Embracing Unification: A Comprehensive Approach to Modern Test Case Prioritization

Andreea Vescan[a], Radu Găceanu[b] and Arnold Szederjesi-Dragomir[c]
Computer Science Department, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

Keywords: Regression Testing, Test Case Prioritization, Unification, Requirements, Test Cases, Faults.

Abstract: Regression testing is essential for software systems that undergo changes to ensure functionality and identify potential problems. It is crucial to verify that modifications, such as bug fixes or improvements, do not affect existing functional components of the system. Test Case Prioritization (TCP) is a strategy used in regression testing that involves the reordering of test cases to detect faults early on with minimal execution cost. Current TCP methods have investigated various approaches, including source code-based coverage criteria, risk-based, and requirement-based conditions. However, to our knowledge, there is currently no comprehensive TCP representation that effectively integrates all these influencing aspects. Our approach aims to fill this gap by proposing a comprehensive perspective of the TCP problem that integrates numerous aspects into a unified framework: traceability information, context, and feature information. To validate our approach, we use a synthetic dataset that illustrates six scenarios, each with varying combinations of test cases, faults, requirements, execution cycles, and source code information. Three methods, Random, Greedy, and Clustering, are employed to compare the results obtained under various time-executing budgets. Experiment results show that the Clustering method consistently outperforms Random and Greedy across various scenarios and budgets.

1 INTRODUCTION

Regression testing is a crucial aspect of software quality assurance, designed to verify that the software continues to function correctly after modifications. It involves reexecuting test cases from a test suite to ensure that previously developed and tested software still performs as expected after changes such as enhancements, patches, or configuration changes. However, given that software projects often operate under time and resource constraints, executing the entire test suite is often impractical. This has led to the development of regression test selection strategies, with the aim of identifying a subset of the test suite that is likely to detect potential faults, while minimizing the cost of testing (Engström et al., 2010). Despite its practical importance, regression testing remains a challenging task due to its inherent trade-off between effectiveness and efficiency, as well as difficulties in accurately capturing the dependencies in complex software systems. There is an increase in the number of publications (Singh et al., 2023) related to it, the main publication fora being conferences, followed by journals. Thus, the regression testing problem is worth exploring, being an omnipresent one during the software development life cycle. It is essential to find efficient and effective test case executions with the aim of obtaining higher qualitative systems.

Regression test selection techniques are classified by researchers (Yoo and Harman, 2010) into three types: Test Suite Minimization (TSM), Test Case Selection (TCS), and Test Case Prioritization (TCP). Testing was mainly centered on structural coverage as in the study (Rothermel et al., 1999), few investigations (Salehie et al., 2011), (Srikanth et al., 2016) considered the relations between functional requirements and faults, and fewer examined dependencies between requirements. Test Case Prioritization is an essential strategy in regression testing with the aim of ordering test cases so that the most important ones are executed first, detecting potential defects early in the testing process. Testing was mainly centered on structural coverage as in the study (Rothermel

[a] https://orcid.org/0000-0002-9049-5726
[b] https://orcid.org/0000-0002-0977-4104
[c] https://orcid.org/0000-0002-1106-526X

Vescan, A., Găceanu, R. and Szederjesi-Dragomir, A.
Embracing Unification: A Comprehensive Approach to Modern Test Case Prioritization.
DOI: 10.5220/0012631000003687
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024)*, pages 396-405
ISBN: 978-989-758-696-5; ISSN: 2184-4895
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

et al., 1999), few investigations (Salehie et al., 2011) considered the relations between functional requirements and faults, and fewer examined dependencies between requirements.

In this paper, we propose a modern testing perspective of the Test Case Prioritization problem by outlining influencing factors on the design of the TCP, with particular applied contexts and adequate applied methods. First, we proposed the use of requirements and the connection to the source code and implemented test cases. These traceabilities, namely, requirements-to-code and code-to-test cases, will allow the two different actions to be operationalized when a change occurs: if a requirement is changed (or improved), this should trigger the test cases associated with it through the source code; and if the change occurs in the source code, this should trigger again the associated test cases. Second, the context when applying TCP may influence the decision on the approach used, that is, the test suite being part of a unit test set, an integration test set, or a system test set. Third, the available information on the system under test (SUT) and the associated artifacts, namely, faults, test cases, time constraints, influence the decision on what method to be used to maximize the detection of potential faults while minimizing the execution cost. Moreover, we have proposed a synthetic dataset with various information available and discussed the solutions for 6 different scenarios using 3 methods to construct the TCP solution.

The paper is organized as follows:

Section 2 outlines the context of the investigation, along with the motivation, TCP background, and state-of-the-art approaches. Our unified approach to modern TCP is outlined in Section 3, while the case study with the synthetic dataset and the 6 scenarios for building TCP solutions are provided in Section 4. The potential impact on researchers and practitioners is provided in Section 5. Section 6 discusses the threats to validity with respect to the investigation of the research. The last section concludes our investigation and also provides a future research agenda.

2 THEORETICAL BACKGROUND ABOUT TEST CASE PRIORITIZATION

This section presents relevant theoretical and technical background elements, starting with the context and the motivation behind this research and continuing with the theoretical elements of Test Case Prioritization and related work.

2.1 Context and Motivation

The Software Development Life Cycle (SDLC) (Burnstein, 2010) is a process used by the software industry to design, develop and test high-quality software. It consists of a detailed plan describing how to develop, maintain, replace, and alter or enhance specific software. The process is divided into different phases, each phase having its specific goals and deliverables.

The traditional SDLC, often referred to as the Waterfall model, has the following stages: Requirements Gathering and Analysis, System Design, Implementation, Integration and Testing, Deployment, and Maintenance. The Waterfall model follows a strict top-down approach, with each phase being entirely completed before the next one begins and not having the possiblity to return to a previous phase, thus changes are difficult to implement once the project is underway. Agile development was introduced as a response to the drawbacks of the Waterfall model and other traditional software development methodologies. It approaches software development in incremental cycles, allowing more flexibility, collaboration, and adaptability. The Agile SDLC model follows the same steps as traditional SDLC, but the execution of these stages is fundamentally different. Instead of a sequential design process, the Agile methodology follows an incremental approach. Developers start with a simplistic project design and then begin working on small modules.

The traditional SDLC, often referred to as the Waterfall model, strictly follows a sequence of stages from Requirements Gathering to Maintenance, with no room for revisiting completed phases. This rigidity can make changes challenging once the project starts. Agile methodologies, developed in response to these issues, adopts an incremental cycle approach to software development, promoting flexibility and collaboration. Although the Agile SDLC model has the same stages as Waterfall, its execution differs, focusing on starting with a simple design and progressively working on small modules.

In February 2001, an Agile Manifesto (Apke, 2015) was drafted and signed. It is short, less than 500 words in length, and simply contains a set of four values and 12 Agile principles. The core values are: individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a project plan. The principles are related to these values and defines some guidelines on how to collaborate with customers and respond to changes efficiently.

Agile methodologies offer several improvements on handling requirements compared to traditional waterfall methodologies: **Flexibility and Adaptability** - allow for changes in requirements at any point in the project, **Frequent Inspection and Adaptation** - emphasize frequent inspection and adaptation, which allows the team to adjust requirements as needed, and to immediately address any issues or changes, **Customer Involvement** - promote close customer collaboration, meaning that requirements can be clarified, validated, and adjusted continuously based on customer feedback, **Prioritization** - requirements are typically prioritized based on value, risk, and necessity, allowing the most important features to be developed first, **Incremental Delivery** - deliver software in small, functional increments. This allows the team to gather feedback and adapt requirements as the project progresses, rather than attempting to define all requirements upfront.

Agile methodologies enhance requirement handling via: *Flexibility and Adaptability*, enabling changes throughout the project; *Frequent Inspection and Adaptation*, allowing constant requirement modifications; *Customer Involvement*, promoting continuous feedback-based requirement adjustments; *Prioritization*, based on value, risk, and necessity to develop essential features first; and *Incremental Delivery*, supporting feedback and adaptation through regular increments of functional software.

Some of the Agile methodologies used are: Scrum (Schwaber and Sutherland, 2017), Kanban (Anderson, 2010), Extreme Programming (XP) (Beck, 2000), and Lean (Poppendieck and Poppendieck, 2003) software development.

A new way of handling customer functionality specifications has also emerged: user stories, traditionally refer to a detailed specification of what a system should do. They are usually more formal and technical in nature. In traditional waterfall methodologies, these are often written up front and then handed off to the development team. They may include functional requirements (what the system must do), non-functional requirements (qualities the system must have, like performance or security), and constraints (limits on the design of the system). , on the other hand, are a tool used in Agile methodologies to represent a small, independent piece of business value that a team can deliver. A user story is usually written from the perspective of an end user and describes a piece of functionality that is valuable to that user. User stories are intentionally written in a less formal and more conversational language. The classic format of a user story is “As a [type of user], I want [an action] so that [a benefit or value]”. They are often accompanied by acceptance criteria that define the boundaries and expectations of the story. Although both user stories and requirements aim to guide the development process, the key differences between them are in their format, their level of detail, and how they are used in the development process.

As we can observe, the introduction of Agile methodologies radically changed SDLC, and while TCP has already seen some small adaptations, we believe that further refinements are needed. We need a way to address the different ways of describing functionalities, frequent changes to them, prioritization and reprioritization of given features, and incremental delivery which, although does not have a direct impact on test cases, but because of resource constraints (especially time, but maybe computing resources too) it can influence how we prioritize our tests.

To conclude, the shift to Agile methodologies has significantly impacted SDLC, and while TCP has already seen some small adaptations, we believe that further refinements are needed. We need to address the different ways of describing functionalities, the frequent changes to them (**change of requirements**), prioritization and reprioritization of given features (**requirement prioritization**), and **incremental delivery**, which can influence the way we prioritize our tests.

2.2 Test Case Prioritization Background

There are many ways to define Test Case Prioritization (TCP), but no one definition covers all the requirements, contexts, and information that play a part in or can affect this process. According to (Graves et al., 1998), the Test Case Prioritization problem can be formally defined as follows:

Definition 1. ***Test Case Prioritization (Graves et al., 1998):*** *a test suite, T, the set of permutations of T, PT; a function from PT to real numbers, f. The goal is to find T ∈ PT such that:*

$$(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')] \quad (1)$$

In Definition 1, the function f assigns a real value to a permutation of T according to the test adequacy of the particular permutation.

Since this definition lacks time constraints for executing test suites, Time-limited Test Case Prioritization (TTCP) is used in papers like (Spieker et al., 2017) and (Vescan et al., 2023b) in order to enhance the TCP problem by adding this constraint. TCP was associated with improvement testing in (Juristo and Moreno, 2004), with the aim of increasing the fault detection rate or decreasing the cost and time of the prioritization process. The authors emphasize that even if TCP is mainly applied for regression testing, it can also be applied at the initial phase of testing or even for software maintenance. A definition of TCP that considers requirements and dependencies in the prioritization process was proposed in (Vescan et al., 2021). In (Paterson et al., 2019), the authors employ the use of defect prediction for TCP. Consequently, given the diverse range of existing TCP approaches, each addressing different facets of the various contextual components involved in the TCP process, it becomes important to establish a unified framework for TCP. This comprehensive approach would ensure consistency and clarity among the ways researchers tackle this complex process.

In order to evaluate the effectiveness of a TCP approach, several metrics are proposed. For example, APFD (Average Percentage of Faults Detected) (Pradeepa and VimalDevi, 2013) measures the effectiveness of a test suite in detecting faults as early as possible during the testing process and is defined as follows:

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{n \times m} + \frac{1}{2n} \quad (2)$$

where: TF_i is the position of the first test case that detects the i-th fault, m is the total number of faults detected by the test suite and n is the total number of test cases.

An extension of APFD to incorporate the fact that not all test cases are executed and failures can be undetected is the Normalized APFD (NAPFD) (Qu et al., 2007):

$$NAPFD = p - \frac{\sum_{i=1}^{m} TF_i}{n \times m} + \frac{p}{2 \times n} \quad (3)$$

where p is the number of faults detected by the prioritized test suite divided by the number of faults detected in the entire test suite. Variations of the aforementioned metrics include, for example, incorporating weights associated with the faults based on their severity.

2.3 State of the Art on Test Case Prioritization

Test case selection and prioritization is an intensively investigated topic in the literature, and it has been approached in several ways by addressing different optimization goals and by using a plethora of techniques in doing so (Pan et al., 2022; Bertolino et al., 2020; Khalid and Qamar, 2019; Kandil et al., 2017; Almaghairbe and Roper, 2017; Medhat et al., 2020). According to the machine learning technique involved, test case selection and prioritization studies could be classified into the following categories: supervised learning, unsupervised learning, reinforcement learning, and natural language processing (Pan R., 2022).

Recent studies apply reinforcement learning (RL) techniques to Test Case Prioritization (TCP) in Continuous Integration (CI) due to the ability of RL to adapt to the dynamic nature of CI without need for full retraining. Once trained, the RL agent can evaluate a test case, assign it a score, and use that score to order or prioritize the test cases. Although most of the RL studies consider only the execution history to train their agent, we have found one study, namely (Bertolino et al., 2020), where, in addition to the execution history, code complexity metrics have also been used. In their very comprehensive research paper, the authors (Bertolino et al., 2020) evaluate and compare 10 machine learning algorithms, focusing on the comparison between supervised learning and reinforcement learning for the Test Case Prioritisation (TCP) problem. Experiments are performed on six publicly available datasets, and based on the results, the authors propose several guidelines for applying machine learning to regression testing in Continuous Integration (CI).

In (Bertolino et al., 2020), the authors also propose some new metrics (Rank Percentile Average (RPA) and Normalized-Rank-Percentile-Average (NRPA)) to evaluate how close a prediction ranking is to the optimal one. Unfortunately, in (Pan R., 2022) it is explained in detail that the NRPA metric is not always suitable. Nevertheless, the paper from (Bertolino et al., 2020) remains a very thorough and elaborate research study, and the authors from (Pan R., 2022) include it in a short list of research papers that are actually reproducible.

The main idea of clustering in the TCP context is the assumption that test cases with similar characteristics, such as coverage and other attributes, are likely to have comparable fault detection abilities. Many articles, such as, for example, (Khalid and Qamar, 2019) utilized the K-means algorithm or variations of the K-means algorithm. The Euclidean distance is the most commonly used similarity measure in clustering, but there are some attempts to use other similarity measures, like the Hamming distance. The paper of (Kandil et al., 2017) is an example in this sense. In this study, the authors represent coverage information for a test case as binary strings, where each bit indicates whether or not a source code element has been covered by a test. An interesting approach is proposed in (Almaghairbe and Roper, 2017) where the authors use clustering for anomaly detection of passing and failing executions. The key idea is that failures tend to be grouped into small clusters, while passing tests will group into larger ones, and their experiments suggest that their hypothesis is valid.

Supervised learning is probably one of the most commonly used ML techniques to address TCP as a ranking problem. Specifically, these techniques typically use one of three distinct ranking models for information retrieval: pointwise, pairwise, and listwise ranking. In (Bertolino et al., 2020), the authors used a state-of-the-art ranking library (Dang and Zarozinski, 2020) and evaluated the effectiveness of Random Forest (RF), Multiple Additive Regression Tree (MART), L-MART, RankBoost, RankNet, Coordinate ASCENT (CA) for TP. Their results show that (MART) is the most accurate model. Although supervised learning can achieve high accuracy, a major issue is that a full dataset should be available before training. In order to support incremental learning, the model often needs to be rebuilt from scratch, which is time-intensive and hence not quite ideal for CI.

The use of NLP seems quite limited. The core motivation to apply NLP techniques is to exploit information in textual software development artifacts (e.g., bug description) or source code that is treated as textual data. In general, the idea is to transform test cases into vectors and then to compute the distance between pairs of test cases. The test cases are then prioritized using different strategies. An interesting approach is proposed in (Medhat et al., 2020), where NLP is used to preprocess the specifications that describe the components of the system under test. Then they used recurrent neural networks to classify the specifications into the following components: user device, protocols, gateways, sensors, actuators, and data processing. On the basis of this classification, test cases belonging to these standard components were selected. Then, they used search-based approaches (genetic algorithms and simulated annealing) to prioritize the selected test cases.

3 THE UNIFIED APPROACH TO MODERN TCP

Our disrupted proposal for the Test Case Prioritization problem is outlined next. We discuss the factors that influence it, namely, considering traceability information, emphasizing the used contexts, the applied methods, and the available information. Figure 1 shows the main elements of the TCP problem that we consider.

3.1 Traceability Perspective

The first aspect related to the TCP unification approach refers to **traceability**. In this regard, two of the aspects presented above related to the Agile perspective are included, namely, *change of requirements* and *requirement prioritization*.

The aspect related to *the change of requirements* is explained. To consider it, we propose using requirements and the link to the source code and to the test cases. These connections allow two different actions to be performed when a change occurs: a change in requirements triggers the associated test cases through the connection with source code, and a change in source code triggers the linked test cases. In this second action, further analysis could be performed to identify requirements that are difficult to implement. Several previously investigated approaches use the change of requirements aspect as in (Tiutin and Vescan, 2022). *Three other considerations* are next discussed:

- We propose to include *a probability of change for a requirement*, thus having more test cases for a requirement that is likely to change often (and also consider the proper design of the test cases such that no test smells exist). Therefore, the impact on the designed and implemented test cases should be considered.
- *Dependencies between requirements* play an important role in the regression testing process and the method used.
- Another important aspect regarding *requirement - test case traceability* refers to the way the connection is considered: from $0, 1$ to whether the requirement is implemented in the test case or not, to a *degree of membership*. Thus, a test case may partially cover a requirement.

The second view regarding **traceability** refers to *the requirement prioritization*, meaning that some requirements are prioritized in the development process, which requires consistent testing. In this respect, using the dependencies between requirements, when a change is performed in one of the prioritized requirements, not only directly associated test cases should be executed, but also the requirements that are dependent on. This aspect remains to be further analyzed, depending on the tested context (unit testing, incremental testing, or system testing), determining whether to store the specific element or compute based on the available information.

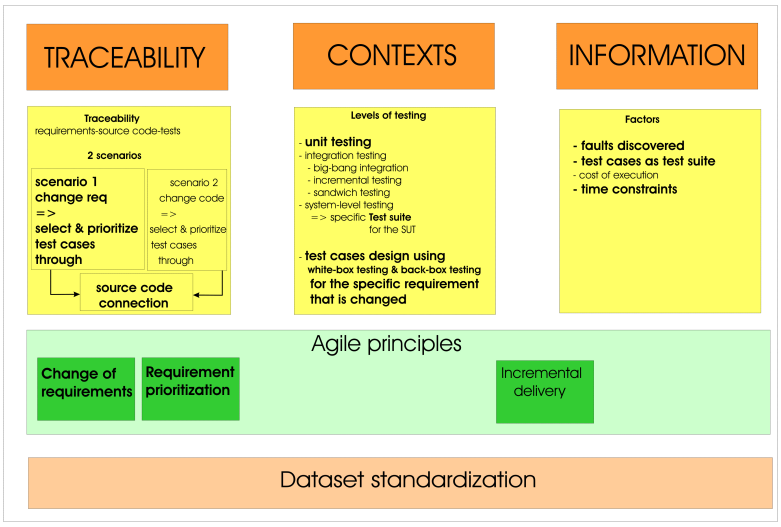

Figure 1: Overview of the approach.

3.2 Contexts Perspective

The second aspect related to the TCP unification approach refers to **the contexts of testing**, more specifically to the level of testing, namely, the type of test cases in the test suite: unit tests, integration tests, system-level tests. We argue that the type of test cases may influence the method used to prioritize them. When a change is made on a specific source code, it may be the case to just run the unit tests associated with that functionality; however, when changing a source code part, depending on the associated requirements, it may be the case that an end-to-end test case suite should be executed. This is included as part of the *incremental delivery* from the Agile perspective.

3.3 Information Perspective

The third aspect related to the TCP unification approach, which is still related to the Agile perspective on the *incremental delivery*, refers to the available **information**, such as faults, cost of execution, time constraints, cycles runs, and so on.

The available **information** on the system under test (SUT) and the associated artifacts, namely, faults, test cases, time constraints, influence the decision on what method to use to maximize the detection of potential faults while minimizing the execution cost. There is a need for the unification for the TCP problem, defining the theoretical framework (input and output as available information, the context of testing) that plays a role in the decision of the method used to prioritize (artificial intelligence methods as, for example, genetic algorithms, nature-inspired algorithms such as ant colony systems, or machine learning algorithms as random forest). Thus, the theoretical framework is common, but the applied method varies depending on the available information and context.

3.4 Scenario Example

Considering each perspective as outlined in Figure 1, a scenario may consider the requirement changes in the context of unit testing with information regarding test cases, faults, and time constraint (execution budget) to execute regression testing. Another scenario might be when the requirements are not available but the results of test case execution in various cycles in the continuous integration environments are available.

Therefore, we advocate that the investigation of TCP should consider the traceability requirement-code-test-fault, the context, and the available information on the SUT.

Furthermore, a particular general dataset is needed for scientific research on the best TCP approaches. As stated above, there exist various datasets (Software-artifact Infrastructure Repository (SIR) (SIR, nd), Defect4J (Just et al., 2014), JTeC (JTeC, nd), GitHub-based mining repositories (GitHub, nd)) constructed with specific information about the system under test. Standardization of the dataset format is needed to facilitate the research of various proposals in the same context. Various standardizations may exist depending on the context of testing, namely, testing level and available information on the SUT.

4 CASE STUDY

A synthetic dataset was constructed as provided in Table 1 containing the information between test cases, requirements and faults, the relationship between test cases and source code, and the links between test cases and cycles, and test case duration.

The synthetic dataset was constructed to be used for the initial evaluation of our proposal considering all the perspectives outlined above about TCP: traceability, context, and available information. In building the dataset we have used various information from the existing datasets such as Software-artifact Infrastructure Repository (SIR) (SIR, nd), Defect4J (Just et al., 2014), JTeC (JTeC, nd), GitHub-based mining repositories (GitHub, nd).

For *requirements* and *source code* we considered value 1 if the test case is connected to the specific requirement or source code. For faults, we considered 1 if the test case failed, and thus found the specific fault. The meanings in the *cycles* column are: 0 if the test case was not run in those specific cycles, 0.5 if it was run but with pass, and 1 when the test case was run with a fail result.

In what follows, we will present the results obtained in various scenarios, having different information available as described in Table 2.

The scenarios are:

- *Scenario 1* (S1): test cases and faults,
- *Scenario 2* (S2): test cases, faults, and requirements, along with the specific requirements that were changed,
- *Scenario 3* (S3): test cases, faults, and the verdict of the last cycle,
- *Scenario 4* (S4): test cases, faults, and cycles, namely, the rounded average verdict cycle,
- *Scenario 5* (S5): test cases, faults, requirements, and cycles,
- *Scenario 6* (S6): test cases, faults, source code, and cycles, namely, the rounded average verdict cycle.

The methods used for the experiments are:

- *Random*: a random generation of the test case is performed.
- *Greedy*: test cases are selected first based on the faults and then based on the other specific conditions as in each scenario.
- *Clustering* (Găceanu et al., 2022): test cases are added in different clusters based on the fault capabilities and based on each scenario-based conditions.

The metric used for the evaluation is the NAPFD (Qu et al., 2007), however, for each scenario, different information is incorporated into the formula. For example, in the requirements-based scenario, the faults that are considered are those linked to the requirements that are changed.

The TCP solutions obtained for each employed method are provided in Table 3. The experiments were performed with various time budget configurations, from 10 to 100. The majority of the best results obtained are for the Clustering method for almost all budgets. The best results are in bold in Table 3.

The obtained solutions for Greedy and Clustering are provided at this link along with the dataset (Vescan et al., 2023a). We present here, due to space limitation, only the solutions for the scenario *S5* for budget 50, for Greedy and Clustering.

The Greedy solution is: [10, 11, 5, 1, 9, 12, 13, 6, 3, 2, 15, 16, 7, 17, 4, 8, 14]. For the Clustering solution, the clusters are also emphasized: [1, 5], [11], [10, 12], [13, 9], [15], [14, 4, 8, 17, 7], [6, 16, 2, 3]. The analysis reveals that while the Greedy solution identifies the correct test cases sooner than the Clustering solution, it overlooks the duration of these tests. In contrast, the Clustering solution incorporates all relevant information, including test duration, thereby offering a more comprehensive and balanced solution.

5 POTENTIAL IMPACT ON RESEARCHERS AND PRACTITIONERS

The article has the potential to disrupt current practice on regression testing, more specifically on test case prioritization. This section details the implications of our contributions for both researchers and practitioners.

For the academic community, our study offers a new perspective on the problem of prioritizing test cases. We introduce a framework that unifies the components that are involved in TCP like artifact traceability, context (e.g. unit testing, integration testing, etc.), and information (e.g. faults, execution costs, etc.). In our experiments, we investigate several scenarios that demonstrate the benefits of a comprehensive approach to TCP.

For professionals in the field, our method could be integrated into their regression testing workflows. Implementing our approach as a plugin in the Integrated Development Environment (IDE), for example, could save significant time otherwise spent on manual test case selection after code changes take place. Addi-

Table 1: Synthetic dataset.

tc	requirements					faults							source code								cycles										d
	1	2	3	4	5	1	2	3	4	5	6	7	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	9	10	
1	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0.5	0.5	1	1	1	1	1	0.5	0.5	0.2
2	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0.5	0.5	1	0.5	0.5	0	0	0	0	0	0.5
3	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	0.5	0.5	0.5	0.5	0.5	0.5	0.8
4	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.3
5	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0.5	1	0.5	1	0.5	1	0.5	1	0.5	1	0.2
6	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0.5	0.5	1	1	0.5	0.5	1	0.6
7	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0.5	0.5	0.5	0.5	0	0	0	0.5
8	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0.5	0.5	0.5	0	0	0	0	0.5	0.5	0.5	0.4
9	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	0.5	0.5	1	0.3
10	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0.5	1	0.5	1	0.5	0.7
11	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0.5	1	0.8
12	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0.5	1	0.5	1	0.5	1	0.9
13	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0.5	0.5	1	0.1
14	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0.5	0.5	0.5	0.5	0	0.5	0.3
15	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	1	0.5	0.5	0.5	1	0.5	0.5	0.2
16	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0.5	0.5	1	0.5	0.5	1	0.8
17	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0.5	0.5	0.5	0.5	0.5	0.5	0	0	0	0	0.5

Table 2: NAPFD values for the Random, Greedy and Clustering approaches for various budget values.

S	TC	faults	reqs	c-lastV	c-avgV	code
S1	✓	✓				
S2	✓	✓	✓			
S3	✓	✓		✓		
S4	✓	✓			✓	
S5	✓	✓	✓		✓	
S6	✓	✓	✓		✓	✓

tionally, we encourage practitioners to consider new aspects of regression testing beyond traditional methods. By presenting novel ideas and future directions, our aim is to inspire them to consider other crucial aspects of regression testing that need to be addressed.

6 THREATS TO VALIDITY

Experiments may be susceptible to specific threats to validity, with research outcomes being affected by diverse factors. The following enumerates potential influences on the results obtained, along with the corresponding actions to mitigate them.

Internal. The primary internal validity threat is the use of the NAPFD metric. It is not designed for multi-criteria based evaluations, its main scope is to find faults as early as possible, without taking into consideration other elements like requirements or source code relations. In order to mitigate this, we have implemented an NAPFD metric that has as goal finding all testcases that expose faults and have the correct requirements and source code links. In the future, a new metric might be needed that can also evaluate partial results, which means, for example, that test cases that expose a fault and have the correct requirement should be higher in the priority than test cases that do not have the correct requirement.

Table 3: NAPFD values for the Random, Greedy and Clustering approaches for various budget values.

Scenario	Budget	Random	Greedy	Clustering
S1	10	13.54	21.43	**42.86**
	25	25.25	35.71	**57.14**
	50	39.29	64.29	**72.22**
	75	51.8	**77.27**	**77.27**
	80	53.93	79.17	**80.77**
	100	64.46	**85.29**	**85.29**
S2	10	8.64	**21.43**	**21.43**
	25	13.47	**28.57**	**28.57**
	50	21.2	**47.62**	45.71
	75	28.16	50.79	**51.95**
	80	29.59	51.43	**52.75**
	100	36.04	53.78	**53.78**
S3	10	9.95	**28.57**	**28.57**
	25	17.78	**42.86**	**42.86**
	50	27.15	56.12	**58.04**
	75	35.51	60.71	**62.5**
	80	36.96	61.69	**63.19**
	100	43.53	**65.13**	**65.13**
S4	10	8.29	21.43	**37.5**
	25	15.77	**42.86**	**42.86**
	50	24.62	**50**	**50**
	75	32.48	**51.95**	**52.04**
	80	33.73	**52.38**	52.04
	100	39.1	**53.78**	52.94
S5	10	5.7	21.43	**28.57**
	25	9.91	**42.86**	**42.86**
	50	17.74	50	**51.43**
	75	24.25	51.95	**53.06**
	80	25.66	52.38	**53.06**
	100	32.01	**53.78**	**53.78**
S6	10	3.48	**21.43**	**21.43**
	25	6.54	37.5	**38.57**
	50	11.94	40.18	**40.71**
	75	17.1	40.91	**41.21**
	80	18.05	41.07	**41.33**
	100	23.14	**41.6**	**41.6**

Construct. In the absence of a dataset that can capture all the information we need, the creation of a synthetic dataset was necessary. We have considered only the case where a test case exposes only one single fault, but in practice this might not always be true. For subsequent research, transforming existing datasets might be the correct way to go.

External. Our approach is validated only on a single dataset. Clearly, more datasets are needed to reliably demonstrate the potential impact of this approach. Provided that an effective method for transforming and repurposing existing datasets is identified, this should logically be the next course of action.

7 CONCLUSIONS AND FUTURE RESEARCH AGENDA

The Test Case Prioritization problem is defined and investigated under various conditions and with different available information on SUT. We have identified several challenges in the unification process that are outlined below, along with the research agenda, namely the identification and implementation of the research steps.

Traceability. The connection between requirements and source code and test cases is a challenge in the unification process. Up until now, to the best of our knowledge, the majority of TCP was considered under the second scenario that we outlined above, namely, a change in the source code to trigger the test cases. Few approaches considered requirements, in this regard the relation requirements to test cases being done through source code. Our research agenda on this aspect will investigate this traceability by linking first requirements to test cases using Behavior-Driven Development (BDD).

Context. An important aspect when performing regression testing is considering the test suite as part of a different level of testing: unit testing, integration testing, and system-level testing. In this challenge, our objective is to investigate the impact of different types of test suites on the method used to prioritize. The complexity of the test cases in the test suite and the number of test cases in the test suite may have an impact on the decision on the method used for prioritization.

Information. The available information of the SUT is another challenge in the TCP problem. Having available faults, the cost of executing the test cases, and the time constraints are just a few of the many factors that influence the TCP solution. Our aim was to investigate the effect of having partial factors and all factors available. Regarding testing time, an analysis of the trade-off between the improved prioritization and the potential increase in testing time will be investigated.

Dataset. While a multitude of datasets for regression testing/TCP exist (as provided above, SIR, Defect4J, JTeC, GitHub), each is specifically tailored to study a unique aspect, which results in a lack of standardization between known and desired outcomes. Furthermore, these datasets should have stronger industry support, in the sense that academic research should align more closely with real-world industry scenarios.

In conclusion, regression testing is an important step during the software development life cycle, and TCP is one of the strategies that leads to finding faults sooner with a minimum execution cost. However, currently, there is no formal definition of TCP that adequately covers the various influencing factors. Since there is currently no TCP formalization that generalizes the various influencing factors, our vision is to fill this gap with a unified TCP framework that adequately integrates numerous perspectives: requirements, context, and information.

ACKNOWLEDGMENT

This work was funded by the Ministry of Research, Innovation, and Digitization, CNCS/CCCDI - UEFISCDI, project number PN-III-P1-1.1-TE2021-0892 within PNCDI III.

REFERENCES

Almaghairbe, R. and Roper, M. (2017). Separating passing and failing test executions by clustering anomalies. *Software Quality Journal*, 25(3):803–840.

Anderson, D. J. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.

Apke, L. (2015). *Understanding the Agile Manifesto*. Lulu.com, United States.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional.

Bertolino, A., Guerriero, A., Miranda, B., Pietrantuono, R., and Russo, S. (2020). Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 1–12, New York, NY, USA. Association for Computing Machinery.

Burnstein, I. (2010). *Practical Software Testing*. Springr, Sprineger Professional Computing.

Dang, V. and Zarozinski, M. (2020). Ranklib.

Engström, E., Runeson, P., and Skoglund, M. (2010). A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30.

GitHub (n.d.). Github. https://github.com/. Accessed: 2023-05-23.

Graves, T. L., Harrold, M. J., Kim, J., Porters, A., and Rothermel, G. (1998). An empirical study of regression test selection techniques. In *Proceedings of the*

20th International Conference on Software Engineering, pages 188–197.

Găceanu, R. D., Szederjesi-Dragomir, A., Pop, H. F., and Sârbu, C. (2022). Abarc: An agent-based rough sets clustering algorithm. *Intelligent Systems with Applications*, 16:200117.

JTeC (n.d.). Jtec: A large collection of java test classes for test code analysis and processing. https://github.com/MSR19-JTeC/JTeC. Accessed: 2023-05-23.

Juristo, N. and Moreno, A. (2004). Reviewing 25 years of testing technique experiments. *Empirical Software Engineering*, 9(7-44).

Just, R., Jalali, D., and Ernst, M. D. (2014). Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, page 437–440, New York, NY, USA. Association for Computing Machinery.

Kandil, P., Moussa, S., and Badr, N. (2017). Cluster-based test cases prioritization and selection technique for agile regression testing. *Journal of Software: Evolution and Process*, 29(6):e1794. e1794 JSME-15-0111.R1.

Khalid, Z. and Qamar, U. (2019). Weight and cluster based test case prioritization technique. *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1013–1022.

Medhat, N., Moussa, S. M., Badr, N. L., and Tolba, M. F. (2020). A framework for continuous regression and integration testing in iot systems based on deep learning and search-based techniques. *IEEE Access*, 8:215716–215726.

Pan, R., Ghaleb, T. A., and Briand, L. (2022). Atm: Black-box test case minimization based on test code similarity and evolutionary search. *arXiv preprint arXiv:2210.16269*.

Pan R., Bagherzadeh M., G. T. e. a. (2022). Test case selection and prioritization using machine learning: a systematic literature review. *Empir Software Eng*, 29:1 – 43.

Paterson, D., Campos, J., Abreu, R., Kapfhammer, G. M., Fraser, G., and McMinn, P. (2019). An empirical study on the use of defect prediction for test case prioritization. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 346–357.

Poppendieck, M. and Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional.

Pradeepa, R. and VimalDevi, K. (2013). Effectiveness of test case prioritization using apfd metric: Survey. In *International Conference on Research Trends in Computer Technologies (ICRTCT—2013). Proceedings published in International Journal of Computer Applications®(IJCA)*, pages 0975–8887.

Qu, X., Cohen, M., and Woolf, K. (2007). Combinatorial interaction regression testing: A study of test case generation and prioritization. In *2007 IEEE International Conference on Software Maintenance*, Los Alamitos, CA, USA. IEEE Computer Society.

Rothermel, G., Untch, R. H., Chu, C., and Harrold, M. J. (1999). 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)*, pages 179–188. IEEE.

Salehie, M., Li, S., Tahvildari, L., Dara, R., Li, S., and Moore, M. (2011). Prioritizing requirements-based regression test cases: A goal-driven practice. In *2011 15th European Conference on Software Maintenance and Reengineering*, pages 329–332.

Schwaber, K. and Sutherland, J. (2017). *The Scrum Guide*.

Singh, A., Singhrova, A., Bhatia, R., and Rattan, D. (2023). *A Systematic Literature Review on Test Case Prioritization Techniques*, chapter 7, pages 101–159. John Wiley & Sons, Ltd.

SIR (n.d.). Software-artifact infrastructure repository. http://sir.unl.edu/. Accessed: 2023-05-23.

Spieker, H., Gotlieb, A., Marijan, D., and Mossige, M. (2017). 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*, ISSTA 2017, page 12–22, New York, NY, USA. Association for Computing Machinery.

Srikanth, H., Hettiarachchi, C., and Do, H. (2016). Requirements based test prioritization using risk factors: An industrial study. *Information and Software Technology*, 69:71 – 83.

Tiutin, C.-M. and Vescan, A. (2022). Test case prioritization based on neural networks classification. In *Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis*, AISTA 2022, page 9–16, New York, NY, USA. Association for Computing Machinery.

Vescan, A., Chisalita-Cretu, C., Serban, C., and Diosan, L. (2021). On the use of evolutionary algorithms for test case prioritization in regression testing considering requirements dependencies. In *Proceedings of the 1st ACM International Workshop on AI and Software Testing/Analysis*, AISTA 2021, page 1–8, New York, NY, USA. Association for Computing Machinery.

Vescan, A., Găceanu, R., and Szederjesi-Dragomir, A. (2023b). Neural network-based test case prioritization in continuous integration. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pages 68–77. IEEE.

Vescan, A., Gaceanu, R., and Szederjesi-Dragomir, A. (accessed November 2023a). Embracing unification:a comprehensive approach to modern tcp.

Yoo, S. and Harman, M. (2010). Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *J. Syst. Softw.*, 83(4):689–701.