FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
DEGREE PROGRAMME IN ELECTRONICS AND COMMUNICATIONS ENGINEERING

Master's Thesis

Reinforcement Learning for Optimizing Regression Testing

Author	Zahra Moolawi
Supervisor	Olli Silvén
Second Examiner	Tuomo Hänninen
Technical supervisor	Teemu Kanstrén

January 2024

Moolawi Z. (2024) Reinforcement Learning for Optimizing Regression Testing. Faculty of Information Technology and Electrical Engineering, Degree Program in Electronics and Communications Engineering, 58 pages.

ABSTRACT

The agile manifesto emerged in 2001, introducing principles that emphasize flexibility, customer satisfaction, and continuous improvement. Continuous Integration (CI) supports these Agile objectives by ensuring that code changes are automatically tested and merged frequently, leading to early detection of issues and allowing teams to adapt quickly. Combining CI with regression testing is a widely recognized as an effective approach in software development, ensuring that new changes do not adversely affect existing functionality. However, regression testing can be challenging due to the time and resources required to test the entire system at every iteration. To address these challenges, one alternative is Test Case Prioritization (TCP). This involves strategically ordering test cases to detect regression faults earlier.

The primary objective of this thesis is to streamline the CI build cycle by improving the fault detection rate. This enhancement will enable software developers to integrate their changes more rapidly and continuously. This thesis adopts the systematic methodology of the Design Science Research (DSR) and was conducted in three phases. First, we reviewed relevant literature to identify opportunities for further research and examine the existing knowledge base. The second phase involved developing a history-based TCP method using Reinforcement Learning (RL), which was subjected to multiple iterations of improvement and evaluation. In the final stage, the results from the initial two stages were synthesized to draw conclusions and identify practical implications.

In this thesis, TCP is modeled as a ranking problem addressed by the RL agent.This agent iteratively adjusts its actions based on feedback (rewards) from its environment, learning to rank test cases adaptively in an optimal order. For RL model development, we utilized the stable-baselines3 framework, specifically employing algorithms like Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), and Deep Q-Networks (DQN). The performance of each algorithm was assessed using the Average Percentage of Fault Detected (APFD) metric. The method was trained on 963 CI builds, encompassing 44,000 test cases extracted from the Nokia CI repository.

The primary contribution of this thesis is the is the creation and assessment of an RL-based TCP method that is accurate in prioritizing test cases and adaptable to changes in CI environment. The experimental results underscore the efficiency of the A2C algorithm, which achieved a mean APFD score of 0.78 across 963 CI cycles, with average testing and training times per cycle of 322 ms and 54129 ms, respectively. Based on these findings, it's evident that RL is a promising approach in software testing, particularly effective in managing the complexities of dynamic testing environment such as the CI environment.

Keywords: agile software development, continuous integration, test case prioritization

Moolawi Z. (2024) Vahvistusoppiminen regressiotestauksen optimoimiseksi. Oulun yliopisto, tieto- ja sähkötekniikan tiedekunta, elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Diplomityö, 58 sivua.

TIIVISTELMÄ

Ketterä manifesti ilmestyi vuonna 2001, tuoden esiin periaatteita, jotka korostavat joustavuutta, asiakastyytyväisyyttä ja jatkuvaa kehittämistä. Jatkuva Integrointi (CI) tukee näitä ketterän ohjelmistokehityksen tavoitteita varmistamalla, että koodimuutokset testataan automaattisesti ja integroidaan usein, mikä johtaa ongelmien varhaiseen havaitsemiseen ja antaa tiimeille mahdollisuuden sopeutua nopeasti. CI yhdistettynä regressiotestauksen kanssa on laajalti tunnustettu tehokkaaksi lähestymistavaksi ohjelmistokehityksessä, varmistaen, että uudet muutokset eivät vaikuta haitallisesti olemassa olevaan toiminnallisuuteen. Regressiotestaus voi kuitenkin olla haasteellista ajan ja resurssien kannalta, jotka vaaditaan koko järjestelmän testaamiseen jokaisessa iteraatiossa. Näihin regressiotestauksen haasteiden ratkaisemikseksi yksi vaihtoehto on testitapausten priorisointi (TCP), joka järjestää strategisesti testitapauksia vikojen varhaisen havaitsemiseksi.

Tämän diplomityön päätavoitteena on suoraviivaistaa CI sykliä parantamalla vikojen havaitsemisnopeutta, jonka seurauksena ohjelmistokehittäjät voivat integroida muutoksensa nopeammin. Tässä työssä on noudatettu Design Science Research (DSR) systemaattista menetelmää, jota toteutettiin kolmessa vaiheessa. Ensimmäisessä vaiheessa tarkastelimme aiheeseen liittyvää kirjallisuutta tunnistaaksemme lisätutkimuksen mahdollisuuksia ja tarkastellaksemme olemassa olevaa tietopohjaa. Toisessa vaiheessa kehitettiin TCP menetelmää, joka pohjautuu testitapausten aiempiin suoritustietoihin ja hyödyntää vahvistusoppimista. Kehitysvaiheessa menetelmä testattiin useiden iterointien aikana parannusten ja arvioinnin kautta. Viimeisessä vaiheessa yhdistettiin ensimmäisistä kahdesta vaiheesta saadut tulokset päätelmien tekemiseksi ja käytännön sovellusten tunnistamiseksi.

Tässä työssä TCP on mallinnettu järjestysongelmana, jota käsitellään vahvistusoppimisen (RL) agentilla. Tämä agentti säätää toimintaansa iteratiivisesti palautteen perusteella ympäristöstään, oppien järjestämään testitapaukset adaptiivisesti optimaalisessa järjestyksessä. RL Malleja on kehitetty käyttäen stable-baselines3, josta sovelsimme algoritmeja kuten Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO) ja Deep Q-Network (DQN). Kunkin algoritmin suorituskykyä arvioitiin keskimääräisen vikojen havaitsemisprosentin (APFD) avulla. Menetelmää harjoiteltiin 963 CI sykliä käyttäen, joka kattaa 44,000 testitapausta, jotka on otettu Nokian CI-repositoriosta.

Tämän diplomityön pääkontribuutio on RL-pohjaisen TCP-menetelmän luominen ja arviointi, joka on tarkka testitapausten priorisoinnissa ja sopeutuva CI-ympäristön muutoksiin. Tulokset korostavat A2C-algoritmin tehokkuutta, joka saavutti keskimääräisen APFD-pistemäärän 0,78 yli 963 CI-sykliä, keskimääräisten testaus- ja harjoitteluaikojen ollessa sykliä kohti vastaavasti 322 ms ja 54129 ms. Näiden löydösten perusteella on ilmeistä, että RL on lupaava lähestymistapa ohjelmistotestauksessa, erityisen tehokas dynaamisten testiympäristöjen, kuten jatkuvan integraation, monimutkaisuuksien hallinnassa.

Avainsanat: ketterä ohjelmistokehitys, jatkuva integrointi, testitapausten priorisointi

CONTENTS

PREFACE

I would like to express my gratitude to Tarmo Kumpula, my line manager at Nokia, for giving me this opportunity. The support I received, which allowed me to work on this project, enabled me to explore a topic that is both of personal interest and of direct relevance to the company.

I would also like to thank Teemu Kanstrén, my technical supervisor. His continuous support, constructive feedback, and expert insights on the thesis topics have been invaluable. Special thanks to Heikki Angria for his guidance and practical advice throughout the project; his insights have been incredibly helpful.

I would like to express my appreciation for Tuomo Hänninen as a reviewer. His insightful feedback consistently enhances the quality of this thesis.

Lastly, I am grateful to my academic supervisor at Oulu University, Olli Silvén. His scientific guidance, commitment to academic rigor, and thorough evaluation have greatly enriched this thesis.

Zahra Moolawi

Oulu, January 24, 2024

LIST OF SYMBOLS AND APPREVIATIONS

A2C	Advantage Actor-Critic
ASD	Agile Software Development
APFD	Average Percentage of Fault Detected
CD	Continuous Delivery
CDE	Continuous Deployment
CI	Continuous Integration
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
DSR	Design Science Research
FEP	Fault Exposure Potential
IS	Information system
IT	Information Technology
LTR	Learning-to-Rank
ML	Machine Learning
MART	Multiple Additive Regression Tree
PPO	Proximal Policy Optimization
RTL	Ranking-to-learn
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SARSA	State-Action-Reward-State-Action
TCM	Test Case Minimization
TCS	Test Case Selection
TCP	Test Case Prioritization
TD3	Twin Delayed Deep Deterministic Policy Gradients

1 Introduction

The software development landscape is characterized by rapid change and dynamism, with organizations in the field constantly exploring innovative methods to deliver higher-quality products quickly and cost-effectively [1]. In response to this demand, the Agile Software Development (ASD) manifesto emerged in 2001 as a direct response to the growing need for faster and more efficient software development approaches [2]. Over the years, leading organizations have successfully embraced ASD, enabling them to release software more frequently while maintaining a high level of confidence in its quality. This highlights the remarkable effectiveness of the ASD methodology in addressing the dynamic demands of modern software development [3].

Continuous Integration (CI) is an essential component in achieving Agile Software Development (ASD) objectives by facilitating frequent code integration, rapid feedback loops, and collaboration among team members [4]. The core aim of CI is to automate and regularly integrate code changes from multiple developers into a shared repository, ensuring that the software remains stable and functional throughout its development lifecycle. CI encompasses automated and frequent code integration, accompanied by automated testing, including regression testing [5]. Regression testing is executed after each development iteration, utilizing the complete test suite to verify that new code changes have not inadvertently introduced defects or regressions into previously functioning features [6].

As software projects grow in both scale and complexity, the conventional ”retest-all” approach, where the entire test suite is executed, becomes increasingly impractical for developers [7]. The increasing complexity of these projects demands a significant investment in time and human effort, which can strain the resources allocated to maintaining software quality throughout its development [8]. Running a comprehensive test suite can demand several days or even weeks, imposing significant demands on both resource allocation and project timelines [9].

Research has proposed various solutions to tackle the challenges associated with regression testing in order to optimize CI/CD processes and enhance overall testing efficiency [6]. One alternative approach for optimizing regression testing is Test Case Prioritization (TCP). This technique involves strategically ordering test cases to ensure that the most critical ones are executed first. This approach enables the early detection of potential defects and ensures efficient allocation of testing resources [10].

In theory, TCP holds great promise as a solution for optimizing regression testing, with a plethora of techniques proposed to enhance its effectiveness. However, when implemented in CI environment, many of these proposed TCP techniques encounter significant challenges [7]. This is because in CI, regression testing occurs more frequently and at shorter intervals, making approaches that require extended periods for exhaustive analysis impractical and inefficient due to the rapid pace of changes [11]. To address these challenges and ensure that the TCP technique can be applied in the CI environment, it must meet two essential criteria:

1. **Speed:** In the context of CI, TCP techniques must possess the capability to quickly prioritize test cases, recognizing the time-sensitive nature of CI. Quick decision-making becomes essential to keep pace with the continuous flow of code changes, ultimately surpassing the traditional ”retest-all” approach in terms of time.

2. **Adaptability:** TCP techniques should demonstrate the ability to seamlessly adjust to the frequent changes that characterize the CI environment. They must exhibit the flexibility to update their prioritization strategy dynamically, ensuring sustained effectiveness even as changes occur on a regular basis.

to bridge the existing research gap, in this thesis an RL-based TCP method for CI environment has been developed. This method excels in both time efficiency and adaptability, making it well-suited for the ever-changing dynamics and time constraints inherent to CI environment.

To map TCP into a RL framework, we conceptualize TCP as a ranking problem. In this setup, the RL agent is tasked with ordering test cases through pairwise ranking comparison. To achieve this, we have carefully designed the fundamental RL components such as environment, observation, action, and reward to align with the specifics of the TCP problem, particularly emphasizing its nature as a ranking problem. This requires a detailed and careful consideration of the interactions between the agent and its environment, ensuring that each component aligns with the unique requirements of TCP in the context of RL.

The developed method is designed to dynamically adapt its prioritization strategy through continuous agent-environment interactions. Within this dynamic framework, the agent makes decisions (actions) within its environment, receives rewards based on the quality of those decisions, and continuously observes the evolving environment. This ongoing interaction enables the agent to adapt and adjust its learning strategy in response to the changes occurring in its environment.

For implementing the method, we utilized Stable Baselines3, a popular Python library for reinforcement learning that provides a collection of pre-implemented and tested RL algorithms, simplifying the process for researchers and developers to experiment with and apply RL techniques in projects. Specifically, we employed the A2C, PPO, and DQN algorithms within this framework to apply RL to the TCP problem.

This thesis is conducted in the following three phases:

1. **Exploration**: During the first research phase, our primary objective was to explore the existing body of related literature within the subject area. This phase involved a systematic and comprehensive survey of relevant literature, allowing us to analyze the current state of knowledge while simultaneously identifying promising research paths for exploration.

2. **Design and Evaluation**: The second phase of this study had a distinct focus on addressing the research problem through the development and evaluation of a specific method. Within this phase, we dedicated our efforts to designing, implementing, and evaluating the technique created to achieve the research objectives. During this phase, the developed method underwent several iterations of enhancements and evaluations to ensure its effectiveness.

3. **Research Synthesis**: Finally, we integrated the findings from the previous two phases to draw conclusions and highlight potential implications.

The thesis structure is designed to provide a comprehensive understanding of the conducted research. It follows a systematic progression through distinct yet interrelated phases, all contributing to the cumulative development of the research findings. Figure 1.1 visually outlines the thesis structure, starting with the initial exploration of the study background and concluding in the final conclusions. This visual roadmap guides readers through the research methodology and the logical flow of the study, emphasizing the deliberate and systematic approach used to ensure that each research phase informs and enriches the subsequent stages.

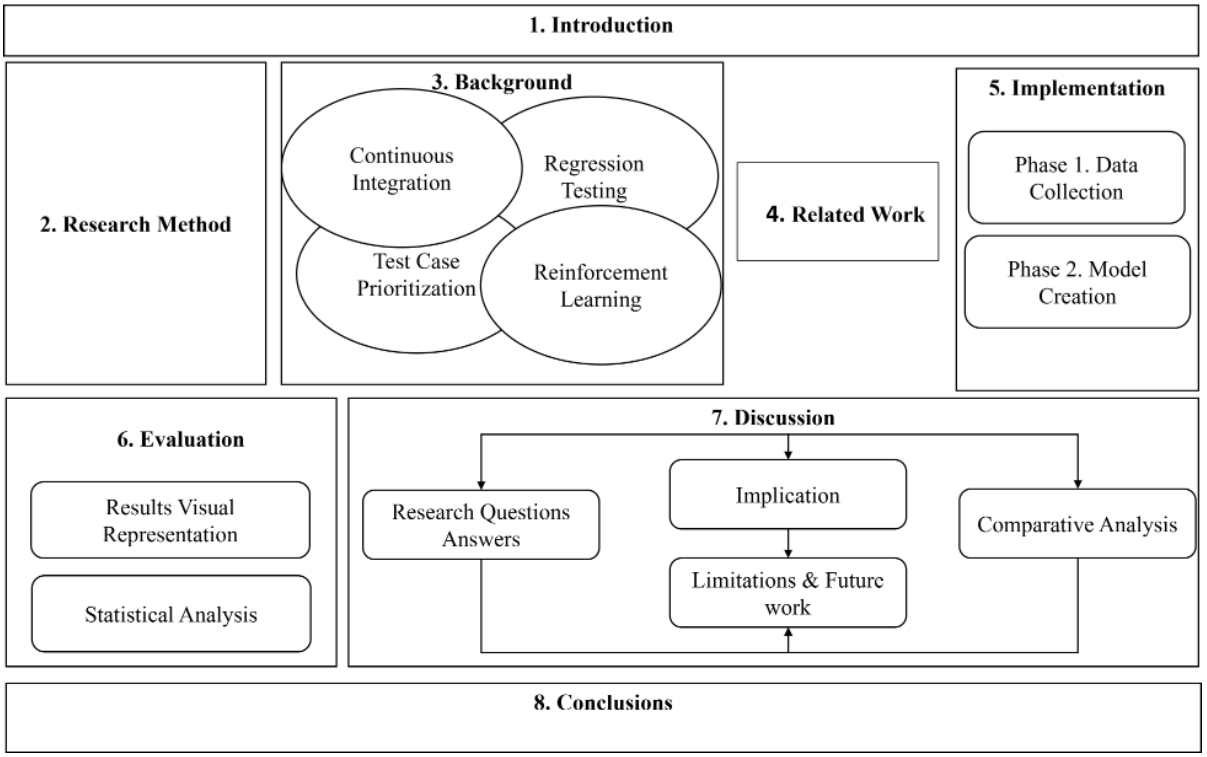

Figure 1.1. Thesis structure.

2 Research Method

This thesis adopts the Design Science Research (DSR) process, a framework that emphasizes the systematic creation and evaluation of Information Technology (IT) artifacts to address specific Information System (IS) problems within an organization. In this chapter, we thoroughly explain the application of DSR in IS, highlighting its crucial role in identifying and resolving the targeted problem. The chapter is divided into five sections.

The first section offers a comprehensive overview of DSR within the IS field, encompassing an examination of its theoretical foundations and practical applications. Subsequently, we systematically explain the DSR process, providing clear guidelines and illustrative examples for each stage.

The third section outlines the specific research methods employed, detailing the research design, analysis techniques, and how DSR principles were applied to ensure rigorous outcomes. Next, we will explore how the DSR introduced in the fourth section is applied in the context of this thesis. Finally, we present the distinctive formulation of research questions in DSR and discuss their relevance to our study.

2.1 Design Science Research in Information System

Generally, information systems are used to improve the efficiency and effectiveness of organizations [12]. The success of information system in any specific context is influenced by a combination of factors including the capabilities of the information system, organizational characteristics, human elements, development methodologies, and the surrounding environment [13]. Research in information system is dedicated to generating and disseminating knowledge in two key areas: firstly, to facilitate the development of effective IT applications for organizational management, and secondly, to enhance understanding and management of information technologies, with a particular focus on their application in managerial and organizational contexts [14].

Research in information system explore the complexity of both technological and social systems. Its primary objective is to investigate the intersection of these two areas and explore the unique phenomena that arise from this interaction [15]. While natural science research methods are effective for understanding existing and emerging phenomena, they often fall short in addressing 'wicked problems', issues that are distinct, complex, and challenging to solve [16]. Addressing these types of problems requires the formulation and assessment of creative and innovative solutions tailored to specific contexts [17]

In the field of information system, research mainly operates within two main frameworks: behavioral science and design science [18]. Behavioral science, drawing on natural science research methods, is dedicated to formulating and validating theories that explain organizational and social dynamics related to the analysis, design, development, and usage of information systems [19]. Its primary aim is to explore and understand how information systems interact with organizations and individuals, ultimately aiming to enhance effectiveness and efficiency.

Conversely, design science research, grounded in engineering and the science of creating artifacts, adopts a practical approach [20]. This paradigm focuses on the development of new and innovative artifacts that address real-world challenges. In essence, design science involves the design, develop and assessment of IT artifacts to address particular organizational challenges [14].

Although behavioral science and design science research paradigms differ fundamentally in their origins and approaches, they are integrally linked in the field of information system research.

There exists a symbiotic relationship between the two: behavioral science is concerned with developing and substantiating theories that explain organizational and social phenomena, whereas design science research is focused on accomplishing innovative concepts within information system [17]. Essentially, while behavioral science seeks to establish theoretical concepts and provide evidence for these theories, design science is dedicated to enhancing the practical application and broader utility of these concepts.

2.2 Design Science Research Explained

The design process in information system can be understood both as an action and an outcome. This dual perspective means that the process encompasses a series of iterative activities for enhancing the unique artifact to solve the IS problem [17]. From this viewpoint, DSR involves two main stages: the creation and evaluation of the design. Furthermore, DSR generates four types of artifacts: constructs, which are the vocabulary and symbols used; models, serving as abstractions and representations; methods, which include algorithms and practices; and instantiations, being the actual implemented systems or prototypes [18]. These artifacts can lead to social innovations or develop new characteristics in technical, social, or informational domains.

Hevner et al. [14] integrate aspects of behavioral science with design science to guide information systems research. This framework is designed to provide a structured approach for researchers in the field of information systems to develop and evaluate IT artifacts while also considering behavioral and social aspects. It introduce a framework that integrates the environment, IS research, and the knowledge base through a series of interrelated cycles. Figure 2.1 illustrates the connection between these cycles.

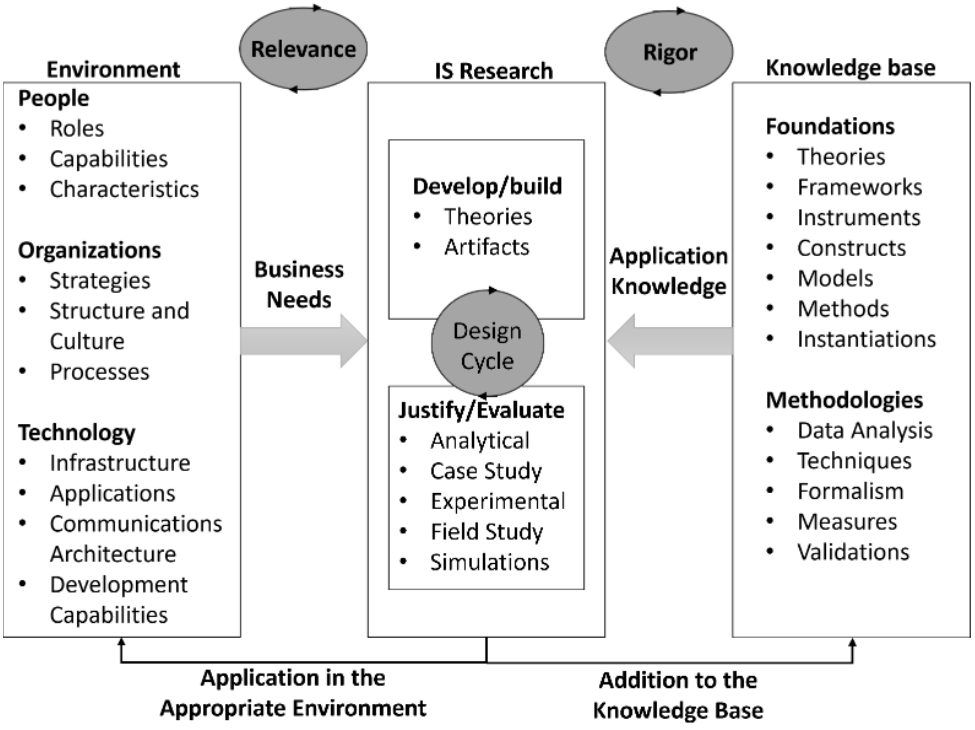

Figure 2.1. Design science research framework in IS (adapted from Hevner et al. [14]).

The environment refers to the context within which information systems exist, including the technologies, people, and organizations that interact with the systems. It sets the stage for the practical problems that design science seeks to address. Design science research within IS, involves the creation and evaluation of IT artifacts with the objective of solving identified organizational problems. These artifacts can be constructs, models, methods, or instantiations that must be evaluated for their utility in addressing these problems.

The knowledge base is an accumulation of theories, methods, and findings from prior research that informs the design science research process. It provides the theoretical foundations and methodologies that guide the creation and evaluation of IT artifacts. The connection between these elements can be explained through three cycles:

1. **Relevance Cycle** connects the environment to the design science activities, ensuring that the research is informed by and contributes to real-world problems.

2. **Rigor Cycle** connects the knowledge base with the design science activities, ensuring that the research is based on a solid theoretical and methodological foundation.

3. **Design Cycle** is where the iterative development of the artifacts takes place, moving between the construction and evaluation of the artifacts and drawing from both the relevance and rigor cycles.

The relationship among these cycles is dynamic. The relevance cycle ensures that research remains connected to practical problems in the environment, while the rigor cycle maintains the research's connection to scientific knowledge. The design cycle is the core of the process where the actual creation and refinement of artifacts occur, guided by inputs from both the relevance and rigor cycles.

This model underscores the importance of maintaining a balance between relevance (practical impact) and rigor (scientific foundation), and it highlights the contribution of design science research to both the environment through the creation of useful artifacts and to the knowledge base through the development of new theories, methods, and insights.

2.3 Desing Science Reseach Methodology

The creation of the Design Science Research Methodology (DSRM) by Peffers et al. [21] was driven by the absence of widely recognized process models for conducting DSR framework. This methodology outlines key principles, practices, and procedures that assist researchers in effectively carrying out and presenting their design science research projects. DSRM was originally developed with three fundamental objectives:

1. Offering a structured framework for design science research.

2. Ensuring consistent alignment with established literature and foundational design science principles.

3. Providing a well-organized conceptual framework for presenting design science research findings.

The DSRM proposed by Peffers et al. [21] is a structured, systematic approach that guides the creation and evaluation of IT artifacts to solve practical problems. This method can start from four different entry points and comprises six sequential steps, as depicted in Figure 2.2.

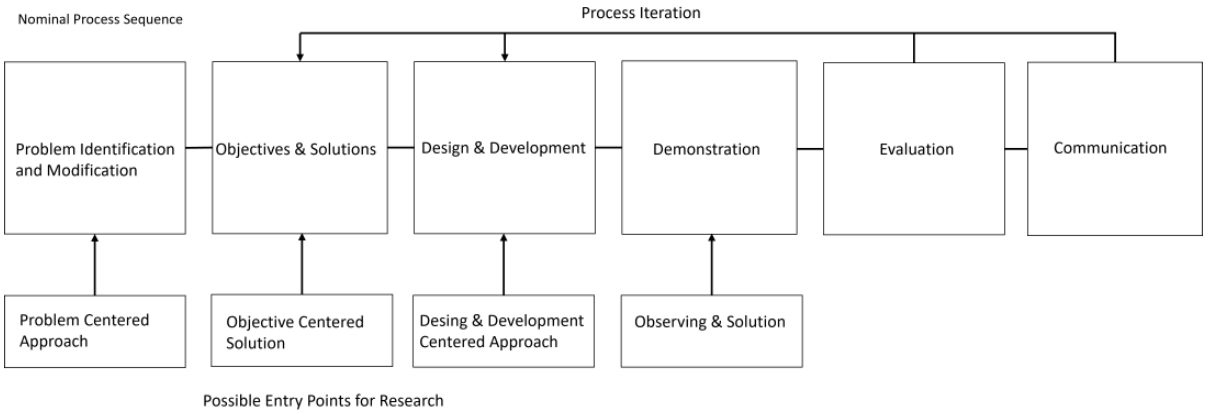

Figure 2.2. Design science research methodology introduced by Peffers et al. [21].

Each entry point, including problem-centered, objective-centered, design and development-centered, and observing and solution, offers flexibility and adaptability in addressing a wide range of research problems and contexts. The inclusion of multiple entry points allows researchers to initiate their design science research based on the most relevant aspect of their study. Each entry point offers a different starting position for the research, and the sequential steps define the progression of the project regardless of where it begins. Each entry point from which research can begin is discussed in detail below.

1. **Problem-Centered Initiation:** This entry point places its primary emphasis on the identification of tangible, real-world problems that demand effective solutions. This approach is essential in guaranteeing that the research maintains direct practical relevance and directly addresses authentic needs within a specific domain. Starting from the problem-centered perspective, the research seamlessly aligns itself with practical, real-world challenges, ensuring its significance for both practitioners and academics who seek applied solutions.

2. **Objective-Centered Solution:** In this entry point, the research starts with a well-defined objective, frequently tied to enhancing a process, improving a system's capabilities, or introducing innovative solutions. Subsequently, the focus shifts towards developing the necessary solution or artifact to achieve these established objectives. It's worth noting that this approach may occasionally overlap with a technology-driven perspective, particularly when the objective involves harnessing or innovating upon existing technology.

3. **Design and Development Centered:** This entry point initiates research with the primary focus on the development of an artifact. In this approach, the artifact itself takes center stage, and considerations regarding its potential applications or the problems it can address are typically made during or after the development process.

4. **Objective & Solution:** This entry point is frequently adopted in situations characterized by substantial technological innovations or novel concepts that researchers aim to explore across diverse practical applications. It diverges from the traditional problem-first approach in that it doesn't start with a specific problem but rather initiates with a solution that seeks relevant problems or objectives to address.

The initial entry point sets the course for the entire research project within the DSRM framework. For example, adopting a problem-centered approach leads to a more targeted and

specific trajectory, whereas commencing the research from the objective and solution entry point can result in a broader exploration of potential applications. These six sequential steps are intricately connected to the chosen entry points, providing a comprehensive process that guides the research from its inception to its conclusion, as follows:

1. **Problem Identification and Motivation:** This step is essential as it involves the identification of the problem to be addressed, establishes the relevance and motivation for the research, and lays the foundational groundwork for all subsequent activities in the research process. It serves as a cornerstone for the entire research project, as every subsequent step in the DSRM builds upon the problem identified in this initial phase.

    - **Key Considerations:** What specific problem or need does this research intend to address, and why is it important to find a solution?

2. **Objective of Solution:** Once the problem is identified, it is imperative for researchers to precisely define their objectives concerning the proposed solution. These objectives should exhibit clarity and measurability, thereby providing a tangible target for what the artifact or solution is expected to accomplish. This clarity serves as a guiding beacon throughout the subsequent stages of design and development, and it is instrumental in the evaluation of the artifact's effectiveness in addressing the identified problem.

    - **Key Considerations:** What are the specific objectives that the proposed solution is intended to achieve in addressing the identified problem?

3. **Design and Development:** With clear objectives established, the design creation process start. In this critical step, the theoretical and conceptual groundwork, carefully laid in the previous phases, undergoes a transformation into a tangible, functional artifact. This phase is of paramount importance as it effectively translates the proposed solution into reality, providing a practical means to address the identified problem and forming the essential foundation for its subsequent evaluation and real-world application.

    - **Key Considerations:** How can the solution be designed and developed to meet the specified objectives and effectively address the identified problem?

    Demonstration: The demonstration phase serves as a compelling proof of concept, offering tangible evidence that the solution not only operates theoretically but also functions effectively in real-world practice. It plays a crucial role in validating the functionality of the artifact and its capacity to effectively address the identified problem. This phase establishes the groundwork for the subsequent evaluation phase by showcasing the practical applicability and potential of the solution.

    - **Key Considerations:** How does the artifact perform in a practical setting?

4. **Evaluation:** The evaluation step is dedicated to the validation of whether the artifact aligns with the objectives defined earlier in the research process. It encompasses a comprehensive assessment of the artifact's effectiveness, efficiency, and usability within its intended context, with a keen focus on how well it addresses the identified problem. This essential phase entails a rigorous examination of the artifact's performance, ensuring it not only meets but also successfully fulfills its designated goals and effectively resolves the targeted issue.

    - **Key Considerations:** How effectively does the artifact achieve its intended objectives?

5. **Communication:** This step involves clearly discussing the contribution of the research to the field. It's important to communicate how the research advances knowledge, solves a specific problem, or introduces a novel solution. It is essential to share the research findings, articulating the contributions to knowledge, facilitating knowledge transfer, encouraging collaboration, and promoting the adoption and impact of the research in both academic and practical domains.

    - **Key Considerations:** How can the research findings, including the development and effectiveness of the artifact, be effectively communicated to both the academic community and industry practitioners to maximize understanding, impact, and application?

2.4 Methodological Approach for this Thesis

This thesis adopts the Design Science Research Methodology (DSRM) proposed by Peffers et al. [21], which is particularly well-suited for the creation and evaluation of IT artifacts. It encompasses a thorough state-of-the-art review and aligns seamlessly with the six activities described in the DSRM framework. The primary emphasis is on the detailed design, development, and evaluation of an artifact. This artifact is designed to not only deliver tangible benefits but also to be easily assessable. This adherence to the six distinct activities, as illustrated in Figure 2.3, provides a structured and comprehensive approach to guide the research process.

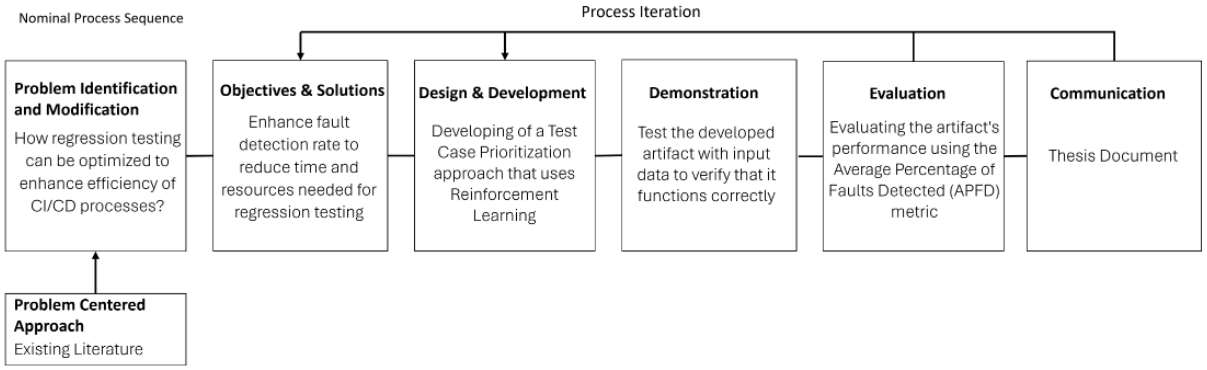

Figure 2.3. Design science research methodology employed in this thesis (Adapted from Peffers et al. [21]).

1. **Problem Identification and Motivation**: This thesis aims to address the challenges associated with regression testing within CI environments. Integral to CI, regression testing verifies that new code changes do not impact existing functionalities. However, the frequency of software updates and increasing project complexity make full-scale regression testing impractical, especially given CI's resource limitations. Research shows that non-adaptive methods using complex TCP algorithms are ineffective in CI. These methods are often limited by time constraints and an inability to adapt to CI's dynamic nature, fail to efficiently prioritize test cases, struggling to meet the evolving requirements of CI environment. As a result, there is growing interest in enhancing regression testing within CI environment through adaptive, time-efficient methods. This is motivated by the need to balance rapid development cycles with the assurance of software quality and

reliability, improving both efficiency and adaptability.

2. **Objective of the Solution**: The ultimate objective of this study is to enhance the efficiency of regression testing by increasing the fault detection rate, thereby reducing the time and resources required to execute the entire test suite and enabling faster development cycles without compromising software quality. The methodology developed for optimizing regression testing should be capable of quickly adapting to the changes inherent in the CI environment, ensuring that regression tests remain relevant and effective throughout the project's evolution. Additionally, this approach focuses on optimizing resource utilization, which leads to more cost-effective development and supports rapid, agile development cycles.

3. **Design and Development:** In the design and development phase, we strictly adhered to industry best practices and specific design principles. These were significantly informed by prior research, especially studies concentrating on the creation of practical solutions and empirical research. Such studies provided a foundation for understanding effective methodologies and outcomes in real-world applications. Methodologically, we embraced formal software engineering techniques to enhance the efficiency of the design and development phase. Instead of developing the RL algorithms from scratch, we integrated contemporary design patterns and utilized a specific RL framework to ensure a solid structural foundation. Our development tools comprised a database for extracting test case execution data, the Python programming language for developing the method, the PyCharm Integrated Development Environment (IDE) to facilitate the development process, and the Stable Baselines3 framework, which provides a set of Python implementations of RL algorithms for research and development. The detailed design and development methodologies are further elaborated in Chapter 5.

4. **Demonstration:** Considering the specific aspects of the problem, objectives, and the available resources, initially we conduct an experiment involving only data from 60 CI builds to test the developed method functionality. The aim was to assess the functionality of the developed method by closely monitoring its ability to prioritize test cases and its responsiveness to newly introduced data, along with evaluating its time efficiency. The assessment of time efficiency involved monitoring the testing and training times of the RL algorithm for each CI cycle. While the method effectively prioritized test cases and adapted well to new data, it did not initially meet our time efficiency expectations. In response, we undertook the crucial step of reducing the number of input features, resulting in a significant improvement in the speed of test case prioritization. This adjustment proved instrumental in aligning the method with our time efficiency objectives.

5. **Evaluation:** In this thesis, the evaluation of the developed artifact is conducted through the use of quantitative metrics. The assessment encompasses measuring the artifact's adaptability to new data and its time efficiency, which involves continuous monitoring of testing and training times during the program's runtime. Furthermore, the artifact's error detection capability is quantified using the Average Percentage of Faults Detected (APFD). The APFD metric offers a weighted average assessment of the faults detected, with scores ranging from 0 to 1, where higher scores signify better and more efficient fault detection. This comprehensive approach ensures a robust and data-driven evaluation of the artifact's performance. The APFD can be mathematically expressed as:

$$\text{APFD} = 1 - \frac{TF1 + TF2 + \cdots + TFm}{nm} + \frac{1}{2n} \tag{2.1}$$

- $TF1, TF2, \ldots, TFm$: The positions of the first test cases that reveal each of m faults.
- n: The total number of test cases in the test suite.
- m: The total number of faults in the test suite.

To comprehensively compare the performance of each distinct RL algorithm, we subjected each of them to a rigorous assessment against the benchmark of random prioritization. To ensure a robust comparison of the algorithms' performance against the benchmark, our methodology comprises three essential steps.

First, we conducted Shapiro-Wilk tests to determine whether the APFD data points of each distinct RL algorithm exhibited a normal distribution. Subsequently, the initial assessment guided us in selecting and conducting the appropriate statistical tests for evaluating whether there exists statistical difference between the performance of the RL algorithms and the benchmark of random prioritization. Finally, we measured the magnitude of any detected statistical difference using Vargha and Delaney's A (VDA) effect size. VDA quantifies the practical significance or magnitude of statistical differences between two group of data APFD, providing valuable insights into the real-world importance of our statistical findings. This comprehensive approach ensures a thorough and rigorous evaluation of the algorithms' performance in comparison to the benchmark.

6. **Communication:** This thesis comprehensively covers the entire research process and its diverse components. It starts with an extensive exploration of the problem, thoroughly elaborating on its significance. Subsequently, it introduces the designed artifact, highlighting its utility, innovation, and the methodology employed in its design and development. Additionally, the document delves deeply into a comprehensive evaluation of the artifact, rigorously assessing its effectiveness, identifying any potential weaknesses, and offering insights into potential future improvements.

2.5 Research Questions

In the context of DSR, the formulation of research questions presents a unique challenge due to the specific nature of the DSR paradigm. Hoang et al. [22] introduced a typology that serves as a guide for creating research questions within the DSR framework originally introduced by Hevner et al. [14]. This typology plays a crucial role in helping researchers articulate research questions within the DSR framework. The typology proposed by Hoang et al. [22] is visualized in Figure 2.4, which provides a visual representation of the different dimensions and aspects that need to be considered when formulating research questions in the context of DSR.

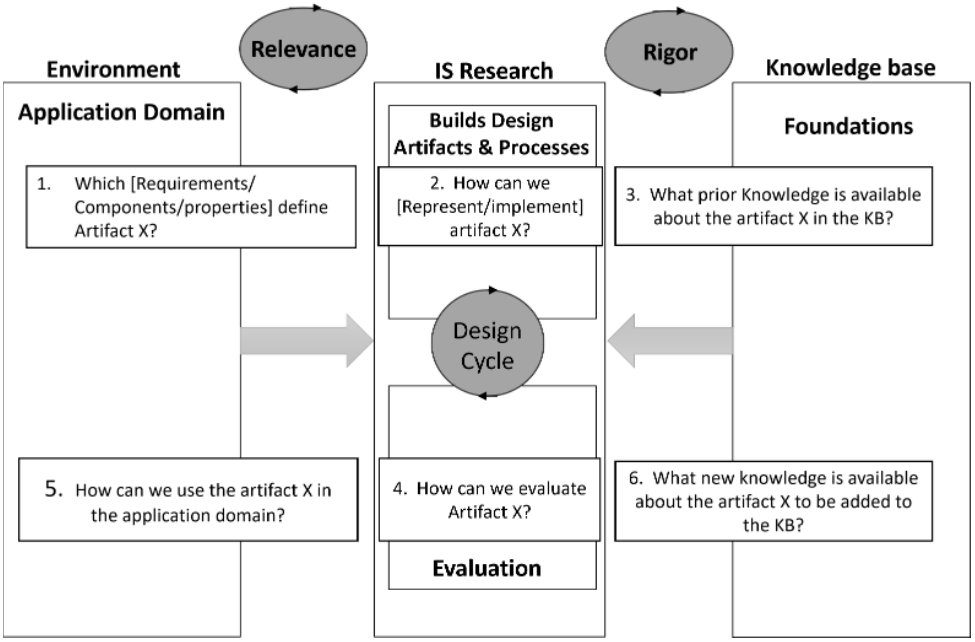

Figure 2.4. Research question formulation in DSR framework (Adapted from Hoang et al. [22]).

The following research questions, designed to guide the thesis structure and align with the principles of DSR process, are as follows:

- ***RQ1*. How can Reinforcement Learning (RL) be applied to Test Case Prioritization (TCP) in Continuous Integration (CI) environment?**
  This research question explores the feasibility of applying RL to TCP within a CI environment. It seeks to identify the technical requirements, necessary components, and properties required for implementing RL-based TCP in CI environment. Furthermore, it intends to examine the resource implications and integration challenges, focusing on the practicality and investment required for effectively implementing RL in TCP processes within CI environment.

- ***RQ2*. Which RL algorithm, A2C, PPO, or DQN, outperforms the others when compared against the baseline of random prioritization?**
  This question is designed to evaluate the developed artifact by comparing the effectiveness of different RL algorithms: A2C, PPO, and DQN, against a baseline of random prioritization. This evaluation aims to identify the most effective algorithm and delve into the trade-offs associated with each, considering aspects such as computational efficiency, accuracy, and adaptability in a CI environment.

- ***RQ3*.What limitations have been identified in using Reinforcement Learning (RL) for Test Case Prioritization (TCP) in Continuous Integration (CI) environment?**
  This question aims to offer a comprehensive understanding of the potential drawbacks, challenges, and constraints that may arise when utilizing RL algorithms in the context of TCP within a CI environment. By conducting a critical assessment of these limitations, the study introduces an element of rigor and balance, ensuring a well-rounded evaluation of the developed model. This approach is essential for identifying areas that may require improvement and for guiding future enhancements, ultimately contributing to the advancement of this field.

The structure of the thesis is built upon these three research questions, each designed to highlight a different aspect of the research. This strategic framework is intended to offer comprehensive insights, spanning from the fundamental applicability to the performance evaluation of various RL algorithms, and concluding with a critical analysis of the limitations.

3 Background

This chapter introduces the context and background of the research, organized into five sections, with each focusing on a fundamental aspect crucial to our analysis. We start by exploring CI/CD practices in the software development process, setting the stage for the study. Subsequent to this, we delve into regression testing within the CI environment, underscoring its essential role in modern software development. The next section addresses the challenges associated with regression testing in the CI environment, providing a deeper understanding of these challenges. In the fourth section, we discuss the concept of TCP, highlighting its importance and applications. The chapter concludes with a detailed explanation of how reinforcement learning works, providing fundamental information for subsequent research stages.

3.1 CI/CD processes in Software Development

In today's highly competitive software market, organizations are under constant pressure to deliver innovative products and features rapidly while ensuring the highest quality standards [23]. To meet these demands, many companies have embraced modern methodologies and practices that are centered around the principles of Continuous Integration (CI), Continuous Delivery (CD), and Continuous Deployment (CDE). These practices, collectively referred to as CI/CD, have become instrumental in helping organizations accelerate their software development and feature delivery processes without compromising on quality[4].

Continuous integration is a software development practice where team members regularly integrate their code changes into a shared repository, typically multiple times a day [5]. The primary objective of CI is to automate the process of building and testing software, ensuring that new code changes are seamlessly integrated with existing code . It aims to detect and address integration issues and bugs at an early stage in the development cycle, thereby reducing the risk of defects in production [24]. CI enables shorter and more frequent release cycles, enhances software quality, and team productivity by providing quick feedback on code changes, facilitating efficient collaboration, and automating repetitive tasks in the development process.

Continuous Delivery (CD) builds upon CI by automating the deployment pipeline. With CD, organizations can automatically and consistently package and deploy their software to various environments, such as development, testing, and staging, in a reliable and repeatable manner [25]. CD ensures that software is always in a deployable state, making it easier to push new features or bug fixes into production quickly and with confidence.

Continuous Deployment (CDE) takes the concept further by automating the release of code changes directly into production, without manual intervention [4]. While not suitable for all applications, this approach can be highly effective for organizations looking to deliver new features and updates to end-users at a rapid pace. It streamlines the deployment process and minimizes downtime while maintaining a high level of quality and reliability.

Figure 3.1 serves as a visual representation of how CI/CD practices work together within an agile framework, demonstrating their collaborative and interrelated role in creating a streamlined and highly efficient software development pipeline. This visualization offers a clear representation of how these practices interact and complement each other, contributing to a comprehensive understanding of their significance in modern software development.

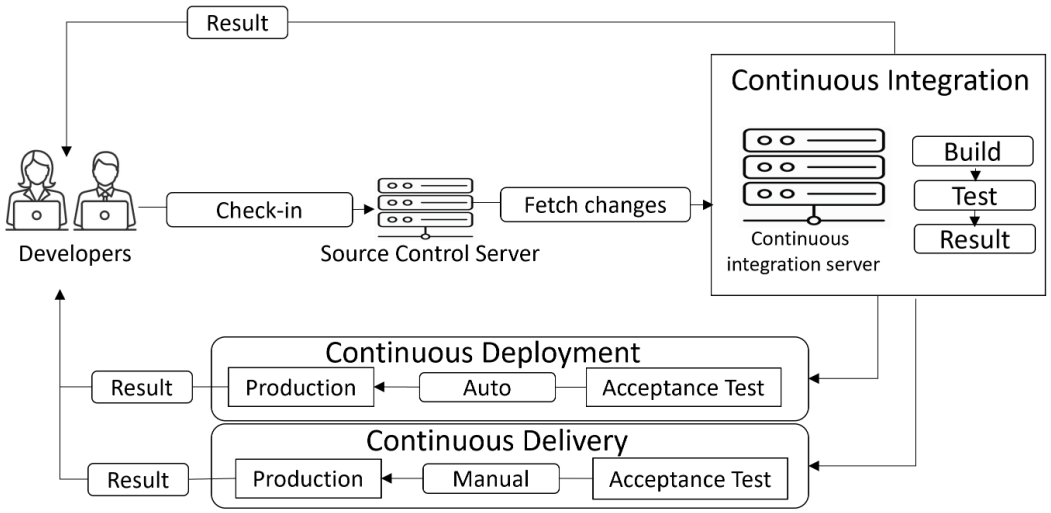

Figure 3.1. Overview of interactions between CI/CD processes (adapted from Shahin et al. [4]).

Figure 3.1 illustrates the following key points:

1. At the initial stage of the pipeline, CI is depicted as a continuous feedback loop where developers regularly commit code changes to a shared repository. After each commit automated build and testing processes are triggered immediately, ensuring that new code modifications are promptly integrated and validated. This iterative process helps identify integration issues early, maintaining code quality.

2. As the pipeline progresses, continuous delivery takes center stage. It demonstrates how the automated deployment pipeline seamlessly packages and deploys code changes across various environments, such as development, testing, and staging. This step ensures that software remains in a deployable state, ready for release at any given time.

3. In the final phase, continuous deployment is presented as the conclusion of the pipeline. It showcases how code changes that have successfully passed through CI and CD are automatically deployed into the production environment without manual intervention. This highlights the potential for rapid and reliable feature delivery directly to end-users.

The figure also contextualizes these CI/CD practices within an agile framework. It illustrates how the iterative and collaborative nature of agile methodologies aligns with CI/CD principles, emphasizing the importance of adaptability, customer feedback, and continuous improvement. This representation underscores the smooth and efficient flow of code through the entire pipeline. It emphasizes the reduction of manual interventions, leading to quicker and more reliable software releases.

CI/CD practices have become essential tools for organizations aiming to meet the demands of the competitive software market. By embracing these methodologies and practices, businesses can achieve faster product delivery, maintain high-quality standards, and stay agile in a constantly evolving landscape.

3.2 Regression Testing challenges

In CI, regression testing emerges as a fundamental and essential component of the software development process. Whenever developers commit changes to the shared code repository, a

sequence of automated tasks, including regression testing, is promptly triggered. This testing practice plays a crucial role in maintaining the integrity of a software application or system [6].

Regression testing is a software testing methodology characterized by its purposeful repetition of tests on an application or system. Its principal objective is to ensure that any recent code alterations, updates, or enhancements have not inadvertently compromised the functionality of the existing software [26]. Essentially, regression testing ensures that software modifications do not compromise the reliability and functionality of an application. It ensures against the unintended consequences of code changes and contributes to the overall quality and stability of the software, aligning with the core principles of continuous integration.

However, in the ever-evolving landscape of software development, the impact of growing complexity and accelerated development cycles has posed challenges for traditional regression testing methods [26]. While regression testing remains essential, its traditional implementation often falls short in terms of efficiency, coverage, and speed. With the continuous growth of software systems, the regression test suite naturally becomes more extensive. Coupled with the increasing complexity of the codebase, executing the entire suite for every code revision becomes impractical within the time constraints of a CI environment [6]. This poses a significant challenge to maintaining efficient and rapid testing processes.

This misalignment with the core objective of CI, which aims to provide rapid feedback to development teams for quick issue identification and resolution, can lead to delays in the feedback loop [27]. Developers may not be alerted to potential problems until much later in the development process, depending the agility CI seeks to promote. Running comprehensive regression tests, especially for large and complex software systems, demands significant computing power and infrastructure [26]. The strain on CI environments can result in higher operational costs, impacting the cost-effectiveness of the development process. To address these challenges and strike a balance between maintaining software quality and enabling rapid integration of code changes, efficient regression testing strategies and optimizations within the CI framework are essential.

In response to the challenges, the software development industry has witnessed notable evolution through the introduction of advanced, efficiency-driven regression testing techniques. Researchers have dedicated their efforts to create advanced solutions to meet the demands and objectives of modern software development processes. These solutions are designed to enhance the effectiveness and efficiency of regression testing, ensuring that software changes undergo thorough examination for defects and regressions while seamlessly aligning with the requirements and objectives of CI and Agile development paradigms. Figure 3.2 illustrates a general model of regression testing and the proposed techniques.

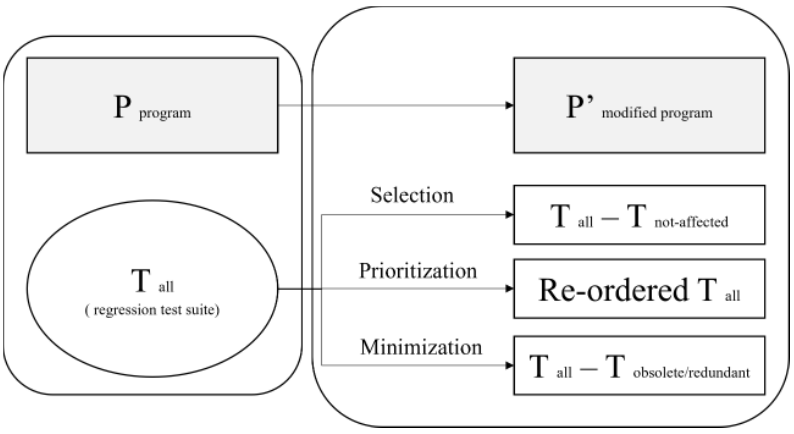

Figure 3.2. Regression testing techniques (Adapted from Haghighatkhah et al. [11]).

In this figure P represent a program, P' a modified version of the program, and T a test suite developed for P. In the transition from P to P', previously verified functionality of P may become faulty in P'. Regression testing primary objective is to validate P', ensuring that recent changes to the system have not adversely affected any previously confirmed functionalities. To enhance the efficiency of this process several techniques have been proposed such as Test Case Selection (TCS), Test Case Minimization (TCM), Test Case Prioritization (TCP), and Test Case Augmentation (TCA).

Test case minimization represents a strategic approach aimed at optimizing the efficiency of a test suite by identifying and removing redundant or obsolete test cases [6]. By eliminating redundant test cases from the original test suite, this technique can save valuable time and resources throughout the testing process. The execution overhead is significantly reduced when fewer test cases are executed, which can lead to quicker test runs, enabling faster feedback to developers and shorter testing cycles.

Test case selection involves selecting a subset of test cases from an existing test suite based on recent changes or modifications in the software [6]. This approach aims to efficiently select and execute those test cases that are most likely to be affected by the recent code changes. It helps in optimizing the testing process, ensuring that the modified code is thoroughly tested while minimizing the time and resources required to execute unaffected parts of the software.

Test case prioritization involves ordering test cases within a test suite based on various criteria, such as their perceived importance, potential impact on the software, or likelihood of uncovering critical defects [6]. The primary objective of TCP is to ensure that the most critical and high-risk test cases are executed early in the testing process, allowing for the early detection of severe issues and optimizing the testing resources by focusing on the most important areas of the software first. This approach helps in improving the efficiency of testing, reducing the time it takes to identify and address critical defects, and ultimately enhancing the overall quality of the software.

Table 3.1 provides a summarized comparison of the three primary strategies, emphasizing their key advantages and disadvantages.

Table 3.1. Comparative Analysis of Regression Testing Strategies

	Minimization	**Selection**	**Prioritization**
Focus	Reducing test suite size	Choosing relevant tests	Ordering tests effectively
Strengths	Eliminates redundancies	Targets recent changes	Enhances fault detection speed
Limitations	May overlook new risks	Might omit some relevant tests	Might be Potentially resource-intensive

While test case minimization and test case selection are essential techniques for optimizing testing efficiency by reducing the number of executed test cases, they come with the potential drawback of unintentionally eliminating test cases that cover critical parts of the software [6]. In contrast, TCP takes a different approach by strategically ordering test cases within a test suite to maximize fault detection rate. TCP aims to strike a better balance between efficient testing and comprehensive code coverage when compared to minimization and selection techniques.

The effectiveness of TCP methods in improving the cost-effectiveness and reliability of regression testing processes is well-established through empirical evidence. Seminal studies conducted by Do et al. [28], Elbaum et al. [7], and Rothermel et al. [29] offer strong validation

of TCP's advantages. These pioneering works not only emphasize TCP's capability to streamline testing procedures but also highlight its crucial role in preserving regression testing as an efficient tool for upholding rigorous software quality standards.

3.3 Test Case Prioritization

Test case prioritization techniques strategically order test cases for execution based on specific criteria, with the aim of increasing the likelihood that when these test cases are used for regression testing in the given order, they will better align with certain objectives compared to original execution orders [10]. TCP can address a diverse range of objectives, which include:

1. **Enhancing Fault Detection:** Testers may seek to increase the rate at which faults are identified, thereby revealing defects earlier during regression testing.
2. **Detecting High-Risk Faults:** The objective may be to identify high-risk faults more quickly, locating them in the testing process's early stages.
3. **Rapid Detection of Regression Errors:** Testers might prioritize uncovering regression errors linked to specific code changes early in the regression testing phase.
4. **Increasing Code Coverage:** The aim may be to accelerate the coverage of testable code in the system under test.
5. **Building Confidence in Reliability:** Testers may prioritize increasing their confidence in the reliability of the system under test at a quicker pace.

Rothermel et al. [30] defined TCP problem as follows:
Given: Test suite T, The set of permutations of T denoted as PT, and an objective function f that maps each permutation to a real number.
Problem: Find $(T' \in PT)$ such that $\forall T''(T'' \in PT)(T'' \neq T')[f(T') \geq f(T'')]$.

In this definition, PT represents the entire set of possible permutations of T. The function f is an operation that assigns a numerical value, referred to as the "award value," to each of these permutations. It is assumed that a higher numerical value indicates a more favorable or desirable outcome. This definition establishes a preferential structure, indicating that some permutations PT of T are considered better than others, as determined by the function f.

TCP techniques encompass a diverse spectrum of strategies, with each strategy customized to prioritize test cases within a test suite based on specific criteria or the data sources they utilize for test case prioritization. Yoo et al. [6] have categorized these techniques according to the information sources they use for prioritization as follows:

- **History-Based:** This technique primarily utilizes information from the historical performance and outcomes of previous test executions as its key information source for prioritizing test cases [31]. This information source includes data on which test cases have previously identified defects or proven to be effective in revealing faults in the software. By analyzing this historical data, the prioritization technique determines the order in which test cases should be executed, giving higher priority to those that have been historically more successful in detecting issues.
- **Coverage-Based:** This approach focuses on specific coverage criteria as the basis for prioritizing test cases. The primary information source used for prioritization in coverage-based techniques is the code coverage data obtained during the execution of previous

test executions [32]. This data includes information on which parts of the code (such as statements, branches, or functions) were executed by each test case. The prioritization process involves analyzing the code coverage information to identify test cases that have achieved high coverage or have covered critical code areas. Test cases that exercise code segments related to recent code changes or those associated with high-risk areas of the software may also receive higher priority.

- **Distribution-Based:** This technique leverages statistical distributions of various criteria, including execution time, resource consumption, fault detection rate, and code coverage, to determine the most effective order for prioritizing test cases [33]. It relies on historical data to analyze how these criteria are distributed across previous test runs. Test cases with extreme values or deviations from the statistical distribution, such as exceptionally long execution times or resource consumption, are prioritized differently based on their performance within this distribution. This approach aims to identify areas of concern or interest within the software and prioritize test cases accordingly, with the goal of enhancing fault detection and efficient resource allocation during regression testing.

- **Human-Based Techniques:** This technique relies on the expertise and experience of software testers and developers, rather than automated algorithms. This approach involves using human judgment to assess various factors like the criticality of features, past defect trends, changes in code, and customer requirements [34]. Testers prioritize test cases based on their understanding of the software, its use cases, and potential areas of risk. This method can be more flexible and context-aware than automated techniques, but it might also be more subjective and less consistent

- **Cost-Aware Techniques:** This technique focus on optimizing the balance between the benefits of testing and the associated costs, like time, resources, and complexity [35]. This method utilizes information such as historical test execution times, resource usage data, and the complexity of test cases to assess the cost-effectiveness of each test. By analyzing these factors, test cases are ordered with minimal expenditure of time and resources. This approach is particularly useful in large-scale projects where testing resources are limited and efficiently managing them is crucial.

- **Probabilistic Models:** This technique utilize statistical techniques to predict the likelihood of each test case uncovering defects, based on historical and contextual data [6]. These models analyze past test results, focusing on defect detection patterns and frequencies, to estimate the effectiveness of test cases. They also consider recent changes in the codebase and areas with high defect densities, as these are often more prone to new issues. By incorporating probabilities and trends from historical data, these models aim to prioritize test cases in a way that maximizes the chances of early defect detection, making the testing process more efficient and targeted.

- **Other Techniques:** Besides the commonly known prioritization techniques, there are other methods like requirement-based [36], risk-based [37], model-based [38], and cost-benefit analysis [39]. Requirement-based Prioritization aligns test cases with key software features or requirements, ensuring critical functionalities are tested first. Risk-Based Prioritization focuses on areas with higher potential risks, such as security or stability concerns. Model-Based techniques utilize software models like state machines to guide test case ordering. Customer Feedback-Based Prioritization takes into account user or client inputs, targeting aspects most valuable to end-users.

The categorization of TCP techniques by Yoo et al. [6] offers a comprehensive framework that represent the various strategies for prioritizing test cases effectively. This is instrumental in understanding the diverse methodologies and their applicability in different contexts. However, the practical implementation of these techniques depends on numerous factors that can significantly affect their efficacy in real-world scenarios. For instance, history-based techniques are particularly effective in mature software systems where there's an abundance of defect logs and historical data [40]. These techniques leverage past information to predict future areas of concern, making them ideal for well-established projects with a substantial testing background.

Conversely, coverage-based techniques are often more appropriate during the initial stages of software development. In such phases, the software undergoes frequent and substantial changes, making it crucial to ensure that new or modified code is thoroughly tested. Coverage-based prioritization focuses on maximizing code coverage, thereby ensuring that all parts of the application, especially recent changes, are adequately tested [32]. This adaptability to the development stage highlights the importance of selecting a TCP technique that aligns with the specific characteristics and needs of the software project at hand. Additionally, integrating insights from other prioritization methods, like risk-based or requirement-based approaches, can further enhance the prioritization process, tailoring it to address the unique challenges and objectives of each project.

3.4 Average Percentage of Fault Detected

One of the primary objective of TCP is to enhance the fault detection rate of a test suite to improve its ability to identify faults more effectively. Elbaum et al. [41] introduced a metric known as the Average Percentage of Faults Detected (APFD) to assess the efficiency of a test suite in terms of its fault detection capabilities. The APFD metric, described as a weighted average, is designed to evaluate the effectiveness of a test suite by measuring the proportion of faults it detects during its operation. The APFD metric ranges from 0 to 1, with higher values signifying more efficient and rapid fault detection. The APFD metric is calculated using the following formula:

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

In this formula TFi represents the position of the test case that detects the i-th fault in the ordered test suite, n is the total number of test cases, and m is the number of faults detected by the test suite.

3.5 Reinforcement Learning

Reinforcement learning is a type of machine learning where an agent learns to make decisions by performing actions in an environment and receiving feedback from those actions. Unlike in supervised learning, where a model is trained on a dataset with known input-output pairs, or in unsupervised learning, where the model discovers patterns or structures from data without explicit labels, RL involves learning to achieve a goal through trial and error [42].

The essential components of RL include the agent, the environment, actions, states, and rewards [43]. The agent in RL is not provided with the correct set of actions to take; instead, it must discover them through the consequences of its actions, typically in the form of rewards or penalties. This method mirrors the way humans and animals learn by interacting with their

environment, making it particularly suitable for tasks where the correct decisions or actions are not known in advance.

- **Agent:** The entity that engages in learning and decision-making, based on its interactions with the environment. It actively takes actions within this environment and receives feedback as rewards or penalties.
- **Environment:** The external system or context within which the agent operates, representing the world where the agent learns and acts. It is characterized by a defined state space and transition dynamics, outlining the rules and conditions under which the agent navigates and evolves its strategies.
- **State (S):** A state is a representation of the current situation or configuration of the environment. It encapsulates all the relevant information needed for the agent to make decisions. States can be discrete or continuous, and the state space is the set of all possible states.
- **Action (A):** Actions are the decisions that the agent can make to interact with the environment. The action space is the set of all possible actions that the agent can take. Actions can also be discrete or continuous.
- **Policy:** A policy is a strategy or a mapping from states to actions. It defines how the agent selects actions in different states. The objective of RL is often to find an optimal policy that maximizes the expected cumulative reward over time.
- **Reward (R):** The reward is a numerical value provided by the environment to the agent after each action. It indicates the immediate benefit or cost associated with taking a particular action in a given state. The agent's objective is to maximize the cumulative reward over time.

These components work together to enable an agent to learn and make sequential decisions in an environment to maximize the expected cumulative reward. This process involves key concepts:

- **Episode:** An episode in reinforcement learning is one complete sequence of the agent's interactions with the environment, starting from an initial state and ending at a terminal state. Each episode provides a complete experience for the agent to learn from, including both successful and unsuccessful outcomes.
- **Step:** A step refers to a single iteration within an episode where the agent takes an action based on its current state. In response, the environment provides the next state and a reward. Each step is a part of the learning process, where the agent incrementally adjusts its strategy based on the feedback (reward) received from the environment.
- **Done Flag:** The 'done' flag is a signal that indicates the end of an episode. It's a boolean value that becomes true when the episode reaches a terminal state, signaling the agent to reset and start a new episode. This flag is crucial for the agent to differentiate between ongoing learning within an episode and the transition to a new learning scenario.

Figure 3.3 illustrate how the agent interact with the environment over the sequence of time steps.

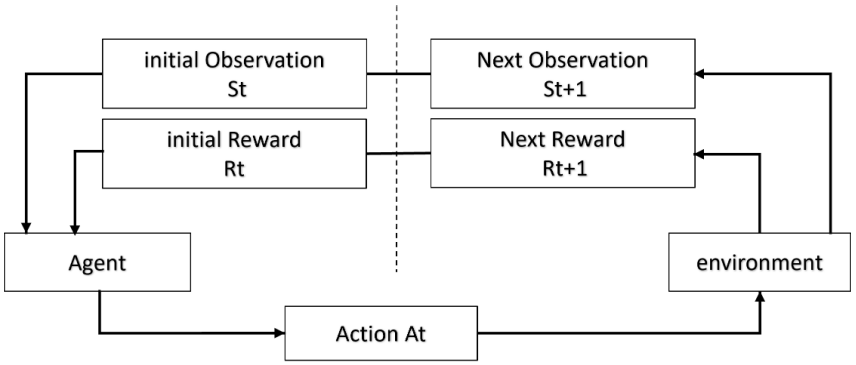

Figure 3.3. Agent environment interaction.

At each time step t, the agent interacts with its environment by perceiving and assessing its current state, and subsequently makes a an action. This action, in turn, influences the state of the environment, potentially leading to a change in the agent's surroundings [44]. As a result of this action-environment interaction, the agent receives feedback from the environment in the form of a reward signal. This reward serves as a numerical evaluation of the immediate consequences of the selected action within the given state.

This sequential process of observation, action selection, state transition between the states, and receiving reward forms the fundamental mechanism of reinforcement learning. The ultimate objective in RL is to optimize the agent's decision-making strategy, referred to as the policy, to maximize the cumulative rewards obtained over time. This cumulative reward encapsulates the notion of long-term success, encouraging the agent to learn and refine its decision-making abilities to achieve optimal outcomes in the dynamic environment.

Throughout this ongoing interaction, the agent continually adapts its policy based on the lessons learned from previous experiences, striving to make more favorable decisions in similar states or contexts in the future. This cycle of learning and adaptation enables the agent to navigate complex environments, discover optimal strategies, and achieve its goals effectively.

3.5.1 Model-based vs model-free

In reinforcement learning, two fundamental methods emerge as prominent strategies for an agent to learn and make decisions in an environment: model-based and model-free methods. Each method possesses unique characteristics, trade-offs, and practical applications.

In a model-based approach, the agent seeks to construct an internal model or representation of the environment's dynamics [45]. This model encapsulates the agent's understanding of how the environment behaves, encompassing knowledge about state transitions, action consequences, and associated rewards. Equipped with this learned model, the agent can simulate and plan potential future interactions within the environment. These simulations allow the agent to evaluate different sequences of actions and estimate their consequences, which in turn aids in making informed and optimized decisions. Model-based methods are known for their efficiency in exploration and their potential to achieve optimal policies with fewer interactions with the real environment.

In contrast, model-free approaches focus on learning directly from interaction with the environment without explicitly building an internal model. These methods prioritize estimating the value function or the policy without modeling the environment's dynamics [45]. Model-free algorithms leverage experiences accumulated through trial and error to determine the best actions and policies based on observed rewards and state transitions. This approach is particularly

suitable for environments with uncertain dynamics or when obtaining an accurate model is impractical or infeasible.

The choice between model-based and model-free reinforcement learning depends on the specific problem, available resources, and the desired balance between exploration and exploitation. Model-based approaches offer efficient exploration and can be advantageous in scenarios where data collection is costly or time-consuming. On the other hand, model-free methods are simpler to implement and well-suited for situations with uncertain or complex environments.

3.5.2 Learning Mechanism

In RL, there exist diverse approaches and strategies that enable agents to learn and adapt in complex environments. These approaches can be broadly categorized into three fundamental paradigms:

- **Value-based** approach is a foundational strategy that revolves around estimating and optimizing the value function, which assigns numerical values to states or state-action pairs within an environment [46]. This value function serves as a crucial guide for the agent, allowing it to assess the desirability of different states or actions. Value-based methods often employ iterative algorithms like value iteration or Q-learning, leveraging the Bellman equation to update value estimates progressively. These algorithms aim to converge to an optimal value function that represents the maximum expected cumulative reward attainable from any given state or state-action pair. Subsequently, value-based methods can derive an optimal policy by selecting actions that maximize the estimated values, leading to effective decision-making in complex and dynamic environments.

- **Policy-based** approach represents a distinct paradigm for solving problems by directly learning the optimal policy, a strategy that dictates the agent's actions in various states of the environment. Unlike the value-based approach, which estimates the desirability of states or actions, policy-based methods focus on modeling the policy itself [46]. The key idea is to parameterize the policy and use optimization techniques to iteratively update these parameters in a way that maximizes the expected cumulative reward over time. By directly learning the policy, these methods are particularly well-suited for handling high-dimensional and continuous action spaces, making them applicable to complex real-world tasks. Policy-based algorithms come in various forms, including policy gradients, where the policy's parameters are updated based on the gradients of expected rewards, and evolutionary algorithms, which use techniques inspired by natural evolution to improve the policy.

- **Actor-critic** approach combines the strengths of both policy-based and value-based methods to improve the efficiency and stability of learning. In this paradigm, the actor is responsible for learning and updating the policy, determining the optimal actions to take in different states of the environment. Simultaneously, the critic estimates the value function, which evaluates the desirability of state-action pairs, providing feedback to the actor regarding the quality of its decisions [46]. By jointly optimizing the policy and value function, the Actor-Critic approach accelerates the learning process, allowing the agent to make more informed and effective decisions in complex environments.

The distinction between value-based and policy-based approaches results in a fundamental trade-off in RL [47]. Value-based methods, particularly in complex environments, tend to be

more sampling efficient as they require fewer interactions with the environment to learn an effective policy. This efficiency arises because they focus on estimating the value function, which directly guides action selection based on the learned values. However, the drawback is that their convergence to the optimal policy is often proven under certain theoretical conditions that are difficult to meet in practice. For instance, these conditions may assume a complete and accurate model of the environment's dynamics or a perfectly tuned exploration strategy. Additionally, value-based methods may require precise hyperparameter tuning to perform optimally in real-world scenarios.

On the other hand, policy-based approaches offer the advantage of stability in their learning process and a guarantee of convergence to an optimal policy. They tend to be more robust in dealing with the challenges of uncertain or complex environments. However, they are often considered sample inefficient, as they require a significantly larger number of samples to achieve this convergence. Policy-based methods update the policy in smaller increments, leading to a more gradual learning progression. This gradual learning can be particularly useful when dealing with sensitive systems where exploration must be limited.

Choosing between value-based and policy-based methods involves considering the trade-offs between sampling efficiency, theoretical assumptions, and stability of learning. The decision often depends on the specific problem, available resources, and the balance between efficient learning and robust performance.

3.5.3 On-Policy vs. Off-Policy Learning

The distinction between on-policy and off-policy learning methods represents a fundamental categorization that impacts how agents learn and make decisions in their environments [48]. Each approach has its unique characteristics, trade-offs, and practical applications.

On-policy methods focus on learning and improving the policy by directly interacting with the environment and collecting data from the current policy. The central idea is to evaluate and update the policy based on the experiences generated by the policy itself [49]. On-policy algorithms often use a concept called "trajectory" or "rollout," which involves executing a sequence of actions in the environment and collecting corresponding rewards. These collected trajectories are then used to update the policy in a way that encourages actions that lead to higher cumulative rewards.

One of the main advantages of on-policy methods is their ability to ensure stable learning and convergence to a near-optimal policy. They tend to be more robust when dealing with environments that are non-stationary or involve continuous changes. However, on-policy methods can be sample-inefficient, as the data they collect is limited to the current policy, making it challenging to reuse past experiences effectively.

Off-policy methods, in contrast, decouple the learning process from the execution of actions. These algorithms allow the agent to learn from a more extensive and diverse set of experiences, including data collected from multiple policies, including past policies or even random exploratory actions [50]. The key distinction is that off-policy methods typically maintain two policies: the "target policy," which is the one being learned or improved, and the "behavior policy," which is used for interacting with the environment.

This decoupling of learning and execution provides off-policy methods with greater sample efficiency. They can reuse previously collected experiences and explore alternative actions more effectively. However, off-policy methods are more sensitive to issues related to exploration and the quality of the data collected by the behavior policy [50]. If the behavior policy deviates significantly from the target policy, it can lead to difficulties in learning an effective policy.

The choice between on-policy and off-policy depends on the specific problem, available data, and the desired trade-offs [48]. On-policy methods offer stability and convergence guarantees but may require more data collection to learn an effective policy. Off-policy methods, on the other hand, excel in sample efficiency and can leverage a broader range of experiences but may be more sensitive to exploration and data quality. Practitioners often select the most suitable approach based on the characteristics of the environment and the agent's objectives. In some cases, a hybrid approach that combines elements of both on-policy and off-policy learning, such as the Actor-Critic architecture, can provide a balanced solution that leverages the strengths of each approach while mitigating their limitations.

3.5.4 Exploration vs Exploitation

One of the fundamental challenges in RL is the exploration-exploitation dilemma, a delicate balance that agents must strike to make optimal decisions in an uncertain environment [51]. This dilemma revolves around the trade-off between two essential objectives: exploration, which involves seeking out new and unexplored actions or states to discover better strategies, and exploitation, which entails choosing actions that are currently known to yield high rewards based on existing knowledge.

Exploration is crucial in RL as it enables the agent to gather valuable information about the environment and discover more optimal policies. By trying new actions or visiting unexplored states, the agent can uncover hidden opportunities and learn from its experiences. However, excessive exploration can be risky, as it may lead to suboptimal decisions and reduced short-term rewards [51]. The challenge lies in determining how much exploration is required to ensure that the agent continues to learn and adapt effectively without sacrificing immediate performance.

Exploitation, on the other hand, involves selecting actions that are expected to yield the highest immediate rewards based on the agent's current knowledge. Exploitative strategies aim to maximize short-term gains and can be effective once the agent has gained a reasonable understanding of the environment [51]. However, relying solely on exploitation can limit the agent's ability to discover better policies or adapt to changes in the environment. It may lead to suboptimal decision-making in the long run if the agent misses out on potentially more effective actions or strategies.

Achieving a balance between exploration and exploitation is essential for successful RL. Several strategies and algorithms have been developed to address this dilemma:

1. **ϵ-Greedy:** This strategy combines exploration and exploitation by selecting the best-known action most of the time (exploitation) but occasionally exploring random actions (exploration) with a small probability epsilon [52].

2. **Upper Confidence Bound (UCB):** This method assign confidence intervals to actions and select actions with higher uncertainty to promote exploration, ensuring that all actions are explored [53].

3. **Thompson Sampling:** This is a Bayesian method, such as Thompson Sampling, maintain a probabilistic model of action values and sample from this distribution to balance exploration and exploitation [54].

The exploration-exploitation dilemma is crucial in shaping an agent's ability to learn and make decisions effectively. Striking the right balance is essential; overly cautious exploration can impede learning, while excessive exploitation can restrict an agent's ability to adapt and

discover optimal policies. Successful RL method often entails carefully selecting or designing exploration and exploitation strategies that are tailored to the specific problem and environment. This approach enables agents to navigate complex tasks and achieve optimal long-term outcomes.

3.5.5 state-of-art RL frameworks

Stable Baselines3 (SB3) is a Python library designed to conduct research and experimentation in the field of RL [55]. Serving as the successor to the previous versions, Stable Baselines and Stable Baselines2, SB3 offers a comprehensive suite of well-documented RL algorithms, making it a valuable resource for both researchers and practitioners.

SB3 includes a wide range of state-of-the-art RL algorithms that have demonstrated their effectiveness across various tasks and environments. These algorithms encompass both policy-based and value-based methods, catering to different problem domains and learning requirements. Built on top of the popular OpenAI Gym, SB3 seamlessly integrates with a diverse set of RL environments, allowing users to experiment with their agents in a standardized framework. This compatibility simplifies the process of defining custom environments for RL applications. Table 3.2 summarizes the algorithms supported by SB3:

Table 3.2. Overview of RL Algorithms from Stable Baselines3

Algorithm	**Learning Approach**	**Policy Dependency**	**Action Space**
A2C	Actor-Critic	On-Policy	Both
DDPG	Actor-Critic	Off-Policy	Continuous
DQN	Value-Based	Off-Policy	Discrete
PPO	Policy-Based	On-Policy	Both
SAC	Actor-Critic	Off-Policy	Continuous
TD3	Actor-Critic	Off-Policy	Continuous

The choice of an RL algorithm is dependent on the nature of the action space defined for the problem at hand. In RL, action spaces can be categorized as either discrete or continuous. In a discrete action space, the set of possible actions is finite and consists of distinct choices that the agent can make. Conversely, in a continuous action space, actions can assume a continuous range of values within a specified range.

4 Related Work

Test case prioritization has been an essential subject in software testing literature, gaining significant attention for its role in enhancing the efficiency of regression testing, particularly within CI environments. Over the years, TCP has evolved from a novel concept to a critical strategy in managing the complexities of modern software development [56]. Researchers have focused on developing various TCP methodologies, starting from heuristic-based approaches to more sophisticated machine learning techniques. The extensive body of research on TCP highlights its growing importance and impact in addressing the challenges of software testing in a dynamic and automated world of software development.

The concept of TCP was first introduced by Wong et al. [57]. In their seminal paper titled 'A Study of Effective Regression Testing in Practice,' they proposed a TCP technique based on code coverage capabilities. However, this initial approach was applied specifically to test cases that had already been selected through a predefined test case selection technique. This seminal work by Wong et al. laid the groundwork for future research and development in the area of TCP, marking the beginning of a more sophisticated and systematic approach to TCP in regression testing.

Inspired by this line of work, Rothermel et al. [10] conducted an empirical study to assess the effectiveness of different TCP techniques in improving fault detection rates. Their findings revealed that all tested TCP methods, including the simplest ones, enhanced fault detection rates compared to non-prioritized approaches. This study substantially contributed to the TCP field by providing empirical evidence of the efficacy of various TCP strategies, highlighting the practical utility of function-level techniques due to their lower costs and opening new possibilities for future research in optimizing TCP methodologies.

Subsequently, Elbaum and Rothermel [58] expanded upon the study by Rothermel et al. [10], broadening the scope of their empirical research to encompass additional software programs and introducing new methods for TCP. They introduced methods like function-coverage and function-level Fault Exposure Potential (FEP) to study the impact of granularity on TCP. Function-coverage was measured by counting the number of functions executed by each test case, while function-level FEP was calculated by the ratio of the number of defects identified and eliminated in a particular function of the software by a specific test case. Their research, driven by the hypothesis that coarser granularity would yield lower APFD values, was confirmed through statistical analysis.

Yoo et al. [6] provided a comprehensive categorization of TCP techniques, systematically organizing them into distinct groups based on the information source used for prioritization. These categories included coverage-based [32], history-based [40], distribution-based [33], cost-aware [39], human-based [34], requirement-based [36], and probabilistic models [41]. This structured classification significantly contributed to TCP by clarifying various strategies and their applications.

Existing research in TCP is mainly divided into heuristic-based and ML-based approaches. Heuristic-based methods, foundational in the field, often fall short in dynamic CI environments due to limited adaptability and scalability [59]. These methods struggle to keep pace with the frequent and rapid changes in CI, resulting in less effective prioritization. Additionally, their scalability is challenged in large-scale projects where the volume of test cases can be overwhelming. These limitations have led to a growing interest in ML-based approaches, which promise greater adaptability to ongoing changes and better scalability, making them more suitable for dynamic CI environments [60].

Durelli et al. [61] highlight the potential of ML in software testing. They initially proposed ML categories for TCP in regression testing. However, upon closer examination, it became clear that two of these categories, supervised and semi-supervised, lacked sufficient references, with

only one supporting reference for the latter. Consequently, the authors decided to streamline the categorization into three main ML categories: supervised, unsupervised, and reinforcement learning.

Bertolino et al. [62] conducted a comparative analysis of two distinct TCP strategies in the CI environment: Learning-to-Rank (LTR) and Ranking-to-Learn (RTL). The LTR method utilizes machine learning to systematically prioritize test cases by analyzing historical data, while the RTL approach dynamically adjusts prioritization based on the results of test executions. Their comprehensive empirical study delves into assessing the effectiveness of these methods in terms of fault detection capabilities and overall efficiency in testing. This comparison not only highlights the strengths and weaknesses of each TCP strategy but also provides valuable insights into their practical implications in enhancing the regression testing process within CI environments.

In CI, any ML-based approach for TCP must efficiently manage vast historical datasets, such as test case executions and code changes, while also dynamically adapting to system and test suite modifications as new data comes in [63]. Supervised ML techniques are adept at processing large data volumes but are limited in their continuous adaptability, often requiring a significant amount of effort, time, and resources to complete. These techniques predominantly operate within a traditional batch learning framework, assuming the complete dataset's availability before training [63].

They lack incremental learning capabilities, which would allow the seamless integration of new data into existing models. Instead, they often reconstruct models from scratch, which is a time-intensive process that can result in potentially outdated models [62]. For example, Muti Additive Regression Trees (MART), an ML method reported to be highly accurate for TCP, does not support incremental learning due to its nature as an ensemble model of boosted regression trees, primarily designed for static datasets.

Among the spectrum of proposed ML-based TCP techniques in CI environments, RL has emerged as particularly effective, especially in its adaptability to the dynamic aspects of CI environments [64]. The scalability of RL models aligns well with the demands of larger, more complex environments. As projects grow and the number of test cases increases, these models can adapt without the need for extensive reconfiguration or manual intervention. This scalability ensures that RL remains an effective tool even as the scope of the software project expands.

Spieker et al. [63] were the first to apply RL to TCP, introducing RETEC, an innovative method for the prioritization and selection of test cases for regression testing in CI environment. RETEC, being an adaptive and model-free method, utilizes RL and historical test data to prioritize test cases. The method primarily relies on a reward function to assess the performance of the prioritized test suite and incorporates test metadata such as historical results and durations. Their evaluation, which encompassed various RL agents and artificial neural networks, demonstrated RETEC's effective prioritization. It achieved results comparable to deterministic methods after approximately 60 CI cycles, without the need for prior specific training.

Subsequent research has built upon these foundational insights, delving into how different reward functions and RL environment state representations affect the efficiency and effectiveness of TCP. Bagherzadeh et al. [65] underscored the adaptability of RL algorithms in identifying test cases most likely to fail. However, the study also observed that the success of RL-based TCP heavily relies on the quality of the state representation and the design of the reward function.

Lousada et al. [66] conducted a thorough exploration of RL in optimizing TCP strategies. In their study, RL was used to adapt to evolving testing environments, leading to a significant improvement in the Normalized Percentage of Fault Detected (NAPFD), a crucial metric for evaluating TCP effectiveness. The research also drew attention to the complexities involved in fine-tuning the RL model and the challenges in developing an appropriate reward function,

highlighting the difficulties of effectively applying RL in TCP contexts.

Li et al. [67] made a significant contribution by introducing weighted reward functions for RL. This innovative approach, designed to balance the quantity of failures and their distribution, addresses a crucial challenge in TCP. The implementation of weighted rewards led to improved prioritization performance. However, the study also highlights the importance of precise design in these weight functions. Such precision is critical to avoiding biases that could negatively impact the learning process, ensuring that the reinforcement learning model remains objective and effective in its prioritization tasks.

Taking a unique approach, Shi et al. [68] focused on the security aspect of software testing. They utilized RL with the goal of more accurately predicting test case failures, aiming to enhance the effectiveness of the security testing process. One of the primary challenges identified in their research was the demand for extensive training data to reach the optimal level of test case prioritization, underscoring a crucial consideration in the application of RL to this area of software testing.

The study by Wu et al. [69] focused on a time-sensitive approach for CI environments. They designed a reward function that factored in the recency and frequency of test case failures to adapt to the fast-paced nature of CI. However, a limitation was the computational overhead required for calculating time windows for each failure event, which could potentially slow down CI pipelines. Optimizing these calculations or adjusting parameters is essential to balance real-time adaptability with computational costs.

Reinforcement learning is gaining increasing popularity in test case prioritization for regression testing within CI environments. Researchers such as Spieker et al. [63] and Chen et al. [70] have acknowledged the potential of RL to revolutionize and provide promising opportunities for improvement in the domain. Its scalability, adaptability, and effectiveness in managing large datasets make RL a compelling choice for enhancing software testing methodologies. As software projects continue to grow in complexity, RL is emerging as a valuable tool for improving the efficiency and effectiveness of test case prioritization in regression testing.

5 Model Creation Process

This thesis follows the Design Science Research Methodology (DSRM) as introduced by Peffers et al. [21]. The focus of this chapter is specifically on the 'Design & Development' step, and it is centered around addressing the first research question: 'How can Reinforcement Learning be applied to Test Case Prioritization?' In this context, we undertake a critical examination of the practicality and feasibility of implementing RL techniques within the TCP, particularly in a CI environment. This involves an in-depth analysis of the application of RL methods to enhance the process of prioritizing test cases, assessing their effectiveness and adaptability in this specific environment.

To successfully integrate TCP into the RL framework, we conceptualize TCP as a ranking problem to be addressed by the RL agent. This conceptualization involves treating TCP as a sequence of interactions between the CI environment and the RL agent, with these interactions being fundamentally driven by the fundamental components of RL: environment, observation, reward function, and action. These components are precisely designed to align with the specifics of the TCP problem. The architecture of our model creation process is detailed in Figure 5.1. This figure provides a comprehensive view of the entire process, starting from data collection and preprocessing steps, and progressing to the development of the RL-based TCP method, illustrating each stage in a clear and structured manner.

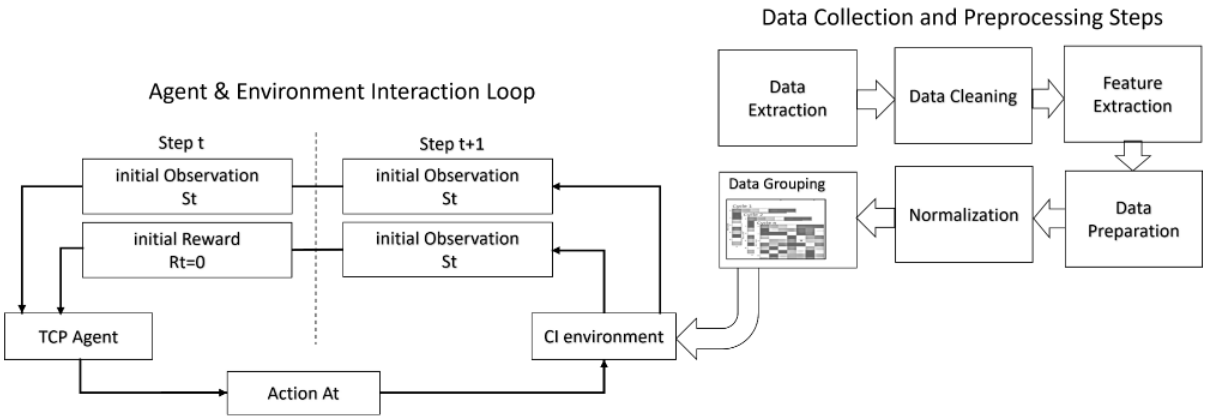

Figure 5.1. Overall architecture of RL-based TCP approach.

This chapter is divided into two distinct sections. The first section is dedicated to data collection and preprocessing, where we outline the methodologies employed for extracting and preparing data. Subsequently, we explore the application of RL to the TCP problem. In this section, we provide a comprehensive exploration that encompasses both the theoretical foundations and practical implementation of RL within the TCP context.

5.1 Data Collection and Preprocessing

The foundation of a successful machine model depends on the quality and relevance of its data [71]. Selecting data features that fail to accurately reflect the operational context can lead to models generating unreliable or misleading predictions when applied in real-world practical scenarios [72]. Fundamentally, the selection of appropriate data features, the overall quality of

the dataset, and the structure of the data are critical components that determine the reliability and effectiveness of machine learning models in real-world applications.

In this section, we present the developed methodology for data collection and preprocessing, which is essential component in ensuring that the data extracted for our model is relevant, accurate, and of high quality with a well-defined structure. Data collection and preprocessing steps are illustrated in Figure 5.2.

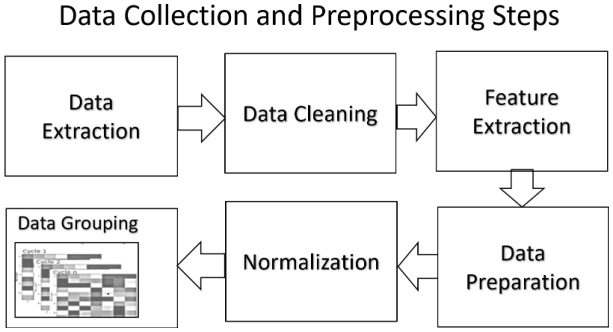

Figure 5.2. Data collection and preparation steps.

The process of data collection and preprocessing is a structured approach designed to prepare data, encompassing six distinct phases. Initially, test case execution data is extracted from the database, which is then subjected to a cleaning phase. Next, the data undergoes feature extraction, identifying features that will be used by the model. After extracting features, the data is further prepared and normalized to establish a uniform and consistent structure. The final stage involves grouping the data according to CI build ID, simplifying the analysis. Each phase of this process is conducted to enhance the data quality, a critical component for the development of a reliable machine learning model. Each stage of this process is explained in detail in the subsequent sections.

5.1.1 Data Extraction and Preparation

For extracting the test execution data from the database, we used the python script to connect and fetch the data. This step involved identifying and extracting only those data entries that reveal patterns of test case failures and their temporal dynamics including:

- **Test Case ID:** This serves as a unique identifier for each test case, making it easier to track.
- **Test Case Status:** The current status (e.g., pass or fail) of the test case is crucial for prioritization, with priority given to the failed test cases over the passed ones.
- **Start Time:** Used to extract temporal features, providing insights into when the test case execution began.
- **End Time:** Also used for extracting temporal features, indicating the completion time of the test case execution.
- **Cycle ID:** This data aim to group test cases that belong to the same CI build cycle, enabling the prioritization of test cases that are associated with the same CI build.

The extracted data is subsequently subjected to a cleaning process to eliminate CI cycles that included test cases missing any of the essential data mentioned above. This action ensures that all test cases within the CI cycles have complete and relevant data, which is crucial for accurate prioritization.

5.1.2 Feature Extraction and Preparation

Feature extraction is an essential step as it transforms the data into a format that highlights the most relevant aspects of the data, enabling the method to make informed decisions and predictions. We carefully selected and engineered features that have the potential to capture patterns, dependencies, and characteristics essential for effective TCP. These features include:

- **Test Case Duration:** This feature involves calculating the duration of each test case by determining the time difference between their start and end times. This information is crucial because it provides insights into the efficiency and performance of each test case. Longer durations might indicate more resource-intensive test cases that could impact testing efficiency.
- **Failure History:** This feature involves creating a list of test case statuses from previous records to identify patterns of failure. Understanding the history of test case failures is essential for prioritization. Test cases with a consistent history of failures are more likely to be prioritized higher, as they could indicate recurring issues in the software.
- **Last Run:** This feature indicates the last time that the test case has been executed. Understanding the recency of a test case's execution is valuable because it helps in assessing its relevance. Test cases that have been executed more recently are assigned a higher priority, as they are more likely to uncover issues in the current version of the software.

After the feature extraction process, each test case now includes the following features: Test Case ID, Test Case Status, Cycle ID, Test Case Duration, Failure History, and Last Run. These features then underwent an additional data transformation process, which, while not strictly falling under the category of data cleaning, played a significant role in preparing the data for subsequent analysis.

1. **Binary Transformation:** This step involves converting the alphabetic values of the test case statuses, namely 'FAIL' and 'PASS', into binary values '1' and '0', respectively. This transformation is essential because binary data is more straightforward to work with in many analytical and machine learning algorithms, simplifying computations and comparisons. It reduces the complexity of the data and improve the efficiency and effectiveness of the analysis process.
2. **Standardization of Test Case Durations:** This step involves optimizing the dataset and ensuring uniformity by computing the average duration for each test case. Instead of assigning distinct durations for identical test cases, we assigned the average duration to each occurrence of the same test case. This process will improve dataset consistency and uniformity.
3. **Normalization:** This step involves normalizing the extracted features utilizing the Scikit-Learn library in Python. Normalization assumes a critical role as it scales all features to

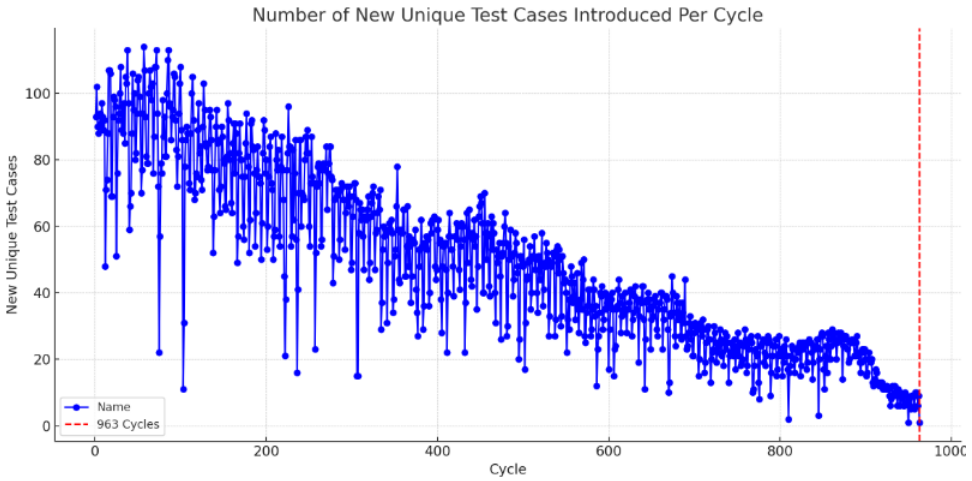

Figure 5.3. Rate of newly introduced test cases per cycle.

a consistent range, a fundamental practice in machine learning. The purpose of uniform scaling is to prevent any individual feature with a larger numerical range from exerting disproportionate influence on the model's predictive outcomes. This step is imperative for preserving the accuracy and reliability of the model, thereby augmenting its resilience and efficacy in predictive capabilities.

4. **Data Grouping:** The decision to group test cases belonging to the same CI build cycle into cohesive sets was made to closely replicate real-world scenarios. In practical software testing, test cases within the same CI build cycle are often executed together, reflecting the interconnected nature of testing efforts for a specific release or iteration. This approach enhances the alignment of the data with the practicalities of software testing processes.

These data preparation and feature engineering processes directly impact the model's decision-making abilities. They ensure data quality, prevent undue bias, enhance adaptability, and align the model with practical scenarios, ultimately leading to more accurate and relevant decisions.

5.1.3 Data Window Selection for Training

Selecting a specific window size for data extraction in machine learning model enhances efficiency by reducing computational demands and allows for more focused analysis of recent and relevant patterns, particularly valuable in dynamic environments. Additionally, window-based approaches facilitate memory management, adaptability to changing data, noise reduction, and dynamic model evaluation. This practice optimizes resource utilization and leads to more effective modeling and decision-making in scenarios where the entire dataset may not be relevant.

The strategy for selecting the window size in the RL-based TCP method in a CI environment is based on monitoring the rate of newly introduced test cases in each cycle. When this rate approaches zero, it signifies that the system's dynamics are stabilizing, indicating that the RL agent has gained a comprehensive understanding of the test cases. Figure 5.3 illustrates this concept.

The graph illustrates a decreasing trend in the number of new unique test cases introduced as the number of cycles increases. Initially, during the early stages of training, new test cases are introduced, enabling the agent to explore different aspects of the system. However, as it progresses towards the 936th cycle, the number of new unique test cases approaches zero. This

suggests that the system has reached a point where the agent has developed a comprehensive understanding of the test cases behavior.

From the perspective of the agent, it initially encounters a diverse range of test scenarios, each providing new insights into their behavior and failure patterns. This accumulation of valuable data enhances the agent's understanding of the test cases dynamics. As testing progresses, particularly by the 963rd cycle, the agent develops a comprehensive understanding of the system, enabling it to proficiently identify and prioritize the most critical test cases.

5.2 Implementation

In this section, we explore the integration of TCP with the RL framework, as depicted in Algorithm 1. The algorithm begins by initializing the reward, observation, and the 'done' flag, marking the start of a new learning episode. Additionally, the variables index0 and index1 are assigned the values 0 and 1, respectively.

Setting the reward to zero at the start ensures that the agent has no initial bias towards certain actions or states. This neutrality enables the agent to explore the environment and learn through experience, free from any prior expectations about the value of different actions. This make the agent to form optimal strategies based on its interactions with the environment, which leads to more effective and unbiased learning outcomes.

The 'done' flag, which is initially set to 'False,' indicates that the episode is ongoing, allowing the agent to continue interacting with the environment. When set to 'True' (1), it signifies the completion or termination of the current episode or task. In this algorithm, the 'done' flag is switched to 'True' under two specific conditions:

1. If (index1 == length(T') - 1): This condition checks whether index1 has reached the end of the list, indicating that it points to the last test case in the test suite T'.

2. Simultaneously, if (index0 <length(T') - 2): This condition checks whether index0 is less than (length(T') - 2), indicating that index0 must be pointing to the second-to-last test case in the test suite.

The interaction loop between the agent and the environment initiates when the environment presents the agent with the initial observation alongside the corresponding reward. This initial observation consists of a pair of test cases, selected from the start of the shuffled test suite T', represented as [T'[index0], T'[index1]]. Upon receiving this observation, the agent must take an action based on the information provided. In the context of pairwise ranking comparison, the action space is binary: either choosing '0' or '1'. Opting for '0' indicates prioritizing the test case at index zero, whereas choosing '1' signifies that the second test case in the observation pair is given higher priority over the first.

Initially, the agent engages in exploration, taking random actions to understand different outcomes and accumulate information about the environment. This phase is critical for the agent to develop a fundamental understanding of the impact of its actions on rewards. As the agent gains knowledge and refines its decision-making policy, it gradually shifts towards exploitation, where it makes more informed and strategic choices. During this phase, the agent favors actions that are more likely to yield higher rewards or more favorable outcomes, reflecting its learning and adaptation to the environment.

Algorithm 1: RL-Based Pairwise Ranking Algorithm for TCP

Input : T: Original test suite
Output : T'': An ordered sequence of test cases in T
:

$T' \leftarrow$ Randomized order of T
$index0 \leftarrow 0$ /* First test case index */
$index1 \leftarrow 1$ /* Second test case index */
$done \leftarrow$ False
$reward \leftarrow 0$
$obs \leftarrow [T'[index0], T'[index1]]$ /* Current pair for comparison from T' */
Invoke episode() /* Episode starts */
Function `step`($action$)
    $reward \leftarrow$ calculate_reward_based_on_action($action$)
    if $action == 1$ **then**
        swap_elements(T', $index0$, $index1$)
    if $index1 < length(T') - 1$ **then**
        $index1 \leftarrow index1 + 1$
    else if $index1 == length(T') - 1$ ***and*** $index0 < length(T') - 2$ **then**
        $index0 \leftarrow index0 + 1$
        $index1 \leftarrow index0 + 1$
    else
        $done \leftarrow$ True
    $obs \leftarrow [T'[index0], T'[index1]]$
    return $done, reward, obs$
Function `calculate_reward_based_on_action`($action$)
    $priority\_test\_case \leftarrow obs[action]$
    $non\_priority\_test\_case \leftarrow obs[1 - action]$
    if $priority\_test\_case.verdict == 1$ ***and*** $non\_priority\_test\_case.verdict == 0$ **then**
        return *1*
    else if $priority\_test\_case.verdict == 0$ ***and*** $non\_priority\_test\_case.verdict == 1$ **then**
        return *0*
    else if $priority\_test\_case.exec\_time \leq non\_priority\_test\_case.exec\_time$ **then**
        return *0.75*
    else
        return *0* /* Default reward */
    else if $priority\_test\_case.failure\_count \leq non\_priority\_test\_case.failure\_count$ **then**
        return *0.5*
    else
        return *0* /* Default reward */

The step function within the environment employs a selection sort algorithm to systematically organize test cases through pairwise ranking comparisons. This sorting process is facilitated by maintaining two distinct subarrays within the main array of test cases: one that is sorted and another that remains unsorted. Initially, the entire test suite is in an unsorted state. The sorting commences with 'index0' serving as a boundary marker, initially positioned at the start of the array. At this point, the sorted portion is empty, while the unsorted portion encompasses all test cases.

During each iteration of the sorting process, the unsorted subarray undergoes shuffling, and the first two test cases are passed to the agent. The agent, upon receiving this pair of test cases, takes action based on both the pair and the reward obtained from the previous iteration. This action effectively shifts the boundary marker by incrementing index0, expanding the sorted subarray, and concurrently reducing the size of the unsorted portion. This iterative process persists as the algorithm consistently identifies the next highest-priority test case within the unsorted section, placing it into its correct position within the sorted subarray. With each iteration, the test case order is refined, progressively transitioning from a completely unsorted state to a fully sorted one.

The algorithm iteratively arranges all test cases based on the selection and sort algorithm, ensuring that each pair of test cases is thoroughly evaluated. This systematic approach guarantees comprehensive coverage and comparison of all test cases, enabling the agent to develop and learn optimal prioritization strategies for the entire test suite. The end result is a well-organized array of test cases, sorted according to their determined priorities, which is instrumental in aiding the learning agent's decision-making process.

5.2.1 Action Space

At each time step, the agent's action space is binary, offering two discrete actions:

1. Action 0: Assign higher priority to the first test case in the current pair.
2. Action 1: Assign higher priority to the second test case in the current pair.

When the agent takes an action, it's essentially making a decision on which test case to prioritize based on its current knowledge and the policy it's following. The chosen test case is then placed before the other in the ordered sequence. The binary nature of the action space simplifies the learning process. Over time, by receiving rewards from the environment, the agent refines this policy to make better prioritization decisions.

5.2.2 Observation Space

The observation space consists of pairs of test cases and their respective features, which are derived from a randomly shuffled test suite denoted as T'. This arrangement forms the observation space [T'[index0], T'[index1]]. Once the agent takes an action, the observation space is updated to present the next pair of test cases. This update is orchestrated by a selection and sort algorithm, where test cases that the agent has prioritized higher are positioned to the left of index0, constituting the sorted section. In contrast, the unsorted portion, located to the right of index0, contains test cases that have been assigned lower priority.

To generate the next observation, the test cases on the right side of the list, separated by index0, undergo shuffling. By selecting the first two test cases from this shuffled portion, the agent is then presented with the subsequent observation of the environment. This systematic process ensures a clear progression of observations for the agent as it refines its prioritization strategy.

5.2.3 Reward Function

The reward calculation is determined by a series of conditional statements:

- If higher priority is given to the test case with a verdict of 1 (indicating a fail) and lower priority to the test case with a verdict of 0 (indicating a pass), the reward is set to 1. This indicates a favorable decision by the agent.
- If higher priority is given to the test case with a verdict of 0 (pass) and lower priority to the test case with a verdict of 1 (fail), the reward is set to 0. This indicates an unfavorable decision by the agent.
- If the verdicts of the test cases are the same and the execution time of the higher priority test case is less than or equal to the execution time of the lower priority test case, the reward is set to 0.75. This favors the prioritization of a test case with a shorter execution time.
- If the verdicts of the test cases are the same and the execution time of the higher priority test case is longer than or equal to the execution time of the lower priority test case, the reward is set to 0. This indicates an unfavorable decision by the agent.
- If the verdicts and execution times of the test cases are the same, and the higher priority test case has a failure count less than or equal to the lower priority test case, the reward is set to 0.5. This indicates that the agent made a reasonable decision based on failure history.
- If none of the above conditions are met, the default reward is set to 0. This indicates that the agent's decision does not receive a positive reward.

This reward function assesses the agent's decision-making in prioritizing test cases, taking into account their verdicts, execution times, and failure counts. The structured rewards aim to guide the agent in learning to prioritize test cases more effectively, thereby maximizing its cumulative reward over time.

6 Results

This chapter presents the results of the research, adapting the Design Science Research Methodology (DSRM) as introduced by Peffers et al. [21], with a focus on the evaluation stage within this framework. The results demonstrate the hypotheses, methodologies, and analytical rigor we adopted and explored throughout the research.

In this chapter, we aim to address research question 'RQ2. Which one of the Reinforcement Learning (RL) algorithms: Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), and Deep Q-Network (DQN), perform better compared to the randomly ordered test suite?'

We have selected random prioritization as our benchmark for comparison, renowned for its unbiased nature in assessing advanced TCP methods. This approach typically yields an APFD score around 0.5, establishing a standardized foundation for evaluating TCP algorithms and streamlining comparisons.

6.1 Visual Representation of Algorithms Performance

In this section, we perform a comprehensive analysis using violin plots to visually represent the APFD distributions associated with the RL algorithms utilized in this thesis: A2C, PPO, and DQN. These plots enable us to visually assess the individual performances of these algorithms and compare them against a benchmark of random prioritization. This comparative analysis yield insights into the performance patterns exhibited by these algorithms. By comparing the results of each algorithm with the benchmark, we can gain an initial understanding of how these RL algorithms perform in terms of fault detection, laying the foundation for deeper investigations into their effectiveness and their potential impact.

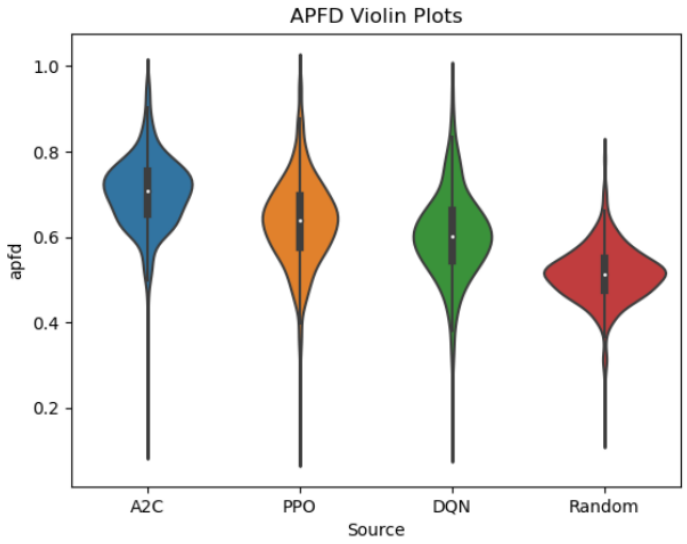

Figure 6.1. A2C, PPO, and DQN versus random prioritization.

The violin plot depicting the A2C algorithm performance reveals a significant concentration of APFD values in the upper range, highlighting the algorithm's effectiveness in fault detection. A2C utilizes an actor-critic architecture, consisting of two essential components: the actor, responsible for policy implementation, and the critic, tasked with evaluating the value function

[73]. The critic component of A2C leverages temporal-difference learning to refine the value function by considering the disparity between expected and actual rewards. This capability enables A2C to quickly adapt to optimal prioritization strategies in response to the dynamic nature of the CI environment. In addition, this dual approach equips A2C with the ability to effectively balance the exploration of new prioritization strategies with the exploitation of proven ones.

PPO, ranking as the second most effective algorithm, exhibits a wider range of APFD values compared to A2C. PPO is designed to avoid drastic policy updates, which is achieved through its clipping function in the objective function. This function limits the size of policy updates, ensuring that the new policy isn't radically different from the old one [74]. While this approach promotes stability and consistency in learning, it can sometimes be a drawback in rapidly changing environments like CI, where adapting quickly to new information is crucial. The conservative nature of PPO in updating policies might result in slower adaptation to new patterns of faults, leading to lower APFD values compared to more aggressive algorithms.

The APFD scores provided by the DQN algorithm are slightly above the central part of the violin plot, a result attributed to its foundation in deep learning [75]. DQN's capability in handling high-dimensional input spaces is a key advantage in complex test environments. However, this same complexity can present challenges. The algorithm may find it difficult to cope with the variety of test scenarios typical in CI environment. In addition, the incorporation of experience replay in DQN, designed to break the correlation between consecutive learning samples, might result in slower adaptation to the most recent changes in test cases. Moreover, the balance between exploration and exploitation, managed through techniques like ϵ-greedy, may not be ideally suited for the quickly changing nature of CI environment.

The APFD values falling below 0.5 in violin plots of A2C, PPO and DQN highlights the unique challenges that RL algorithms encounter due to the exploration-exploitation dilemma. These suboptimal results are largely a consequence of the inherent need for these algorithms to explore new actions. Exploration is a fundamental aspect of the learning process in RL, requiring a careful balance between experimenting with novel strategies and utilizing established, effective ones. However, this exploration phase, while critical for the algorithm's long-term adaptation and learning, can lead to suboptimal decision-making in the short term.

The unpredictability associated with new actions further complicates the algorithm's capacity to consistently make optimal decisions. As a result, during this exploration phase, the APFD score might temporarily decline, reflecting a period of reduced efficiency. This phase is a natural part of the RL algorithm's progression as it learns to navigate the dynamic CI environment, yet it accounts for the instances of lower APFD values observed in the data.

6.2 Statistical Analysis

Statistical analysis plays a pivotal role in evaluating the performance of RL algorithms against the benchmark of random prioritization. While the violin plots presented in Section 6.1 provide visual insights, undertaking a comprehensive statistical analysis is essential for accurately determining the significance of the observed performance differences. The primary aim is to validate the results derived from each RL algorithm through rigorous statistical examination, encompassing both statistical testing and the measurement of effect sizes.

6.2.1 Shapiro-wilk Test

To select the appropriate statistical test for comparing RL algorithm performances against the benchmark of random prioritization, we initially perform the Shapiro-Wilk test. This test calculates the W statistic, indicating the degree of conformity of a dataset to a normal distribution. A W statistic closer to 1 suggests data resembling a normal distribution. The test also provides a p-value, with a low value (usually <0.05) indicating non-normal data and leading to the rejection of the normality null hypothesis.

Table 6.1. Shapiro-Wilk Test Result

Algo	**W Statistic**	**P-value**	**H0 hypothesis**
A2C	0.9653	2.9333e-14	reject H0
PPO	0.9853	3.6417e-8	reject H0
DQN	0.9862	8.4260e-8	reject H0
random	0.9578	8.5055e-10	reject H0

Table 6.1 presents the outcomes of the Shapiro-Wilk test applied to the APFD data. These findings robustly reject the null hypothesis of normality for the APFD data points associated with A2C, PPO, DQN, and random prioritization. The W statistic, falling below the critical threshold of 1, coupled with exceedingly small p-values, collectively suggest that these samples do not originate from a normal distribution.

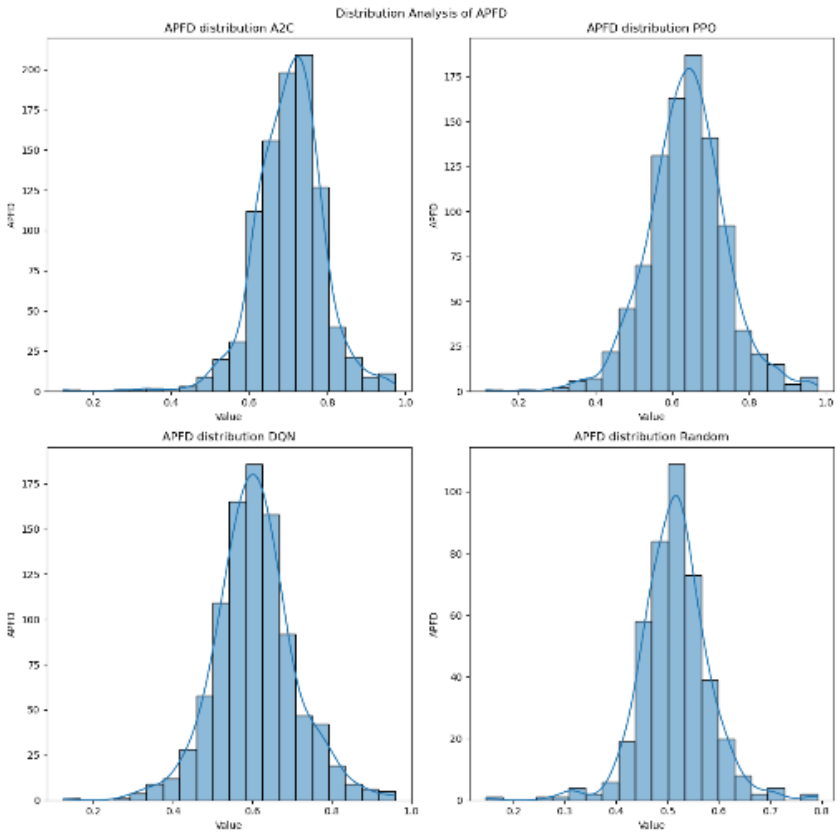

Figure 6.2. APFD distributions.

Figure 6.2 illustrates the APFD distributions graphically. Both the Shapiro-Wilk test and the histograms confirm the non-normal distribution of the APFD data. The test provides statistical evidence, while the histograms visually illustrate this with their noticeable skewness, leptokurtosis, and presence of outliers, deviating from the typical bell-shaped curve of a normal distribution. This combination of statistical and visual evidence strongly supports the non-normality of the APFD data points.

6.2.2 Statistical Test

Given the non-normal distribution of the data, using a non-parametric statistical test like the Mann-Whitney U test is appropriate for comparing the performance of RL algorithms against random prioritization. This test is ideal for scenarios with two distinct groups and is particularly effective when the dependent variable, such as APFD, is ordinal or continuous but not normally distributed. This approach is well-suited for our analysis, which involves assessing each RL algorithm against a benchmark, effectively creating multiple comparisons of two independent data groups.

The Mann-Whitney U test calculates two elements: the U-statistic and the p-value. The U-statistic, calculated by ranking all data across both groups and summing these ranks for each group, quantifies the degree of separation between the groups based on their ranked values. A lower U-statistic indicates a greater difference or distance between the groups in terms of their ranked values, while a higher U-statistic suggests less separation. A low p-value indicates that the observed differences are unlikely to have occurred by random chance alone, providing evidence against the null hypothesis and suggesting that there is a significant difference between the groups being compared. Conversely, a high p-value suggests that the observed differences could reasonably occur by random chance, leading to the acceptance of the null hypothesis, which implies no significant difference between the groups.

Table 6.2 illustrates the outcomes of the Mann-Whitney U test, which compares the performance of A2C, PPO, and DQN against the benchmark of random prioritization using APFD data.

Table 6.2. Mann-Whitney U Test Results

Algo	U-statistic	P-value
A2C	75091.0	1.232×10^{-217}
PPO	168507.0	4.7252×10^{-124}
DQN	231583.5	8.80477×10^{-76}

The results presented in the Table 6.2 clearly demonstrate the effectiveness of the algorithms A2C, PPO, and DQN when compared to the standard benchmark of random prioritization. The high U statistic and correspondingly low p-values indicate that there is a significant difference between the two groups, despite the apparent similarity in their ranked values. Specifically, the A2C algorithm, with its U-statistic of 75091.0 and an extremely low P-value of 1.232×10^{-217}, indicates a significant deviation from random prioritization, suggesting a highly effective prioritization strategy. Similarly, the PPO algorithm, with a U-statistic of 168507.0 and a P-value of 4.7252×10^{-124}, and the DQN algorithm, with a U-statistic of 231583.5 and a P-value of 8.80477×10^{-76}, also demonstrate significant improvements over random prioritization.

6.2.3 Effect Size Analysis

After confirming a statistically significant difference in performance between the RL algorithms and random prioritization through the Mann-Whitney U test, we proceed to quantify the magnitude of this difference using Vargha-Delaney A (VDA) analysis. VDA analysis provides a measure of the extent of performance disparity between the RL algorithms and random prioritization.

The VDA value ranges from 0 to 1, representing the probability that a randomly selected observation from one group will have a greater value than one from another group. A VDA of 0.5 indicates no effect, implying similarity between groups. Values above 0.5 imply a higher likelihood of greater values in the first group, while values below 0.5 suggest the opposite. VDA provides a clear and interpretable way to understand the size and direction of group differences, aiding practical decision-making in statistical analysis.

Table 6.3. VDA Effect Size vs. Random Prioritization

Algo	**Effect Size**
A2C	0.91692
PPO	0.813574
DQN	0.743829

Table 6.3 represents the results of the VDA effect size analysis, underscoring the superior performance of the three examined RL algorithms: A2C, PPO, and DQN, when compared to the standard baseline of random prioritization.

The A2C algorithm stands out with an impressive VDA effect size of 0.91692, signifying a high likelihood of outperforming the baseline. This substantial effect size provides strong evidence of its effectiveness in test case prioritization. Similarly, the PPO algorithm, with a VDA effect size of 0.813574, also significantly outperforms the baseline, although not to the same degree as A2C. The DQN algorithm, with a slightly lower VDA effect size of 0.743829, still demonstrates a noteworthy improvement over the baseline.

6.2.4 Computational Complexity Analysis

Having demonstrated the performance difference among RL algorithms through both visual and statistical analyses, with the A2C algorithm emerging as the top performer, our attention now shifts to a crucial aspect: time efficiency. This consideration is essential in determining the most suitable algorithm among A2C, PPO, and DQN for integration into the CI environment. In this context, we have recorded the training and testing times of each algorithm during their runtime. These time metrics are of essential as they serve as an indicators, reflecting the efficiency of each algorithm.

- **Training time** signifies the duration required by an algorithm to learn from the provided data and optimize its parameters. This metric is critical as it highlights the computational load and efficiency during the learning phase. A shorter training time implies quicker model convergence and reduced computational demands, which is particularly valuable in resource-constrained CI environment.

- **Training time** signifies the duration required by an algorithm to learn from the provided data and optimize its parameters. This metric is critical as it highlights the computational

load and efficiency during the learning phase. A shorter training time implies quicker model convergence and reduced computational demands, which is particularly valuable in resource-constrained CI environment.

Table 6.4 presents the average training and testing times for the A2C, PPO, and DQN algorithms.

Table 6.4. Time Complexity Analysis

Algo	**Training Time (ms)**	**Testing Time (ms)**
A2C	54129	322
PPO	62667	199
DQN	56333	219

A2C stands out as the most time-efficient algorithm during the training phase, requiring only 54,129 ms to learn from the data. This quality makes it highly valuable in resource-constrained environments or situations where rapid model updates are crucial. In testing, however, A2C exhibits a slightly longer time of 322 ms, which, while competitive, is slightly behind PPO and DQN.

PPO closely follows A2C with a training time of 62,667 ms. Although slightly longer than A2C, it maintains its competitive edge and offers excellent training efficiency. PPO excels in testing time efficiency among the algorithms, requiring only 199 ms to make predictions or decisions based on new data post-training. This highlights PPO's suitability for applications prioritizing quick decision-making in real-time scenarios.

DQN, with a training time of 56,333 ms, falls between A2C and PPO in terms of training efficiency. Similarly, DQN shows efficient testing, with a time of 219 ms, closely matching PPO in testing efficiency.

7 Discussion

In this chapter, we summarize the findings of the study, setting the stage for a deeper understanding of the topic. This chapter is further divided into three sections. First, we will provide the answers to the research questions, highlighting the insights gained through rigorous analysis. Following this, the results are integrated into the existing literature, identifying how this research contributes to the knowledge base. Subsequently, we will explore the implications of these findings, their relevance in practical scenarios. Finally, acknowledging the limitations of the study, we propose directions for future research to build upon these findings.

7.1 Research Questions Answers

- **RQ1. How can Reinforcement Learning (RL) be applied to Test Case Prioritization (TCP) in Continuous Integration (CI) environment?**
  To integrate TCP into the RL framework, we approach TCP as a ranking problem to be addressed by an RL agent through pairwise ranking comparison. The detailed agent-environment interaction is precisely implemented through the careful design of fundamental RL components such as environment, observation, action, and reward function to align with the TCP problem. For the algorithm implementation in this thesis, we utilized Stable Baselines3, a python library that provides a collection of easy-to-use implementations of RL algorithms.

- **RQ2. Which RL algorithm, A2C, PPO, or DQN, outperforms the others when compared against the baseline of random prioritization?**
  In comparing A2C, PPO, and DQN RL algorithms to random prioritization, the study found significant performance differences. A2C outperformed with an average APFD values of 0.78, followed by PPO and DQN with 0.65 and 0.6, respectively. The Mann-Whitney U test confirmed these differences with very low p-values. A2C also had the highest effect size (0.91692) compared to PPO 0.813574) and DQN (0.743829). In terms of time efficiency, A2C with fastest in training (54,129 ms) and slightly longer in testing (322 ms). PPO with a training time of 62,667 ms and was fastest in testing (199 ms). DQN's training time was 56,333 ms with a testing time of 219 ms. Overall, A2C excelled in fault detection capability and training efficiency, making it ideal for CI environment requiring quick model updates.

- **RQ3.What limitations have been identified in using Reinforcement Learning (RL) for Test Case Prioritization (TCP) in Continuous Integration (CI) environment?**
  We encountered two notable limitations in the design of RL algorithms: firstly, the challenge of fine-tuning hyperparameters, which involves a time-consuming and computationally intensive process of finding the optimal settings for parameters. Secondly, designing distinct reward functions tailored to each algorithm's learning mechanism can also be time-intensive, often demanding domain expertise. This complexity becomes particularly relevant when working under time constraints, highlighting the need for automated hyperparameter optimization methods and efficient approaches to generate reward functions.

7.2 Comparative Analysis

This thesis signifies a substantial departure from conventional heuristic-based and traditional machine learning approaches often employed for TCP in CI environment. It introduces dynamic,

responsive, and intelligent TCP solutions that adeptly address the challenges of adaptability while also eliminating the need for exhaustive analysis and time-consuming prioritization strategies in CI environment. This innovative approach not only marks a significant advancement in the field but also holds the potential to greatly enhance the efficiency and reliability of TCP in CI environment. These environments, characterized by their ever-evolving nature, require the adoption of more intelligent solutions to meet their evolving demands.

The integration of RL into TCP, pioneered by Spieker et al. [63], represented a significant advancement over traditional heuristic-based and machine learning-based approaches. Spieker et al. introduced the RETEC method for test case selection and prioritization, revolutionizing the field of software testing by providing a flexible approach capable of adapting its selection and prioritization strategy to the dynamic nature of CI environment. However, they encountered challenges stemming from the constrained and homogeneous data available for effectively training the developed method.

Wu et al. [69] also encountered challenges in their application of RL to TCP due to limitations related to the scope and diversity of the data for training the developed method. These limitations presented a significant challenge in harnessing the complete potential of RL within the context of CI environment. Such constraints in data diversity and extent posed considerable challenges for RL methodologies, impeding their ability to effectively develop and refine the requisite capabilities for optimal performance.

Researchers, including Lionel et al. [65], Busjaeger et al. [76], and Lima et al. [77], rely on standard open-source data such as Paint Control, IOF/ROL, and the Google Open-Source Data Set (GSDTSR) for their empirical investigations in this field. Lionel et al. [65] noted that these datasets often have limited scope, featuring restricted feature diversity, product variation, and an imbalanced distribution of failure rates. This lack of representative data makes it challenging to compare techniques consistently across different datasets and may not accurately reflect real-world scenarios.

In contrast to existing studies, this thesis leverages a comprehensive and robust dataset provided by Nokia. This dataset enables us to harness the full potential of the developed RL-based TCP method. We were able to determine the optimal data window size for the training phase of our RL algorithms, rather than being restricted by the quantity of the available data. The strength of our approach is further underscored by the utilization of rigorous training data sourced from a large-scale, real-world CI dataset. This extensive empirical validation represents a remarkable advancement in the application of RL to the field of software testing.

Leveraging this extensive and high-quality dataset leads to substantial improvements in TCP accuracy within a CI environment. The A2C algorithm's accuracy performance closely matches MART [62], a renowned benchmark in TCP for its high accuracy. While emphasizing accuracy, it's worth noting that MART relies primarily on static data, lacking the essential adaptability crucial in a CI environment. In contrast, our method not only closely approaches MART's accuracy but also possesses the capability to adapt its learning strategy, an essential component for TCP in the dynamic context of a CI environment.

7.3 Implications

The findings of this study hold significant promise for the software development landscape. The developed RL-based TCP method not only enhances software quality by identifying and addressing defects earlier in the development process but also contributes to substantial cost savings. Addressing issues during the development phase is notably more cost-effective than post-deployment corrections. Moreover, integrating this methodology within CI processes

accelerates development cycles, providing agility in responding to code changes and ultimately leading to faster software releases. This advantage is particularly critical in today's fast-paced markets, offering a competitive edge.

One standout implication of this study lies in the adaptability facilitated by reinforcement learning. Unlike static prioritization methods, reinforcement learning allow the developed TCP method to dynamically adjust its prioritization strategy. This adaptability proves invaluable in the context of CI, where code changes frequently. Continuously learning and adapting to evolving code and test scenarios, this approach ensures that the testing process remains effective and aligned with the ever-evolving requirements of the software development environment. Beyond just enhancing fault detection rates, it streamlines the testing process, optimizing resource allocation, and improving overall efficiency in software development.

7.4 limitations and Future Work

In this work, significant progress has been achieved in advancing TCP through the application of RL algorithms in a CI environment. However, as with any research, our work is not without its challenges and limitations. In this section, we will delve into the specific limitations encountered during the development phase, highlighting areas where further improvement and exploration are required. These limitations are essential to consider, as they shape the trajectory of future research and inform our roadmap for enhancing the efficacy of our approach.

1. **RL Hyperparameters tuninging:** Hyperparameters are critical in reinforcement learning as they dictate an algorithm's behavior and performance. In this work we have use default hyperparameters, gaining insights into TCP algorithms. Tuning these hyperparameters is complex due to the need to balance trade-offs without prior knowledge of the environment. Finding the right settings is challenging because of the high-dimensional search space, potential interactions between hyperparameters, and the computational cost of RL experiments.

2. **Reward Function:** Due to time constraints, we designed only one reward function for all the RL algorithms in this thesis. However, creating distinct reward functions for each RL algorithm, tailored to their individual learning mechanisms, is essential because different algorithms rely on unique approaches to explore and adapt to environments. A one-size-fits-all reward function may not suit these variations, potentially leading to suboptimal performance. Customizing reward functions to match each algorithm's characteristics ensures more effective learning and optimal outcomes.

Future work can significantly enhance the effectiveness of the developed method by addressing two key areas. Firstly, given the crucial role of hyperparameters in reinforcement learning, there is a clear need to explore more advanced hyperparameter tuning methods. Techniques such as grid search, Bayesian optimization, or leveraging tools like Optuna [78] can be investigated to fine-tune hyperparameters for each RL algorithm, potentially resulting in improved performance and faster convergence.

Secondly, to further enhance the method's efficacy, it is essential for future research to delve into the customization of reward functions for each RL algorithm. Tailoring reward functions to align with the unique learning mechanisms of each algorithm can facilitate better adaptation and more efficient learning. Achieving this customization requires a thorough understanding of each algorithm's requirements, enabling the alignment of the reward structure accordingly. These complementary direction of research hold the potential to significantly advance the

field of reinforcement learning and contribute to the optimization of RL algorithms in various applications.

8 Conclusions

In this work, we explored the integration of the TCP problem into a RL framework to address regression testing challenges in CI environment. The ultimate objective was to develop a TCP method that increases the fault detection rate of a test suite. Utilizing RL, this method can adjust its prioritization strategy in response to the inherent changes in CI environment. The results provided by the developed method demonstrate its feasibility and potential to enhance the efficiency and effectiveness of regression testing in the rapidly evolving software development landscape.

We implemented RL algorithms using stable-baselines3, selecting A2C, PPO, and DQN due to their support for discrete action spaces, which aligns with the requirements of the TCP problem. Among these, A2C demonstrated high performance, achieving an average APFD score of 0.78 across 963 CI cycles, indicating notable accuracy in prioritizing test cases based on test execution data. Notably, A2C was the most time-efficient during the training phase, requiring only 54,129 ms to learn from the data. This efficiency is particularly valuable in resource-constrained environments or in situations where rapid model updates are crucial, such as in the CI environment.

This research distinguishes itself in the field by leveraging high-quality and extensive data extracted from Nokia's CI repository. Access to such a rich, real-world CI data is a critical advantage, often a notable constraint in other research. In many instances, the full potential of developed methods in this domain remains unrealized due to limited data sources. The availability and use of this extensive data have been instrumental in advancing the research, enabling us to rigorously test and improve the developed method under realistic conditions, ensuring that the results are not only theoretically sound but also practically applicable in real-world CI environment. This direct alignment with real-world scenarios enhances the credibility and relevance of our findings and significantly contributes to the practical application of our research in the field.

The RL-based TCP method developed in this research marks a significant advancement in both academic research and practical applications. Academically, it enriches the existing body of knowledge by demonstrating how AI techniques can be innovatively applied to optimize software development processes. Practically, this method significantly increase the efficiency of regression testing, saving time and resources while effectively adapting to the ever-changing demands of CI environment. This adaptability is crucial for maintaining high software quality and reducing operational costs, thereby offering a tangible benefit to the fast-paced world of software development.

Despite these notable achievements, our study faced certain limitations. Primarily, due to time constraints and the extensive knowledge required to design distinct reward functions, we employed uniform reward functions across all algorithms. While practical, this approach may have limited the exploration and full potential of each individual algorithm. Furthermore, the complexity of hyperparameter tuning presented a challenge, requiring a delicate balance and in-depth understanding to optimize each algorithm's performance effectively. Future research in this field should ideally focus on developing algorithm-specific reward functions, tailored to the unique characteristics and potentials of each method. Additionally, more efficient and accessible hyperparameter tuning methods would greatly enhance the exploration and application of these advanced AI techniques in practical scenarios.

In conclusion, this thesis bridges a gap in applying RL to TCP in CI environment and paves the way for future advancements in software testing. The integration of AI and software testing will lead to the development of more intelligent, efficient, and adaptive testing processes in software development. I hope this work inspires further innovation, driving the evolution of software engineering practices to meet the demands of a technology-driven world.

9 REFERENCES

[1] Kandil P., Moussa S. & Badr N. (2015) A methodology for regression testing reduction and prioritization of agile releases. In: *2015 5th international conference on Information & Communication Technology and accessibility (ICTA)*, IEEE, pp. 1–6.

[2] Beck K., Beedle M., van Bennekum A. et al. (2001), Manifesto for agile software development. `https://agilemanifesto.org/`, Accessed: [insert date here].

[3] Fowler M., Highsmith J. et al. (2001) The agile manifesto. Software development 9, pp. 28–35.

[4] Shahin M., Babar M.A. & Zhu L. (2017) Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE access 5, pp. 3909–3943.

[5] Fowler M. & Foemmel M. (2006), Continuous integration.

[6] Yoo S. & Harman M. (2012) Regression testing minimization, selection and prioritization: a survey. Software testing, verification and reliability 22, pp. 67–120.

[7] Elbaum S., Rothermel G. & Penix J. (2014) Techniques for improving regression testing in continuous integration development environments. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 235–245.

[8] Labuschagne A., Inozemtseva L. & Holmes R. (2017) 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*, pp. 821–830.

[9] Engström E. & Runeson P. (2010) A qualitative survey of regression testing practices. In: *Product-Focused Software Process Improvement: 11th International Conference, PROFES 2010, Limerick, Ireland, June 21-23, 2010. Proceedings 11*, Springer, pp. 3–16.

[10] Rothermel G., Untch R.H., Chu C. & 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)*, IEEE, pp. 179–188.

[11] Haghighatkhah A., Mäntylä M., Oivo M. & Kuvaja P. (2018) Test prioritization in continuous integration environments. Journal of Systems and Software 146, pp. 80–98.

[12] Geerts G.L. (2011) A design science research methodology and its application to accounting information systems research. International journal of accounting Information Systems 12, pp. 142–151.

[13] DeLone W.H. & McLean E.R. (1992) Information systems success: The quest for the dependent variable. Information systems research 3, pp. 60–95.

[14] Hevner A.R., March S.T., Park J. & Ram S. (2004) Design science in information systems research. Management Information Systems Quarterly 28, pp. 6.

[15] Gregor S. & Hevner A.R. (2013) Positioning and presenting design science research for maximum impact. MIS quarterly pp. 337–355.

[16] Rittel H.W. & Webber M.M. (1973) Dilemmas in a general theory of planning. Policy sciences 4, pp. 155–169.

[17] Hevner A., Chatterjee S., Hevner A. & Chatterjee S. (2010) Design science research in information systems. Design research in information systems: theory and practice pp. 9–22.

[18] March S.T. & Smith G.F. (1995) Design and natural science research on information technology. Decision support systems 15, pp. 251–266.

[19] Davis F.D., Bagozzi R.P. & Warshaw P.R. (1989) User acceptance of computer technology: A comparison of two theoretical models. Management science 35, pp. 982–1003.

[20] Simon S.J., Grover V., Teng J.T. & Whitcomb K. (1996) The relationship of information system training methods and cognitive ability to end-user satisfaction, comprehension, and skill transfer: A longitudinal field study. Information systems research 7, pp. 466–490.

[21] Pfeffers K., Tuunanen T., Gengler C.E., Rossi M., Hui W., Virtanen V. & Bragge J. (2006) The design science research process: A model for producing and presenting information systems research. In: *Proceedings of the First International Conference on Design Science Research in Information Systems and Technology (DESRIST 2006), Claremont, CA, USA*, pp. 83–106.

[22] Thuan N.H., Drechsler A. & Antunes P. (2019) Construction of design science research questions. Communications of the Association for Information Systems 44, pp. 20.

[23] Duvall P.M., Matyas S. & Glover A. (2007) Continuous integration: improving software quality and reducing risk. Pearson Education.

[24] Chen L. (2015) Continuous delivery: Huge benefits, but challenges too. IEEE software 32, pp. 50–54.

[25] Humble J. & Farley D. (2010) Continuous delivery: Reliable software releases through build. Test, and deployment automation. Pearson Education 1.

[26] Minhas N.M., Petersen K., Börstler J. & Wnuk K. (2020) Regression testing for large-scale embedded software development–exploring the state of practice. Information and software technology 120, pp. 106254.

[27] Elbaum S., Rothermel G. & Penix J. (2014) Techniques for improving regression testing in continuous integration development environments. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 235–245.

[28] Do H. & Rothermel G. (2006) On the use of mutation faults in empirical assessments of test case prioritization techniques. IEEE Transactions on Software Engineering 32, pp. 733–752.

[29] Rothermel G., Harrold M.J., Von Ronne J. & Hong C. (2002) Empirical studies of test-suite reduction. Software Testing, Verification and Reliability 12, pp. 219–249.

[30] Rothermel G., Untch R., Chu C. & Harrold M. (2001) Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering 27, pp. 929–948.

[31] Cho Y., Kim J. & Lee E. (2016) History-based test case prioritization for failure information. In: *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, pp. 385–388.

[32] Di Nardo D., Alshahwan N., Briand L. & Labiche Y. (2015) Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system. Software Testing, Verification and Reliability 25, pp. 371–396.

[33] Leon D. & Podgurski A. (2003) A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In: *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, IEEE, pp. 442–453.

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

[35] Mukherjee R. & Patnaik K.S. (2021) A survey on different approaches for software test case prioritization. Journal of King Saud University-Computer and Information Sciences 33, pp. 1041–1054.

[36] Kavitha R., Kavitha V. & Kumar N.S. (2010) Requirement based test case prioritization. In: *2010 International Conference on Communication Control and Computing Technologies*, IEEE, pp. 826–829.

[37] Hettiarachchi C., Do H. & Choi B. (2016) Risk-based test case prioritization using a fuzzy expert system. Information and Software Technology 69, pp. 1–15.

[38] Panigrahi C.R. & Mall R. (2010) Model-based regression test case prioritization. ACM SIGSOFT Software Engineering Notes 35, pp. 1–7.

[39] Do H., Rothermel G. & Kinneer A. (2006) Prioritizing junit test cases: An empirical assessment and cost-benefits analysis. Empirical software engineering 11, pp. 33–70.

[40] Kim J.M. & Porter A. (2002) A history-based test prioritization technique for regression testing in resource constrained environments. In: *Proceedings of the 24th international conference on software engineering*, pp. 119–129.

[41] Elbaum S., Malishevsky A.G. & Rothermel G. (2002) Test case prioritization: A family of empirical studies. IEEE transactions on software engineering 28, pp. 159–182.

[42] Kaelbling L.P., Littman M.L. & Moore A.W. (1996) Reinforcement learning: A survey. Journal of artificial intelligence research 4, pp. 237–285.

[43] Li Y. (2017) Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274 .

[44] Sewak M. (2019) Deep reinforcement learning. Springer.

[45] Huang Q. (2020) Model-based or model-free, a review of approaches in reinforcement learning. In: *2020 International Conference on Computing and Data Science (CDS)*, IEEE, pp. 219–221.

[46] Zhang H. & Yu T. (2020) Taxonomy of reinforcement learning algorithms. Deep Reinforcement Learning: Fundamentals, Research and Applications pp. 125–133.

[47] Nachum O., Norouzi M., Xu K. & Schuurmans D. (2017) Bridging the gap between value and policy based reinforcement learning. Advances in neural information processing systems 30.

[48] Hausknecht M., Stone P. & Mc O.p. (2016) On-policy vs. off-policy updates for deep reinforcement learning. In: *Deep reinforcement learning: frontiers and challenges, IJCAI 2016 Workshop*, AAAI Press New York, NY, USA.

[49] Andrychowicz M., Raichuk A., Stańczyk P., Orsini M., Girgin S., Marinier R., Hussenot L., Geist M., Pietquin O., Michalski M. et al. (2020) What matters in on-policy reinforcement learning? a large-scale empirical study. arXiv preprint arXiv:2006.05990 .

[50] Precup D. (2000) Eligibility traces for off-policy policy evaluation. Computer Science Department Faculty Publication Series p. 80.

[51] Coggan M. (2004) Exploration and exploitation in reinforcement learning. Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University .

[52] Paavai Anand P. et al. (2021) A brief study of deep reinforcement learning with epsilon-greedy exploration. International Journal Of Computing and Digital System .

[53] Auer P. (2000) Using upper confidence bounds for online learning. In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*, IEEE, pp. 270–279.

[54] Ouyang Y., Gagrani M., Nayyar A. & Jain R. (2017) Learning unknown markov decision processes: A thompson sampling approach. Advances in neural information processing systems 30.

[55] Raffin A., Hill A., Gleave A., Kanervisto A., Ernestus M. & Dormann N. (2021) Stable-baselines3: Reliable reinforcement learning implementations. Journal of Machine Learning Research 22, pp. 1–8.

[56] Suleiman D., Alian M. & Hudaib A. (2017) A survey on prioritization regression testing test case. In: *2017 8th International Conference on Information Technology (ICIT)*, IEEE, pp. 854–862.

[57] Wong W.E., Horgan J.R., London S. & Agrawal H. (1997) A study of effective regression testing in practice. In: *PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering*, IEEE, pp. 264–274.

[58] Elbaum S., Malishevsky A.G. & Rothermel G. (2000) Prioritizing test cases for regression testing. In: *Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis*, pp. 102–112.

[59] Bagherzadeh M., Kahani N. & Briand L. (2022) Reinforcement learning for test case prioritization. IEEE Transactions on Software Engineering 48, pp. 2836–2856.

[60] Meçe E.K., Paci H. & Binjaku K. (2020) The application of machine learning in test case prioritization-a review. European Journal of Electrical Engineering and Computer Science 4.

[61] Durelli V.H., Durelli R.S., Borges S.S., Endo A.T., Eler M.M., Dias D.R. & Guimarães M.P. (2019) Machine learning applied to software testing: A systematic mapping study. IEEE Transactions on Reliability 68, pp. 1189–1212.

[62] Bertolino A., Guerriero A., Miranda B., Pietrantuono R. & 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*, pp. 1–12.

[63] Spieker H., Gotlieb A., Marijan D. & 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*, pp. 12–22.

[64] Nagabandi A., Clavera I., Liu S., Fearing R.S., Abbeel P., Levine S. & Finn C. (2018) Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. arXiv preprint arXiv:1803.11347 .

[65] Bagherzadeh M., Kahani N. & Briand L. (2021) Reinforcement learning for test case prioritization. IEEE Transactions on Software Engineering 48, pp. 2836–2856.

[66] Lousada J. & Ribeiro M. (2020) Reinforcement learning for test case prioritization. arXiv preprint arXiv:2012.11364 .

[67] Li G., Yang Y., Wu Z., Cao T., Liu Y. & Li Z. (2021) Weighted reward for reinforcement learning based test case prioritization in continuous integration testing. In: *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, IEEE, pp. 980–985.

[68] Shi T., Xiao L. & Wu K. (2020) Reinforcement learning based test case prioritization for enhancing the security of software. In: *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, IEEE, pp. 663–672.

[69] Wu Z., Yang Y., Li Z. & Zhao R. (2019) A time window based reinforcement learning reward for test case prioritization in continuous integration. In: *Proceedings of the 11th Asia-Pacific Symposium on Internetware*, pp. 1–6.

[70] Chen J., Lou Y., Zhang L., Zhou J., Wang X., Hao D. & Zhang L. (2018) Optimizing test prioritization via test distribution analysis. In: *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 656–667.

[71] Domingos P. (2012) A few useful things to know about machine learning. Communications of the ACM 55, pp. 78–87.

[72] Kelleher J.D., Mac Namee B. & D'arcy A. (2020) Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies. MIT press.

[73] Grondman I., Busoniu L., Lopes G.A. & Babuska R. (2012) A survey of actor-critic reinforcement learning: Standard and natural policy gradients. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42, pp. 1291–1307.

[74] Schulman J., Wolski F., Dhariwal P., Radford A. & Klimov O. (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 .

[75] Fan J., Wang Z., Xie Y. & Yang Z. (2020) A theoretical analysis of deep q-learning. In: *Learning for dynamics and control*, PMLR, pp. 486–489.

[76] Busjaeger B. & Xie T. (2016) Learning for test prioritization: an industrial case study. In: *Proceedings of the 2016 24th ACM SIGSOFT International symposium on foundations of software engineering*, pp. 975–980.

[77] Lima J.A.P. & Vergilio S.R. (2020) A multi-armed bandit approach for test case prioritization in continuous integration environments. IEEE Transactions on Software Engineering 48, pp. 453–465.

[78] Akiba T., Sano S., Yanase T., Ohta T. & Koyama M. (2019) Optuna: A next-generation hyperparameter optimization framework. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631.