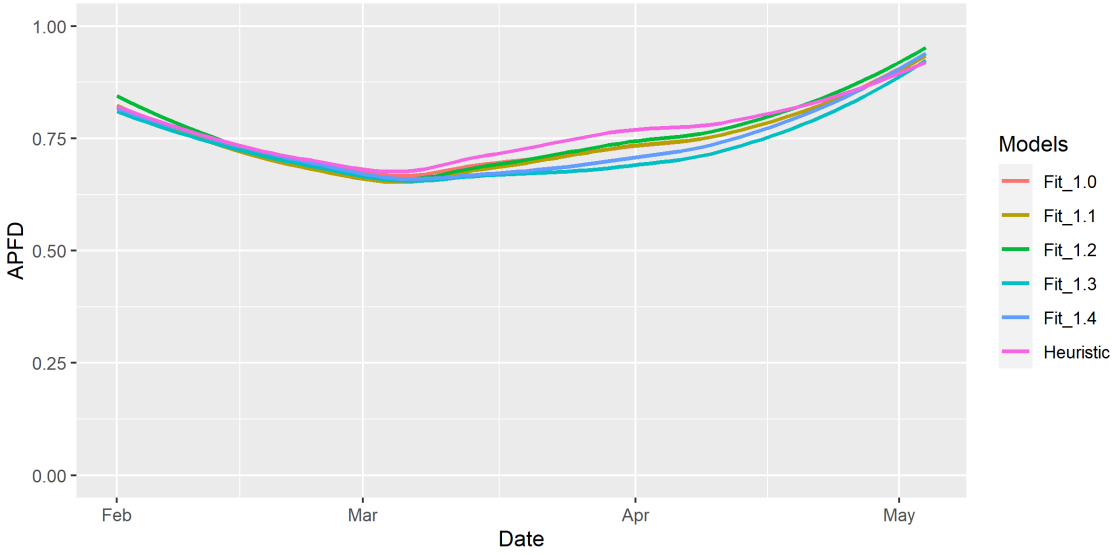

Figure 4.1: Average APFD measurements for Bayesian and Heuristic models

Figure 4.1 shows how the different Bayesian models were performing compared to each other. The Y axis shows the APFD measure for the Bayesian or Heuristic model on a specific date, and the X axis represents the date ranging from early February to the beginning of May. Due to the high number of dates recorded with the APFD measure, a smoothing function was applied to the figure to improve clarity and find trends in the models' performance over time. All models performed comparably, and there were no significant differences that could be observed regarding the APFD

measure. Using the average APFD performance of the different models, the model called fit_1.1 in the figure was selected as the best one overall. This model was later compared to the heuristic model and baselines of alphabetical order and random order.

Interestingly, the models' APFD differed between periods of the project. As shown in figure 4.1, the models provided APFD values of well over 0.75 both during the first couple of weeks and during the last month of the project. During the middle period of the project, the APFD measurements shrunk to around 0.65 on average.

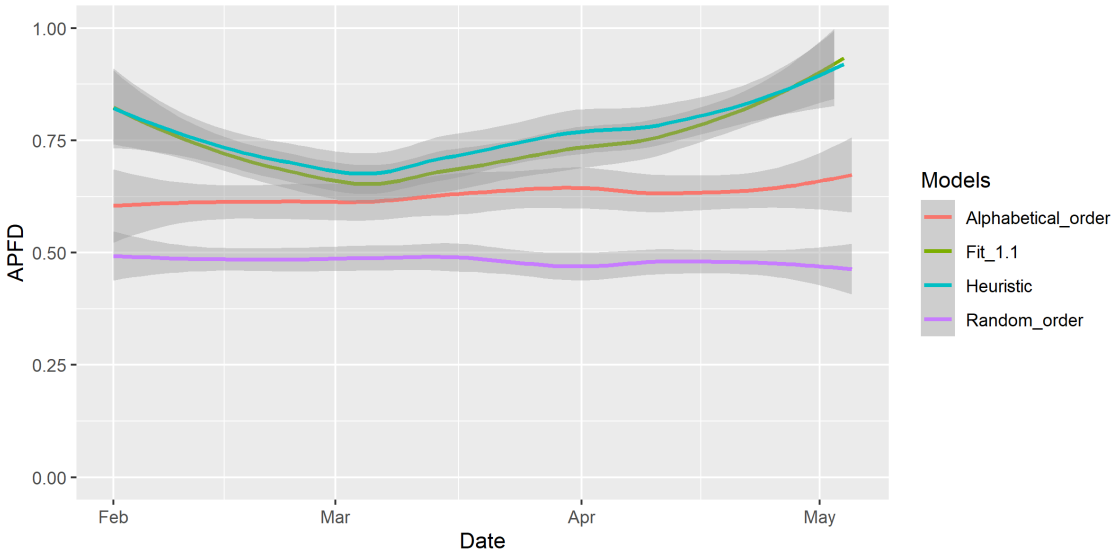

Figure 4.2: Best performing Bayesian model comparison with baselines and heuristic model

Figure 4.2 shows how well the best Bayesian model performed compared to the heuristic model and two different baselines. Once again, the lines represent the APFD measure of the models on the Y-axis, and now a gray field represents the standard deviation as well. The first baseline was the random priority and is shown in purple in the figure. As expected, this baseline produced APFD measurements of around 0.5 throughout the project. The other baseline is an alphabetical priority, which was the priority already in use at the case company. Interestingly, this baseline performed significantly higher than the random baseline on average. A reason for this higher performance is largely because there was a large set of test cases in the suite, all beginning with the letter A which quite often detected faults during regression suite execution. This means that any model that employs an alphabetical ordering would most likely perform higher than the random baseline, simply due to this specific set of test cases. Further inspection of this figure reveals how similarly the Bayesian and heuristic model performed. Both models performed showed a significantly higher APFD than the alphabetical ordering baseline. Interestingly, the heuristic model performed slightly better than the Bayesian model, even though the heuristic model was less complex and significantly easier for testers to employ as means of test case prioritization.

To investigate the impact of using different historical periods in the Bayesian model, differing number of days used for predicting was investigated on the company data. Five different models were generated, each using a different number of days when predicting. 3, 5, 7, 10, and 20 days were used. The models can be specified as follows:

$$\begin{aligned} Verdict &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= 1 + \beta^{f3} * f3 \end{aligned} \tag{4.7}$$

$$\begin{aligned} Verdict &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= 1 + \beta^{f5} * f5 \end{aligned} \tag{4.8}$$

$$\begin{aligned} Verdict &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= 1 + \beta^{f7} * f7 \end{aligned} \tag{4.9}$$

$$\begin{aligned} Verdict &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= 1 + \beta^{f10} * f10 \end{aligned} \tag{4.10}$$

$$\begin{aligned} Verdict &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= 1 + \beta^{f20} * f20 \end{aligned} \tag{4.11}$$

These models were all trained on the complete dataset, and the measurement of APFD over time was then calculated and compared between them. The results of this comparison can be observed in figure 4.3. Here the Y-axis again shows the APFD measurements over time, and the X-axis shows the dates ranging from early February to the beginning of May. The different lines here represent the APFD measurements of Alphabetical ordering, Random ordering, and Heuristic ordering. Furthermore, there are lines representing the different lengths of history used for predicting the outcome of future regression suite executions. The lines are tagged in the legend by how many days' worth of data was used for prediction, with $fail\_03$ meaning three days' worth of data was used, $fail\_05$ meaning five days' worth of data, and so on.

Once again, the developed models showed a significantly higher average APFD than both of the baselines. Just like the previous models, these performed significantly better in the first two weeks and the last month of the collected data, and worse during the middle portion. This is displayed by the fact that all the models except

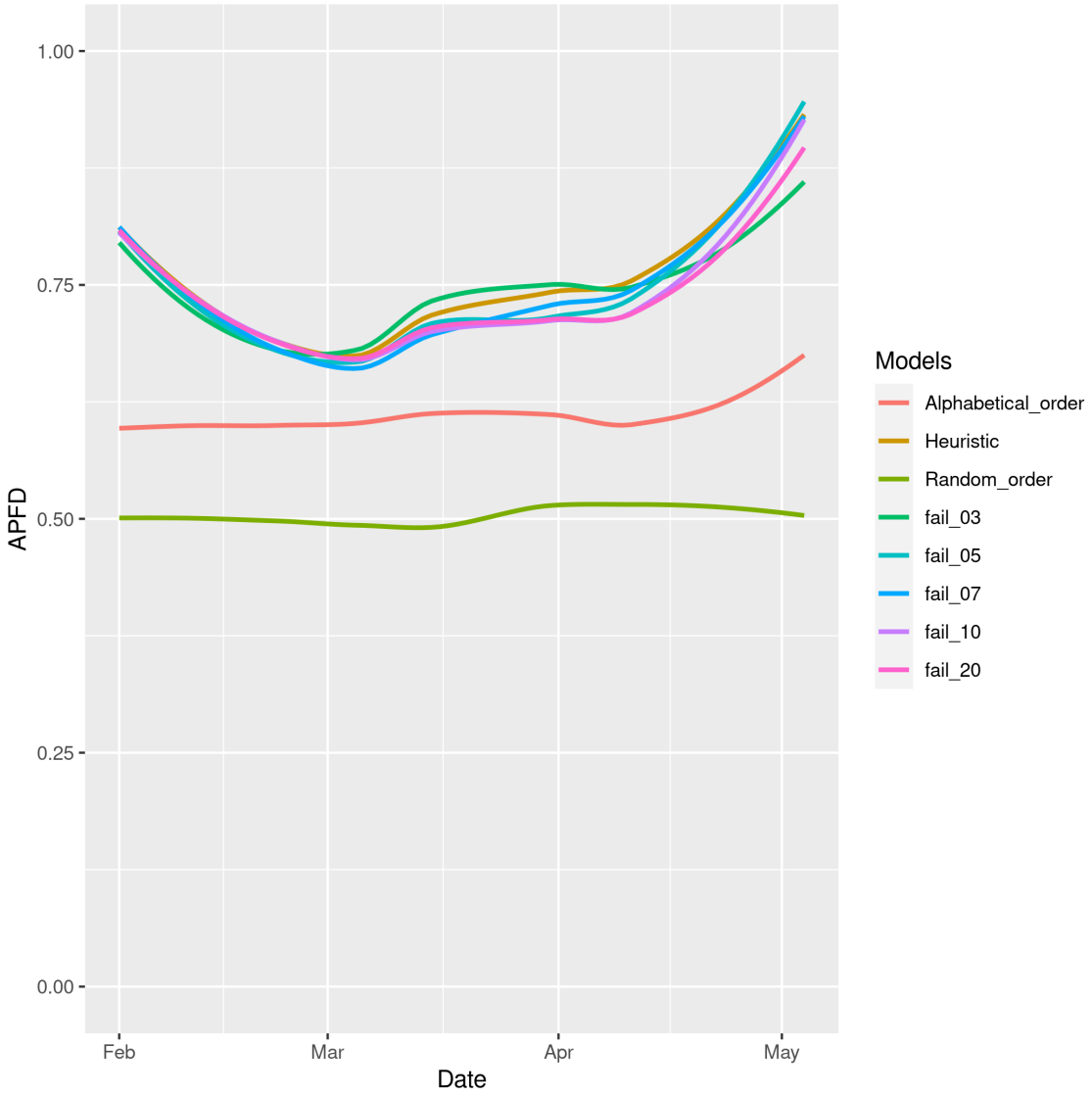

Figure 4.3: APFD measurements over time for Bayesian models with differing history length usage

the baselines had APFD measures above 0.75 during the first two weeks of February, dipping down to around 0.65 during March and slowly improving over time, topping out at values around 0.9 towards the beginning of May. Despite the relatively low performance during the middle portion of the project, they still showed a higher APFD than both the baselines on average. Interestingly all the Bayesian models performed comparably, despite their drastically different history length usage. This could imply that the dataset had some intrinsic properties that affected the performance of the models. If the outcome from every execution was highly similar, meaning the same test cases that failed previously often failed again, then the length of the history used should play a smaller role. However, in this case, the models should be able to easily score very high in the APFD measure, which was not the case here. If the outcome from every execution was very dissimilar instead, the effect could also be that the length of history used does not matter as much. This would

probably slightly favor a shorter history length over the longer ones, and this effect can be seen, slightly, during the middle portion of the collected dataset. This could mean that there is something to be gained by changing the history length used for outcome prediction depending on the volatility of the outcomes in the dataset. For highly volatile regression suites, perhaps a shorter time span is effective, while for highly stable suites, a longer time span could be preferable.

4.4.1.2 open-source data (RQ2&3)

For the open-source data, similar models were implemented as in the case of the case company dataset. The difference between the open-source data to the case company's data is that the open-source data does not include any sort of grouping factor like the case company data did. Therefore, any group factor could not be included for the Bayesian models in the open-source data.

The Bayesian models developed for the open-source data can be specified as the following:

$$\begin{aligned} Verdict &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= \beta^{f7} * f7 \end{aligned} \tag{4.12}$$

$$\begin{aligned} Verdict &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= \beta^{f21} * f21 \end{aligned} \tag{4.13}$$

$$\begin{aligned} Verdict &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= \beta^{f7} * f7 + \beta^{f21} * f21 \end{aligned} \tag{4.14}$$

$$\begin{aligned} Verdict &\sim \text{Bernoulli}(p) \\ \text{logit(p)} &= \beta^{f7} * f7 + \beta^{f21} * f21 + \beta^{d} * duration \end{aligned} \tag{4.15}$$

To effectively compare the models between the different datasets, the aim is to let the models be as similar as possible when it is not possible for them to be absolutely the same. The best-performing model on the company dataset has all its used metrics also available in the open-source dataset so that the model could effectively be compared. The results of comparing the Bayesian models developed on the open-source dataset with the heuristic priority are shown in figure 4.4.

Figure 4.4 includes the APFD measurements over the complete dataset. Once again, the Y axis represents the APFD measures over time for the different models, and

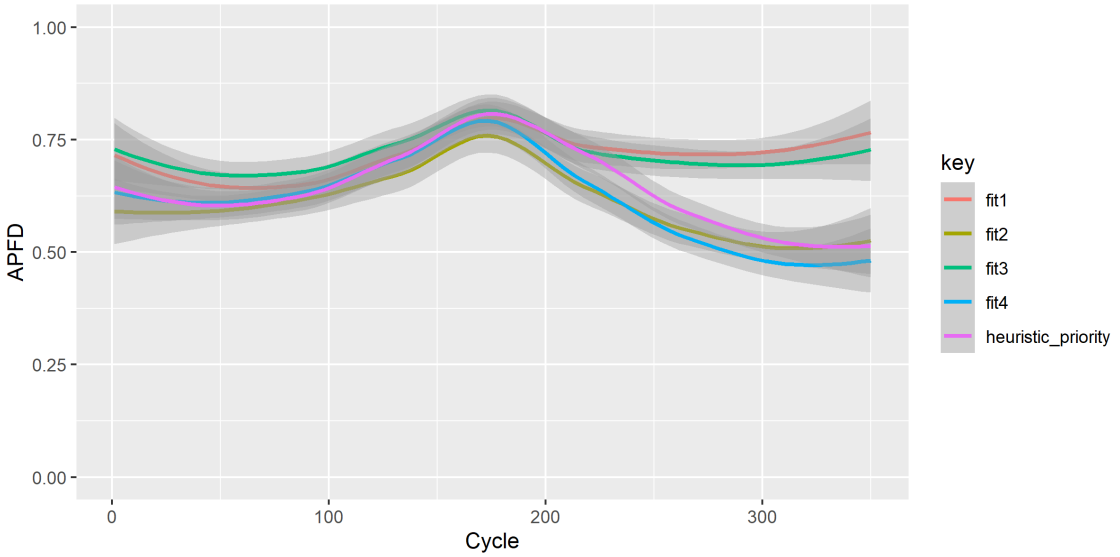

Figure 4.4: Model comparison Open-Source Dataset

the X axis represents the consecutive cycles recorded in the open-source dataset. The different models tagged with the name fit and a number represent the different Bayesian models evaluated. These are presented in the legend in the same order as they were presented in model form above, meaning *fit1* represents model 4.12, *fit2* represents model 4.13 and so on.

The APFD values that can be viewed are similar on average compared to the Company dataset, with APFD values of around 0.7. However, the characteristics of the dataset and the model's ability to predict fault-detecting test cases differ significantly. On the open-source dataset, the models performed generally better in the middle portion of the dataset, and worse towards the beginning and towards the end, which is the opposite of what occurred in the company dataset. Additionally, there were some significant differences between the models on this specific dataset. Models named *fit1*, and *fit3*, both performed showed, on average, higher APFD values than the other Bayesian models and the heuristic model on the second half of the dataset. *Fit1* had an average APFD of 0.72, and *fit3* had an average APFD of 0.71. What sets these models apart is their high usage of the history length of 7 days, as opposed to the longer history of 21 days or the inclusion of the test case duration as a parameter. This indicates that any model which relies heavily on close-range historical values performed produced higher APFD than those relying on a longer historical horizon for prediction, and simply the inclusion of the duration parameter reduced the efficiency of the model, differing almost 0.2 points of APFD from the stronger models towards the later cycles of the dataset.

To visualize how the variance evolved for the models on the data set over time, figure 4.5 is helpful. This figure shows the APFD measurements on the Y-axis and the consecutive cycles of the dataset on the X-axis. The sharp black line represents the exact APFD measurements per cycle for the model named *fit1* and the thicker blue line shows how the model performed over time. A striking feature that can

be interpreted from the figure is the amount of variance from cycle to cycle that was present in the open-source dataset. The difference between the smoothed line plot and the plot showing the exact APFD values for the model is large. While the smoothed line plot is useful in detecting the key trends of the models' performance over time, the model's performance from one cycle to the next is somewhat unstable, with rare cases producing APFD measurements close to 0 ad other times close to 1.

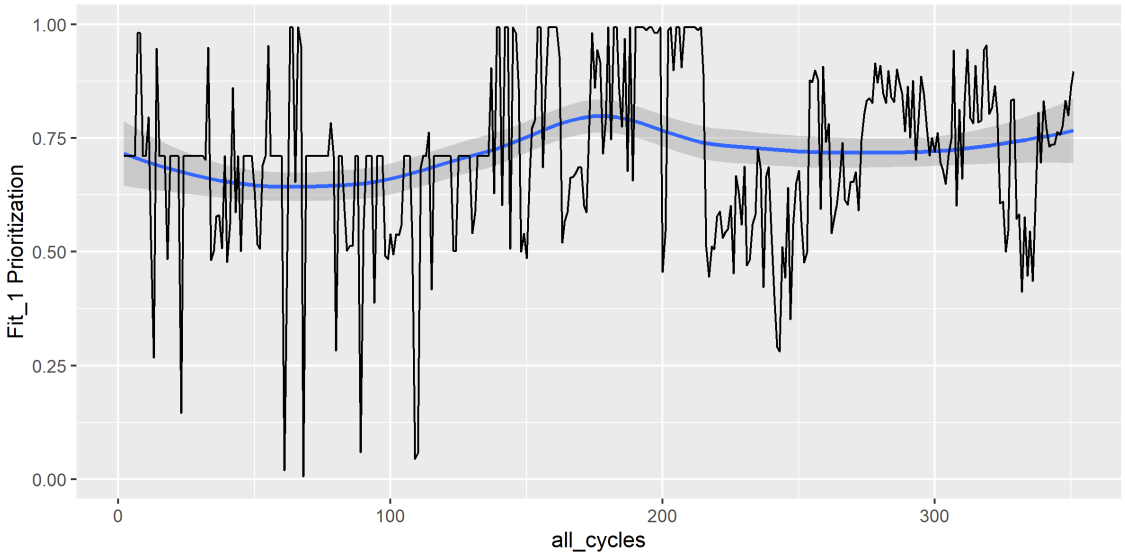

Figure 4.5: Performance comparison with differing training data size

Similar diagrams depicting the variance and exact APFD measurements on both the open-source dataset and the company dataset can be found in the appendix.

4.4.1.3 Data size impact on Bayesian models

To investigate the data size impact on the models, additional model comparisons were made. A set of different models were generated and fitted. Each of the models was trained on a limited number of days' worth of data. One week was the lowest amount of data used for training, and the highest amount used was the entire dataset. All the training data was collected beginning from the end of the cycles, meaning that when training the model using one week's worth of data, the last seven cycles of the dataset were used as opposed to the first seven cycles. The model denoted as *fit1* in figure 4.4 was the one selected to be used when training on differing amounts of data.

As can be observed in figure 4.6, the models using different training amounts seem to all outperform the heuristic model towards the end of the dataset, no matter how much data was used for training. This is an interesting difference between the Company dataset and the open-source dataset, where the heuristic model seemed to almost outperform the Bayesian models on the company dataset, whereas on the open-source dataset the heuristic priority had slightly lower APFD measurements in the early cycles and showed APFD values of almost 0.2 lower than the Bayesian models during the final third of the recorded cycles. Another point to make is

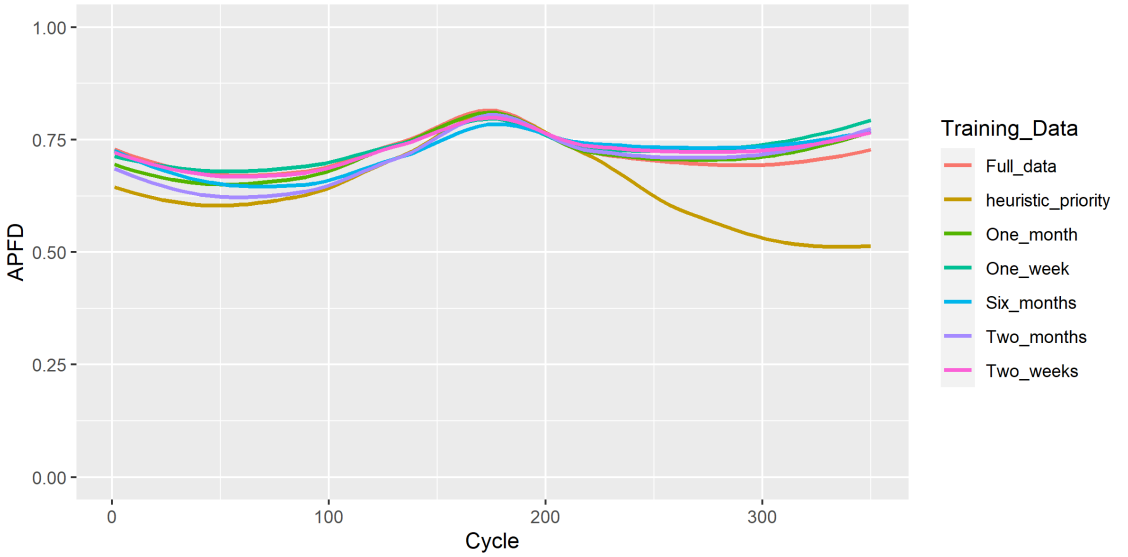

Figure 4.6: Performance comparison with differing training data size using a smoothed line plot.

how similarly the different data sizes performed using the Bayesian models on this dataset. The exact differences can be viewed in diagram A.1 in the appendix, figure 4.6 shows that the Bayesian models do not differ more than about 0.05 values of APFD anywhere on the dataset, despite the difference in data sizes between the lowest and the highest being about 50 times.

4.4.2 Company requirements results

This section will present the results of the Bayesian models when assessing the different company requirements. Different diagrams and visualization were used for this assessment, and these will be presented here.

4.4.2.1 Prioritize critical functionality

When assessing the critical functionality of the models the diagram displayed in figure 4.7 was used. Because there were so many test cases listed in the diagram, the criticality level was not distinguished on the Y-axis, but instead using shades of blue. Darker shades of blue represent higher criticality, and lighter shades of blue represent lower criticality. Therefore, the Y-axis does not differ between test cases but instead is placed statically at 1 uniformly to improve clarity when assessing the different shades of blue. The X-axis represents each of the 244 test cases included in the dataset. This type of diagram was accessible by the company supervisors, as the shades could easily be interpreted as criticality, even when the number of test cases was large and criticality was highly spread out for the execution order.

Figure 4.7 shows the criticality spread using the heuristic priority model. The criticality can be observed to be fairly grouped in the prioritized suite. Since the criticality of test cases was applied to test groups at a time, it could imply that the

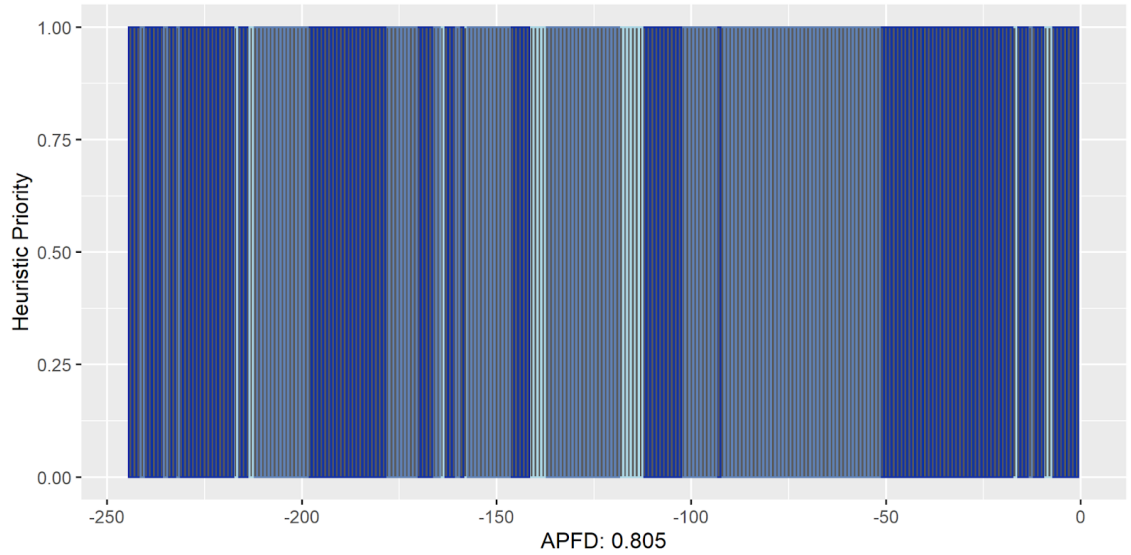

Figure 4.7: Criticality evaluation heuristic model

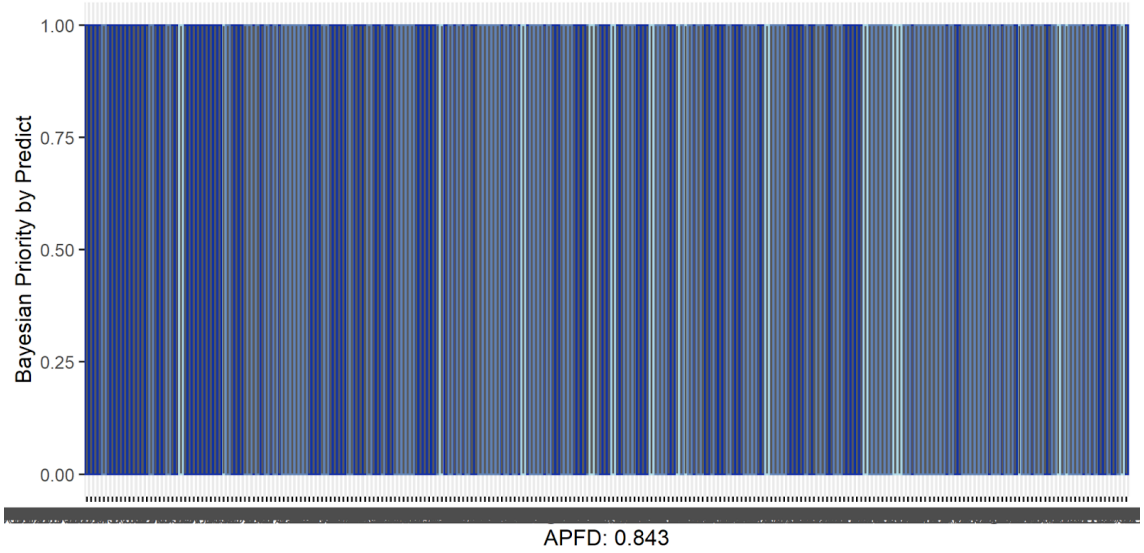

Figure 4.8: Criticality evaluation Bayesian model

heuristic model prioritized the test cases often together with test cases of the same group. In figure 4.8 the prioritization using the Bayesian model is shown. The order of the test cases in terms of criticality was closer to random than for the heuristic model. It seems also that critical test cases were placed earlier in the execution order than less critical test cases, however, the difference is rather small. Since neither of the models explicitly attempts to modify the priority of the test suite according to criticality, it is expected to see no significant prioritization of criticality in these diagrams.

4.4.2.2 Automatic Collection and modeling

Assessing the level of automation in the TCP solution was the only evaluation framework that did not include any form of diagrams for displaying it. This was a requirement that was important to the case company, but not something that could easily be measured objectively in the project. When executing the tasks during the action-taking, special consideration was made to this requirement. This resulted in the Data collection being completely automatic, with no daily developer time needing to be used for maintaining the program. The level of automation for modeling was also made completely automatic and was scheduled to be both retrained and run in conjunction with the data collection program.

4.4.2.3 Consistent Test Execution

One of the requirements was established to be that the model should prioritize tests that had gone without execution higher than test cases that had been executed very recently. When assessing this requirement another visualization was used. Figure 4.9 is an example of the figure described in section 4.3.1.4 which was used as the evaluation framework for this requirement.

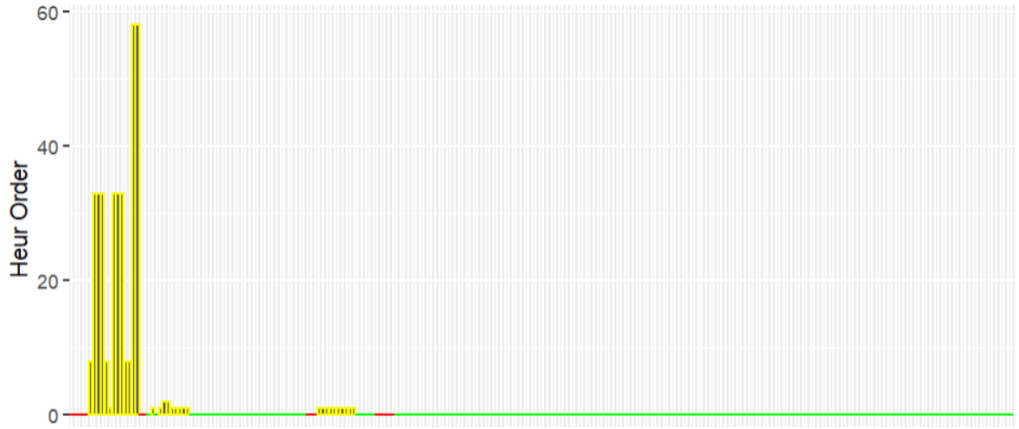

Figure 4.9: Time since last execution by test case

This visualization displays the order of the prioritized test cases on the X-axis, with the color of the bars displaying the outcome of this specific suite execution. The height of the bars represented the time that passed since they were last executed. Because all the test cases were included in the regression suite, meaning there was no test case selection performed explicitly, the models did not differ in how long they allowed tests to remain not executed. The reason some test cases were not executed (the yellow bars with height in the figure), was due to a manual flagging of the test cases not to be executed in the build system. These were test cases that were tagged by the testers due either to the test case not functioning properly or requiring further development.

5

Discussion

This chapter will provide a discussion of the results found during the study. It will also discuss the benefits and drawbacks of the use of Bayesian statistics for TCP and how it compares in terms of APFD and case company requirements with the heuristic model and machine learning model. Furthermore, it will provide a discussion about the research questions and how they can be answered using the results gathered throughout the study. Lastly this chapter will cover possible threats to validity, what the risks are and what steps were taken to mitigate those risks.

5.1 SME company requirements for TCP solution

To address the first research question stated in this thesis a thorough investigation into the specific requirements that the company had on a possible test case prioritization model was conducted. Several key requirements were elicited, and they were all presented in chapter 4.1. The requirements were stated as follows:

1. Prioritize Critical Functionality,
2. Prioritize High Fault Probability Tests,
3. Automatic Collection and modeling,
4. Long since executed test cases should be executed again.

The case company had a significant level of focus on requirements not directly tied to the model's ability to yield a high fault detection capability. Requirements that were focused more on the process surrounding the test case prioritization were of high priority, such as the prioritization of tests that were not executed for some time, or the level of manual intervention required for the model's functionality. Furthermore, the ability to prioritize not only tests that often reveal faults, but also tests that are critical to the main functionality of the SUT was also perceived to be of great importance.

5.2 APFD comparison between Bayesian models and a Heuristic model

The developed Bayesian models and the Heuristic model showed similar performance in terms of APFD on the company dataset, but a larger difference was seen in the favor of Bayesian models on the open-source dataset. On the company dataset, the need to accommodate several key requirements which can more easily be developed using heuristic or rule-based models might make these a superior choice over Bayesian models for the case company and other SMEs that are in the early phase of conducting TCP. On the open-source dataset, there was a larger performance difference between the Bayesian models and the developed heuristic model. This performance difference together with its impressive performance with very low amounts of previous historical test execution data might lead companies to choose to adopt such a model instead of heuristic approaches.

5.3 Data size impact on the Bayesian models

To answer research question three, the results from chapter 4.4.1.3 can be reviewed. One of the key strengths displayed by the Bayesian models using historical execution information also becomes apparent when observing these results. The Bayesian model which used only one week's worth of training data, and the Bayesian model which used the complete dataset for training produced very similar levels of APFD throughout the entire dataset. This could mean that the Bayesian models require very little data to become proficient at predicting test outcomes, at least on historical execution outcome data. Another explanation could be that the dataset containing historical test outcomes is rudimentary enough that the relationships contained in the data source are easily learned. Either way, both the effortless data extraction process and the low data size requirement provide significant evidence that historical test execution information can serve as a strong base for TCP models for SMEs.

5.4 Comparison between Bayesian models and ML-based models

As no machine learning models were developed during this project, the comparison that can be made between the Bayesian models and the Heuristic model is a comparison of the results gained by research implementing Machine Learning models in the same open-source datasets as the Bayesian models in this study. Spieker et. al. developed a network-based reinforcement learning model and evaluated its performance in the same open-source dataset as was used in this project [38]. The reinforcement learning model seems to require about 60 cycles on the open-source dataset to become proficient in early fault detection prediction, which is significantly longer than the 7 suite execution cycles required by the Bayesian model developed in this project.

Another machine learning model that can be compared with the Bayesian models is the deep learning model called DeepOrder developed by Marijan et. al. DeepOrder presented an average APFD of 0.76 throughout the dataset. This is comparable to the developed best performing Bayesian models developed in this project, namely *fit1* and *fit3* that had an average APFD on the open-source dataset of 0.72 and 0.71 respectively. DeepOrder performs about 0.05 APFD values better than the developed Bayesian models on average.

5.5 Bayesian modeling in TCP

The benefits of Bayesian modeling are two-fold. First, it can be argued that the Bayesian models require less amount of data to perform equally to corresponding Machine Learning models due to the lower number of parameters and the use of priors when designing the Bayesian models. The second benefit is the possibility to extract uncertainty-information about the outcome of the predictions, whereas in machine learning models this is a significantly more difficult task. The certainty information can be used to answer business-related questions which are broader than the task of TCP, such as if there is specific functionality that often breaks (giving the model a high uncertainty whether the test case focusing on that specific piece of code will pass or fail). Uncertainty information regarding the test case's outcome can also be of interest to the testers, as it can present valuable insights into characteristics of the test case, such as its level of flakiness or ability to find faults. The uncertainty-information regarding test cases was something presented by the company supervisors as something that could be of interest, to detect which test cases presented irregular outcomes. The use of uncertainty information in test cases was not further investigated in this study, but for future research, it would be an interesting topic to pursue and investigate its use for the testing process or the company in general.

One of the most important requirements elicited from the case company was the importance of not letting some tests be left without execution for long periods. Having every test case be executed at least once in a few regression suite executions was deemed important for a TCP solution. An ML or Bayesian prioritization model that aims to maximize early fault detection would not natively consider this requirement. Executing test cases that the model is almost certain will not expose a fault would most likely lower the fault detection capability of the model. Making an ML or Bayesian model take this into consideration is not an easy task. However, rule-based prioritization could easily accommodate this requirement by simply prioritizing test cases when it has been a long time since that test was previously executed. This together with the surprisingly high performance of the heuristic prioritization on the company data makes a strong case for SMEs to employ heuristic or rule-based initially to accommodate both the requirements around a TCP solution and to increase early fault detection capabilities.

5.6 Focus on historical data sources

As presented by some early references, historical information seemed to provide strong indications for predicting future faults. According to Ramirez et al. [39], as well as experiences during this project, historical information seem often highly effortless to extract from a build system used for running nightly tests. This goes very well together with the automatic collection and modeling that the company had as a requirement since often the extraction of historical and verdict data can be automatically extracted and fed to a model for processing. This is therefore a benefit of special use to SMEs which may not already have an established procedure for collecting data sources for TCP.

For further improvement and sophistication of the models, a larger set of data sources could be employed. As noted in section 4.1.2, the TEPIA Taxonomy was used to guide the selection of data sources in this project. A list of the selected, as well as disregarded data sources, was presented. For the case company specifically, further investigation into the use of metrics from the third attribute category could be conducted to improve the TCP models. Specifically, the Dependency and the Similarity attribute group could with additional work be implemented at the case company because the required information sources were able to be extracted from the existing test suite. Many attributes from the attributes groups that were used in this project but disregarded in favor of other more readily available metrics could also have been used to improve the TCP models. Such attributes include allocation time of the total suite, (time-) cost of the test cases, resource utilization, age of the test cases, test code size, and change information regarding the test cases.

5.7 Events and Stability

As discussed in section 4.4, the models performed slightly differently between different parts of the dataset. For example, on the closed source dataset, all models showed lower APFD during the middle portion of the dataset, and higher APFD towards the edges. It is difficult to interpret what the reasons for this instability of the models are. Because the models are based on analyzing the historical outcomes of test cases, the temporary loss of performance in the models may be due to events that affect the SUT and testing practices in different ways. One kind of event could be code changes to the SUT: new features are added or old code is removed, causing test cases that used to find no faults to suddenly find many faults, or high fault detecting tests to suddenly find no faults. Changes to test cases could also happen, causing test cases to behave differently than before. Because the model bases its prioritization on historical information, it may need several cycles to adapt to changes in the SUT. Something else that may impact the stability of the models is problems relating to the infrastructure or build systems execution of the regression suites. If some parts of the infrastructure fail (e.g., connection problems between software components, unpredictable hardware behavior, or hardware faults), some tests may suddenly fail, but due to temporary infrastructure issues instead of faults or bugs being found in the code.

5.8 Options regarding testing strategy

Companies may go through development phases with releases where the testing focus may need to be changed, for example, to test the new features more extensively. This need is something that Strandberg et. al. share in their paper, where they develop a prioritization model using weights that the testers can manually change to focus the regression testing on certain things which are necessary for the testers to focus on during the current state of the development process [26].

For the case company in this study, in a real-world application, priority over different aspects of the testing may be highly relevant. The company may also undergo different phases of the development process that could contribute to testers wanting to change or manipulate settings regarding how the prioritization is made. This is a tough challenge to solve using some more advanced statistical tools like Bayesian models or Machine Learning, but probably surprisingly simple for rule-based or heuristic prioritization algorithms. This presents further arguments that companies may choose a method where testers may apply weights to the prioritization engine depending on the current testing focus of the company, instead of a more sophisticated prioritization model which can provide strong predictions of tests finding faults only.

5.9 Threats to Validity

When conducting any research, it is important to pay attention to the different risks and sources of bias or error that could impact the validity or reliability of the results of the study. Acknowledging and discussing these risks are important steps towards producing accurate results and trustworthy conclusions. In the following sections I will discuss the main threats to validity I believe to be important in this project, as well as what preemptive measures were taken to reduce the effect of those threats.

5.9.1 Internal Threats to Validity

The internal validity relates to the extent to which the results can be attributed to the intervention applied, and not to some other extraneous or unforeseen factors. In this project, there are two parts to this threat. First, in order to make conclusions of the model's performance, the right type of model need to be tested. Bayesian models are a large class of statistic models that can be constructed and used in many different ways. There can be models that are constructed differently and that view the input dataset differently than the models developed during this project. Those different models may even perform test case prioritization better or worse than the models of this project. The existence of such models can only be discovered through conducting further research into the use Bayesian models for prioritizing test cases. To make sure that the models developed in this study are representative enough of Bayesian models to accurately answer the research questions, and thus to reduce the impact of this threat to internal validity, some steps were taken. First, a tried and tested method for developing Bayesian models were followed as part of the

development of the models in this project [45].

Second, the only dependency of the Bayesian model, beyond the construction of the model itself, is the input dataset. The use of statistical modeling to determine the priority ordering of the suite's test cases is based on discovering some patterns in the input dataset that can help us predict which test cases are more or less likely to fail in an upcoming execution. Therefore, the results and performance of the model is directly dependent on the input dataset and can not be evaluated on its own. The fact that the model needs to be evaluated on a specific set of tests is a problem of generalizability, and will be discussed further in the section on external threats to validity. Due to the model's dependency on the input dataset, there is a risk that the evaluation framework used does not accurately describe the model's performance, but instead is impacted by factors, such as characteristics of the input dataset. Mitigation of this risk can be done by using an evaluation measurement that can test the model's performance on any dataset. The ubiquitous evaluation measurement for test case prioritization models that are based on historical data, called APFD, was used to this end. This measurement only compares the output (the test case ordering) produced by the model to the actual outcomes of the tests for that specific execution, and therefore limiting its dependency to the model. There are a few other very similar measurements that could have been used, such as NAPFD or APFDc, both of which has their utility. APFDc is not applicable in this project because the Cost (the lowercase c part of the name) was not of interest for the research questions nor used as a feature in the model. NAPFD could have been used for a more accurate reading in case of many missing test cases per execution, but it not as widely applied in other research so it was left out in preference of the APFD measurement and the generalizability its high prevalence can provide.

5.9.2 External Threats to Validity

Because the model performance is dependent on the dataset used, it is a large threat to external validity and to the ability to generalize the results of this study. One way to combat this, and the way that I have chosen in this project was to test the model on more than one dataset. This means that the models were first trained and tested on the company specific dataset, then the same models (sometimes with an alteration to match the input dataset) were trained and evaluated on an open-source dataset that has been used in many other studies. This should make the results more generalizable and comparable to the results of other studies. However, the two different datasets share some similarities regarding their characteristics, such as the total percentage of fail rate, number of data-points, and the number of test cases. Because models are dependent on the input dataset, both for training and evaluation, it is not clear that the developed models will perform equally well on datasets with radically different characteristics, such as having a lower or higher fault rate of the tests, lower number of test cases, or concealing other patterns and reasons for test case failures. These patterns may be easier for some models to pick up than others. The only way to investigate the impact of dataset characteristics on the model would be to apply and evaluate them on radically different datasets.

Furthermore, another threat to external validity is the fact that the models needed to change slightly between the two datasets. This is due to the fact that some features were collected in one dataset but not the other. This introduces another variable to the test, which may contribute to the performance of the models and the results of this study. This risk was mitigated by keeping the Bayesian models as similar as possible, only opting out of the missing feature of the dataset.

5.9.3 Construct Validity

Construct validity refers to the extent to which the measurements used in the study accurately reflects the concepts or constructs being studied. There are several measurements and evaluation frameworks used throughout this study, each aiming at successfully capturing the performance of the developed models along different dimensions. The different dimensions have slightly different reasons for study, and the related evaluation frameworks have different strengths and weaknesses regarding their measurements used. The main measurement for evaluating the performance of a test case prioritizing model, both in this study and in many other studies in this field, is the use of the APFD metric. The APFD metric is a clearly defined objective measure of a model's ability to predict faults of test cases on a particular execution cycle of a regression suite. It has been used in many studies previously, and there are varieties to it that capture the model's performance under uncommon circumstances. This metric was selected to mitigate the risk of having an imprecise or ill-defined measurement for the model performance.

However, due to the way this study investigates the requirements of the case company, and evaluating the developed models according to these requirements, a set of new evaluation frameworks were developed ad hoc. Because these evaluation frameworks were developed during the course of this study, they are at risk of being imprecise or without a standardized definition. Furthermore, it is not obvious that these evaluation frameworks would translate correctly to different organization with different views of how these requirements should be measured. Because the focus of this study and it's generated evaluation frameworks are SMEs and similar organizations, they do not necessarily need to generalize to other kind of organizations. Still, in order to make the constructs as valid as possible, three different steps were taken. Action research was followed, an adapted version of GQM was applied, and the First, the action research methodology was followed in order to have a clearly structured and repeatable way of developing the evaluation framework. Second, an adaptation of the GQM approach was used to generate the new evaluation frameworks. Third, the evaluation frameworks were developed and evaluated in an iterative process together with the company supervisors and practitioners that were the intended interpreters and consumers of the metrics. These three steps all contributed to mitigating the threats to construct validity in this study.

6

Conclusions

The increasing costs of regression testing is an important issue in software development as an organization's testing process becomes more sophisticated. The effects of this increasing cost can be observed in longer regression suite executions, taking hours or even days to complete, and thus slowing developer feedback. One of the common techniques used to tackle this issue is the use of test case prioritization. The common methods of implementing test case prioritization are causing a dilemma for especially small to medium-sized companies when choosing which method to pursue. On one hand, Heuristic models may prove too simplistic to make good enough predictions of future test execution outcomes, and on the other, Machine Learning models may require more training data than the company currently possesses. This thesis aims to solve this dilemma by implementing Bayesian models instead, which can provide strong predictive capabilities with lower data requirements than Machine Learning models. To this end, I have investigated the requirements of a small to medium-sized company regarding the implementation of test case prioritization. Furthermore, a comparison between Bayesian models, a Heuristic model, and a Machine Learning model was made in terms of the requirements elicited from the case company. An analysis of the specific data size requirements of the Bayesian models was also conducted.

The evidence shows that Bayesian models can have an advantage over the Heuristic model when it comes to early fault detection. The results also show that as little as seven days' worth of data can parallel the performance of the same Bayesian model trained on the complete dataset, while also requiring much less than the 60 days required by a Machine Learning model. On large datasets, the Machine Learning models still seem to perform slightly better than Bayesian models. However, while Bayesian models require very little historical data to perform test case outcome prediction, it may not be enough for the challenges faced by small to medium-sized companies during common business operations. Software releases and other irregular activities may require the ability to prioritize the regression suite and testing efforts in general in ways not provided by sophisticated statistical models. To ensure a total test coverage over time, prioritization of critical functionality of the system, and the ease of collecting data and model execution, small to medium-sized companies may receive larger benefits by designing a Heuristic or rule-based model that better suits the business needs.

Bibliography

[1] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 2, pp. 1–43, 2022.

[2] A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. C. Briand, "Scalable and accurate test case prioritization in continuous integration contexts," *CoRR*, vol. abs/2109.13168, 2021.

[3] Bayes Theorem, "Bayes theorem — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/wiki/Bayes%27_theorem`, 2022. [Online; accessed 09-September-2022].

[4] Markov Chain Monte Carlo, "Markov chain monte carlo — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/wiki/Markov_chain_Monte_Carlo`, 2022. [Online; accessed 09-October-2022].

[5] Artificial Neural Network, "Artificial neural network — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/wiki/Artificial_neural_network`, 2022. [Online; accessed 09-September-2022].

[6] M. Staron, "Action research as research methodology in software engineering," in *Action Research in Software Engineering*, pp. 15–36, Springer, 2020.

[7] R. Vasa, J.-G. Schneider, and O. Nierstrasz, "The inevitable stability of software change," in *2007 IEEE International Conference on Software Maintenance*, pp. 4–13, IEEE, 2007.

[8] Bohner, "Impact analysis in the software change process: a year 2000 perspective," in *1996 Proceedings of International Conference on Software Maintenance*, pp. 42–51, 1996.

[9] A. Labuschagne, L. Inozemtseva, and R. Holmes, "Measuring the cost of regression testing in practice: A study of java projects using continuous integration," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, (New York, NY, USA), p. 821–830, Association for Computing Machinery, 2017.

[10] H. K. N. Leung and L. J. White, "Insights into regression testing (software testing)," *Proceedings. Conference on Software Maintenance - 1989*, pp. 60–69, 1989.

[11] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[12] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test case prioritization: an empirical study," in *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*, pp. 179–188, 1999.

[13] K. Fischer, F. Raji, and D. Onaszko, "Software retest techniques.," tech. rep., COMPUTER SCIENCES CORP FALLS CHURCH VA, 1982.

[14] J. Hartmann and D. J. Robson, "Techniques for selective revalidation," *IEEE Software*, vol. 7, no. 1, pp. 31–36, 1990.

[15] T. Y. Chen and M. F. Lau, "Dividing strategies for the optimization of a test suite," *Information Processing Letters*, vol. 60, no. 3, pp. 135–141, 1996.

[16] J. Offutt, J. Pan, and J. M. Voas, "Procedures for reducing the size of coverage-based test sets," in *Proceedings of the 12th International Conference on Testing Computer Software*, pp. 111–123, ACM Press New York, 1995.

[17] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong, "An empirical study of the effects of minimization on the fault detection capabilities of test suites," in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pp. 34–43, IEEE, 1998.

[18] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," in *1995 17th International Conference on Software Engineering*, pp. 41–41, 1995.

[19] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "An empirical study of junit test-suite reduction," in *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, pp. 170–179, 2011.

[20] W. Wong, J. Horgan, A. Mathur, and A. Pasquini, "Test set size minimization and fault detection effectiveness: a case study in a space application," in *Proceedings Twenty-First Annual International Computer Software and Applications Conference (COMPSAC'97)*, pp. 522–528, 1997.

[21] D. Winkler, S. Biffl, D. Mendez, and J. Bergsmann, *Software Quality: Quality Intelligence in Software and Systems Engineering: 12th International Conference, SWQD 2020, Vienna, Austria, January 14–17, 2020, Proceedings*, vol. 371. Springer Nature, 2020.

[22] H. Agrawal, J. Horgan, E. Krauser, and S. London, "Incremental regression testing," in *1993 Conference on Software Maintenance*, pp. 348–357, 1993.

[23] G. Rothermel and M. J. Harrold, "Selecting tests and identifying test coverage requirements for modified software," in *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, pp. 169–184, 1994.

[24] G. Rothermel and M. Harrold, "A safe, efficient algorithm for regression test selection," in *1993 Conference on Software Maintenance*, pp. 358–367, 1993.

[25] F. Vokolos and P. Frankl, "Empirical evaluation of the textual differencing regression testing technique," in *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pp. 44–53, 1998.

[26] P. E. Strandberg, D. Sundmark, W. Afzal, T. J. Ostrand, and E. J. Weyuker, "Experience report: Automated system level regression test prioritization using multiple factors," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 12–23, 2016.

[27] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study," in *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*, pp. 179–188, IEEE, 1999.

[28] S. M. Arabbaygi, "A bayesian framework for software regression testing," *Master of Applied Science in Electrical and Computer Engineering, University of Waterloo, Canada.*, 2008.

[29] J. Parejo, A. B. Sánchez, S. Segura, A. Ruiz-Cortés, R. Lopez-Herrejon, and A. Egyed, "Multi-objective test case prioritization in highly configurable systems: A case study," *Journal of Systems and Software*, vol. 122, pp. 287–310, 2016.

[30] H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *2005 International Symposium on Empirical Software Engineering, 2005.*, pp. 17–18, 2005.

[31] B. Korel, L. H. Tahat, and M. Harman, "Test prioritization using system models," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*, pp. 559–568, IEEE, 2005.

[32] B. Korel, G. Koutsogiannakis, and L. H. Tahat, "Application of system models in regression test suite prioritization," in *2008 IEEE International Conference on Software Maintenance*, pp. 247–256, 2008.

[33] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pp. 119–129, 2002.

[34] S. Mirarab and L. Tahvildari, "A prioritization approach for software test cases based on bayesian networks," in *Fundamental Approaches to Software Engineering* (M. B. Dwyer and A. Lopes, eds.), (Berlin, Heidelberg), pp. 276–290, Springer Berlin Heidelberg, 2007.

[35] J. A. Prado Lima and S. R. Vergilio, "Test case prioritization in continuous integration environments: A systematic mapping study," *Information and Software Technology*, vol. 121, p. 106268, 2020.

[36] D. Marijan, A. Gotlieb, and S. Sen, "Test case prioritization for continuous regression testing: An industrial case study," in *2013 IEEE International Conference on Software Maintenance*, pp. 540–543, 2013.

[37] A. Sharif, D. Marijan, and M. Liaaen, "Deeporder: Deep learning for test case prioritization in continuous integration testing," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 525–534, 2021.

[38] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 12–22, 2017.

[39] A. Ramírez, R. Feldt, and J. R. Romero, "A taxonomy of information attributes for test case prioritisation: Applicability, machine learning," *ACM Transactions on Software Engineering and Methodology*, 04 2022.

[40] R. O'Brien, "An overview of the methodological approach of action research," *Faculty of Information Studies, University of Toronto*, 1998.

[41] European Commission, "Commission recommendation of 6 may 2003 concerning the definition of micro, small and medium-sized enterprises (text with eea relevance) (notified under document number c(2003) 1422)." `http://data.europa.eu/eli/reco/2003/361/oj`.

[42] GQM, "Gqm — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/wiki/GQM`, 2022. [Online; accessed 09-October-2022].

[43] Python Org., "Python: General-purpose programming language." `https://www.python.org/`, 2022. [Online; accessed 09-September-2022].

[44] The R Project, "R: Statistical programming language." `https://www.r-project.org/`, 2022. [Online; accessed 09-September-2022].

[45] C. A. Furia, R. Feldt, and R. Torkar, "Bayesian data analysis in empirical software engineering research," *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1786–1810, 2019.

[46] P. Padmnav, G. Pahwa, D. Singh, and S. Bansal, "Test case prioritization based on historical failure patterns using abc and ga," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pp. 293–298, IEEE, 2019.

A

Appendix 1

Figure A.1: APFD measurements of the Bayesian models with different data size and the Heuristic model on the open source dataset.

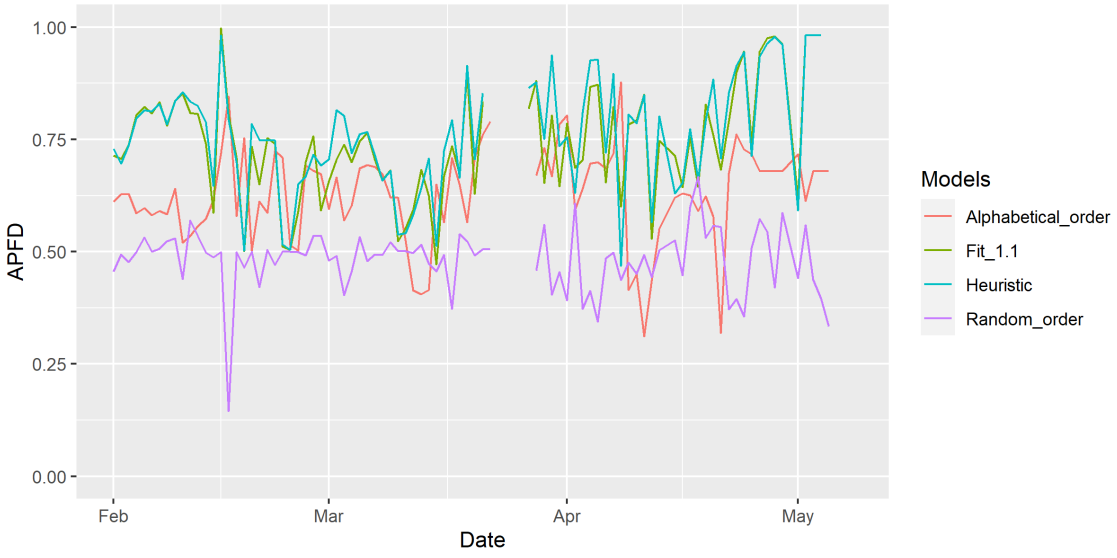

Figure A.2: APFD measurements of the Heuristic model, the initial Bayesian model, and the baselines alphabetical and random order.