



YEDITEPE UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

ANALYSIS OF
TEST CASE SELECTION TECHNIQUES FOR REGRESSION TESTING

A Thesis Submitted
by
Baturay Çağlasın

In Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

Supervisor
Prof. Dr. Mert Özkaya

Istanbul - 2025

ANALYSIS OF
TEST CASE SELECTION TECHNIQUES FOR REGRESSION TESTING

by
Baturay Çağlasın

Approved by:

Prof. Dr. Mert Ozkaya
(Yeditepe University)
(Thesis Supervisor)

.....

Dr. Deniz Akdur
(Co-Supervisor)

Prof. Dr. Akhan Akbulut
(Istanbul Kultur University)

.....

Assoc. Prof. Dr. Dionysis Goularas
(Yeditepe University)

.....

DATE OF APPROVAL:/...../20...

DECLARATION OF ORIGINALITY

I hereby declare that this thesis is my own work and that all information in this thesis has been obtained and presented following academic rules and ethical conduct. I have fully cited and referenced all material and results as required by these rules and conduct, and this thesis study does not contain any plagiarism. The necessary permissions have been obtained if any material used in the thesis requires copyright. No material from this thesis has been used to award another degree.

To the best of my knowledge and belief, it contains no material previously published or written by another person nor material accepted for the award of any other degree except where due acknowledgment has been made in the text.

I accept all kinds of legal liability that may arise in cases contrary to these situations.

Baturay Çağlasın

Signature

ABSTRACT

ANALYSIS OF

TEST CASE SELECTION TECHNIQUES FOR REGRESSION TESTING

Today, the development of software products to satisfy users' requirements has been expedited by the relentless competition that exists in the modern world. These products are in a state of intense competition with one another. Engineers who are developing these products are forced to compete against time as a consequence of this competition. The necessity of developing products quickly within a short period brings along software errors. Testing units bear the responsibility of identifying these software errors before users, as they must quickly execute their pre-prepared test cases. In this regard, testing selected test cases within a limited time has become a necessity.

In terms of the Average Cost per Execution (APXC) measure, the goal of this project is to make the regression test process as efficient as possible by making it easier for test engineers to do regression tests. It shows two approaches to testing the right test case in a short amount of time.

The first approach named clustered nature-inspired selection aims to conduct testing through selected tests within the test case set. It is based on clustering similar tests within the test case set using Jaccard, a similarity-based approach existing in the literature, and then selecting test cases within the clustered subsets using nature-inspired algorithm approaches.

The second approach named direct nature-inspired selection, on the other hand, conducts a selection process directly using nature-inspired algorithms without any grouping among the test cases. These approaches are designed to perform multi-layered selections, discussing which combination among the approaches will yield the optimal result.

ÖZET

REGRESYON TESTLERİ İÇİN TEST SENARYO SEÇİM TEKNİKLERİ ANALİZİ

Günümüz dünyasında artan amansız rekabet sonucu, geliştirilen yazılım ürünlerinin sayıları, kullanıcıların ihtiyaçları doğrultusunda hızla çoğalmaktadır. Bu ürünler, birbirleri arasında büyük bir rekabet içerisinde. Bu rekabet sonucu, ürünler üzerinde çalışan mühendisler zamanla da yarışmak durumundadırlar. Hızlı zamanda, hızlı ürün geliştirilme zorunluluğu, beraberinde yazılımsal hataları da beraberinde getirir. Bu yazılımsal hataları kullanıcılardan önce test etme görevi üstlenen test birimleri; önceden hazırladıkları test senaryolarını kısa bir süre içerisinde yapmak durumundadırlar. Bu bakımdan, kısıtlı zamanda seçili test senaryolarının test edilmesi bir ihtiyaç haline gelmektedir.

Bu proje, test birimlerinin regresyon testleri sürecinde yaşadıkları zorlukları azaltarak regresyon test süresindeki verimliliği Ortalama Yürütme Başına Maliyet (APXC) metriği kullanarak maksimize etmeyi amaçlamıştır ve kısıtlı zaman aralığında doğru testin test edilmesi için iki yaklaşım sunulmuştur.

Kümelenmiş doğadan ilham alan seçim adındaki ilk yaklaşım, test senaryo kümesindeki seçilmiş testler vasıtasıyla gerçekleştirmeyi hedefler. Test senaryo kümesindeki benzer testleri literatürde var olan benzerlik tabanlı yaklaşımlarından biri olan Jaccard ile gruplamak, ardından gruplanmış test alt kümelerindeki test senaryolarını doğadan ilham alınan algoritma yaklaşımları ile seçimler gerçekleştirmek temeline dayanır.

Doğrudan doğadan ilham alan seçim adındaki ikinci yaklaşım ise test senaryoları arasından herhangi bir gruplama gerçekleştirmeden direkt olarak doğadan ilham alınan algoritmalar ile bir seçim çalışması gerçekleştirir.

Bu yaklaşımlar ile, çok katmanlı seçimler gerçekleştirilerek; yaklaşımlar arasında hangi kombinasyonun optimum sonucu vereceği tartışılır.

ACKNOWLEDGEMENTS

It is with immense gratitude that I acknowledge the support and help of Professor Mert Ozkaya. Pursuing my thesis under his supervision has been an experience that broadens the mind and presents an unlimited source of learning.

I would like to thank my mother Sevil Çağlasın, my father Şenol Çağlasın, and my beloved wife Doğa Alataş Çağlasın for their endless love, making everything more beautiful. The study would not have been possible without their support and comprehension of my circumstances. I wish to convey my appreciation to my cats, Bidik, Karamel, and Shelby, for inspiring me to focus on nature-inspired algorithms. Witnessing their methodical, algorithmic processes prompted the notion that I could explore this topic. I recognized an absence in the research related to this topic, which further motivated me to do this study.

This thesis and my years of computer engineering study at Yeditepe University are the most significant gifts I have bestowed upon myself in my life. I desire for the study to improve the software quality assurance industry, of which I am honored to serve as a contributor.

TABLE OF CONTENTS

DECLARATION OF ORIGINALITY	iii
ABSTRACT	iv
ÖZET	v
ACKNOWLEDGEMENTS	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	x
LIST OF TABLES	xii
LIST OF ABBREVIATIONS	xiii
1. BACKGROUND	1
1.1. SOFTWARE ENGINEERING	1
1.1.1. Software Development Life Cycle	2
1.2. SOFTWARE TESTING AND TEST ENGINEERING	3
1.2.1. Software Quality	4
1.2.2. Software Test Methodologies	5
1.2.3. Software Test Techniques	6
1.2.4. Regression Testing and Software Environments	7
1.2.5. Software Test Case	9
1.3. SOFTWARE TEST CASE SELECTION AND TECHNIQUES	10
1.3.1. Software Test Case Selection	10
1.3.2. Similarity-Based Clustering Techniques	12
1.3.3. Nature-Inspired Selection Algorithms	13
2. INTRODUCTION	15
2.1. MOTIVATION	15
2.1.1. Understanding Testers' Perspectives - A Survey Study	16
2.1.2. Characteristics of Survey Participants	17
2.1.3. Analysis of Survey Results	17
2.1.4. Discussions	24
2.2. AIM AND OBJECTIVES	26
2.3. REPORT STRUCTURE	28

3.	RELATED WORK	29
3.1.	RELATED WORK OF TEST CASE PRIORITIZATION	29
3.2.	RELATED WORK OF TEST CASE SELECTION	33
3.3.	SUMMARY	35
4.	RESEARCH METHODOLOGY	37
4.1.	DETERMINING THE CLUSTERING METHODS	38
4.2.	DETERMINING THE TEST CASE SELECTION METHODS	39
4.3.	DETERMINING THE SUCCESS CRITERIA	40
4.4.	DETERMINING THE TEST CASES	41
4.4.1.	Data Collection	41
4.4.2.	Data Synthesis	43
4.4.3.	Data Analysis	44
5.	PROPOSED APPROACH	45
5.1.	CLUSTERING PROCESS BEFORE TCS	46
5.2.	THE DIRECT APPLICATION OF TCS	48
6.	EVALUATION	49
6.1.	CLUSTERED NATURE-INSPIRED SELECTION APPROACH	50
6.1.1.	Clustering Process	52
6.1.2.	Test Case Selection Process	56
6.2.	DIRECT NATURE-INSPIRED SELECTION	65
6.3.	SYSTEM PROTOTYPE FOR DIRECT NATURE-INSPIRED SELECTION	65
7.	RESULTS	69
7.1.	RESULTS FOR THE CLUSTERED NATURE-INSPIRED SELECTION	69
7.2.	RESULTS FOR THE DIRECT NATURE-INSPIRED SELECTION	73
7.2.1.	Stochastic to Deterministic Transition	74
7.2.2.	Repeated Test Case Situations	76
7.3.	RESULT BY SUCCESS CRITERIA	77
8.	DISCUSSION	83
8.1.	LESSONS LEARNED	84
8.2.	THREATS	85
9.	CONCLUSION	87
10.	FUTURE WORKS	88
	REFERENCES	89

APPENDIX A: TEST CASES.....	96
APPENDIX B: SIMILARITY MATRIX BY JACCARD	103
APPENDIX C: SIMILARITY DISTANCE MATRIX BY JACCARD	107
APPENDIX D: SURVEY QUESTIONS	111



LIST OF FIGURES

Figure 1.1. Some of the testing methodologies	6
Figure 1.2. Testing techniques [44]	7
Figure 1.3. An example software release flow	8
Figure 1.4. The idea of test case selection for the clustered nature-inspired selection..	11
Figure 2.1. Question one.....	18
Figure 2.2. Question two	18
Figure 2.3. Question three	19
Figure 2.4. Question four.....	20
Figure 2.5. Question five	21
Figure 2.6. Question six.....	21
Figure 2.7. Question seven	22
Figure 2.8. Question eight	23
Figure 2.9. Question nine.....	23
Figure 2.10. Question ten.....	24
Figure 4.1. Flow chart of the research methodology	38
Figure 4.2. Web-based e-commerce site replica.....	42
Figure 4.3. Web-based e-commerce site replica modules	43
Figure 4.4. Test cases synthesizing data	43
Figure 6.1. The similarity-based matrix based on Jaccard	52
Figure 6.2. The similarity-based distance matrix based on Jaccard	53

Figure 6.3. Heatmap of the Jaccard matrix	53
Figure 6.4. Heatmap of the Jaccard distance matrix	54
Figure 6.5. Bat algorithm Java code	57
Figure 6.6. Grey wolf algorithm Java code	59
Figure 6.7. Ant colony algorithm Java code	60
Figure 6.8. African buffalo algorithm Java code	62
Figure 6.9. Particle swarm algorithm Java code	63
Figure 6.10. Dashboard of OptiTest	66
Figure 6.11. Algorithm selection in OptiTest	67
Figure 6.12. Example output in OptiTest	68
Figure 7.1. Repeated test cases in African buffalo optimization	76
Figure 7.2. Comparison of the algorithms by durations	82

LIST OF TABLES

Table 1.1. Software test case structure	9
Table 3.1. TCP related works summarized.....	32
Table 3.2. TCS related works summarized.....	35
Table 6.1. Test cases for the approaches	49
Table 6.2. Test cases separated by keywords	51
Table 6.3. Cluster I bat algorithm parameters	56
Table 6.4. Cluster I grey wolf algorithm parameters.....	58
Table 6.5. Cluster I ant colony algorithm parameters.....	59
Table 6.6. Cluster I African buffalo algorithm parameters.....	61
Table 6.7. Cluster I particle swarm algorithm parameters.....	62
Table 6.8. Results of the other clusters	64
Table 7.1. Results of the clustered nature-inspired selection	70
Table 7.2. Results of the direct nature-inspired selection.....	73
Table 7.3. Evaluation of the success criteria	77
Table 7.4. Execution durations by the approaches and algorithms	79
Table 7.5. APXC evaluation of the clustered nature-inspired selection.....	80
Table 7.6. APXC evaluation of the direct nature-inspired selection	81

LIST OF ABBREVIATIONS

pbest	Personal best
gbest	Global best
u	Test case u
v	Test case v
I_u	Set of features rated by test case u
I_v	Set of features rated by test case v
$I_u \cap I_v$	The number of common features between the two test cases
$I_u \cup I_v$	The total number of features across both test cases
maxiter	The number of algorithm maximum iterations
Customer	Person who uses a software product
Test unit	Individuals who perform software testing
Test suite	Set of test cases
TC	Test case
SDLC	Software development lifecycle
TCS	Test case selection
TCP	Test case prioritization
TSR	Test suite reduction
TCM	Test case minimization
APXC	Average percentage of execution cost
APFD	Average percentage of fault detection
APCC	Average percentage of code coverage
APRC	Average percentage of requirements covered
IEEE	Institute of Electrical and Electronics Engineer
SBSE	Search-based software engineering
IoT	Internet of things
GA	Genetic algorithm
ABO	African buffalo optimization
GWO	Grey wolf optimization
ACO	Ant colony optimization

BA	Bat algorithm
FA	Firefly algorithm
PSO	Particle swarm optimization
CSA	Cuckoo search algorithm
DA	Dragonfly algorithm
BCO	Bee colony optimization
RQ	Research question
HACGA	Hybrid ant colony – genetic algorithm
SART	Similarity for regression testing
HBR2O	Hybrid battle royale-based remora optimization
BRO	Battle royale optimization
RO	Remora optimization
CI	Continuous integration
UI	User interface
AI	Artificial intelligence

1. BACKGROUND

The important role that software products play in our lives is undeniable in the modern world. Examples of how the concept of software has influenced the lives of modern individuals include the water we obtain from a vending machine, the POS devices we use to make payments at the grocery store, the office programs we open to complete our tasks, and the video games we play for enjoyment. Improving the quality of these software products is becoming increasingly critical as the number of users increases. This expectation is reasonable, as users anticipate that the software they depend on will operate flawlessly and satisfy their requirements.

In this section of the study, the significance of the software engineering concept is examined from a comprehensive standpoint. A comprehensive analysis of the primary focus of this work, software test cases and their selections has been conducted by examining the steps a software product undergoes to remain operational and expounding on the importance of software testing.

1.1. SOFTWARE ENGINEERING

In 1968, Bauer declared that software engineering is what we require, and Bauer organized the Garmisch conference that established our discipline [1]. Software engineering is essential for the state, society, national and international industry, and institutions. Software is critical to the operation of the modern world. The majority of electrical items are equipped with computers and control software, and computer-based systems manage national infrastructures and services. A software-controlled mobile phone is utilized by more than 75% of the global population [2]. It is imperative to comprehend the historical development and present state of software engineering to predict its future.

The importance of a refined software engineering process in the future is becoming increasingly critical as a result of the emergence of various technological advancements, including Industry 4.0 and the Internet of Things (IoT). Furthermore, software engineering is a well-established field that has its own set of application structures, procedures, and methods. Today, software engineering makes use of these well-established approaches and tools in a range of disciplines, including information processing applications like databases,

commerce, and e-commerce. Software engineering serves a critical role in real-time industries, facilitating the automation of energy control in factories, transportation systems, and manufacturing processes. Software engineering is no longer just guided by rules that were made for professional software developers. More and more companies are building systems that are connected to the Internet. The primary goal of these expanding industries is to transition from the fundamental coding necessary to guarantee the proper operation of machines to the assurance that the software developed achieves the highest quality standards and fulfills its intended purpose [3].

1.1.1. Software Development Life Cycle

According to Rastogi, the Software Development Life Cycle (SDLC) is a clear and organized way to make a stable, high-quality software system [4]. There are four primary phases in the software development life cycle analyzing requirements, designing, developing, maintaining, and testing. [5]. Additionally, there are sub-phases, such as the feasibility study, software deployment, maintenance, and improvement [6].

The requirement analysis phase focuses on the project as an entire unit, figuring out what it needs and laying the basis for the product. The design process, on the other hand, is all about coming up with ideas for the new product and planning its structure, features, and safety measures. After this, the coding or development phase turns the system design into source code that can be run, making the first version of the product. Lastly, the testing phase makes sure the software product is complete by making sure it works and meets certain standards [6].

It is a structured set of phases and tasks that occur during the software development process [7]. In the field of software project management, picking the best SDLC model is a very important choice. The task of managing software projects is complicated and has many parts [8].

In project management, it is important to have strong oversight over responsibilities and individuals in order to reach certain goals, like reducing costs and time or increasing profit and performance. Getting software finished faster is becoming an important requirement for getting software on the market on time, which is necessary to gain market share and make more money [9]. A search problem or an optimization problem is one that tries to find the

best way to reduce the time it takes to finish a task. Search-Based Software Engineering (SBSE) methods can be used to deal with these kinds of problems [10].

This study was mostly about the testing phase of the SDLC. Individuals should refrain from thinking of the testing phase as the final phase in the SDLC or as a phase that can be skipped. The following sections discuss how essential the testing phase is. Gelperin and Hetzel declare that the SDLC models put testing at the beginning of a project instead of at the end of it. Because of this change, individuals currently think that testing is not just a step that happens after the software execution; it also affects the software requirements and designs at the same time. Individuals think of testing as having an important effect on development from the beginning of a project [11].

The point of this study had not been to find the best SDLC model to solve an optimization problem; it also had not been intended to investigate every phase of the SDLC individually to find approaches to make things improved. The point is to improve the testing phase of the SDLC.

1.2. SOFTWARE TESTING AND TEST ENGINEERING

Software testing has become more important as the number of individuals using software products has grown and changed. The goal of quickly created software products is to stay one step ahead of the developers of similar products. However, if users use a software product that is not tested, they may encounter issues ranging from minor to critical. These kinds of issues in software are known as bugs. Sometimes these issues or errors only manifest themselves in the visual layer, not in the functionality; however, critical bugs, especially on a functionality basis, can significantly break the application.

Individuals are capable of errors. The purpose of the testing is to find bugs and see if the software program meets the requirements from both a technical and customer desire or business point of view. Testing is an approach to figuring out how accurate the software is, which means how long it will work without issues in certain conditions. In the long run, this approach decreases the cost of development while improving the efficiency of it. Delivering a high-quality result is a key component of software testing that builds confidence among users [28].

There are states in the literature that emphasize the significance of test software applications. Bertolino states that testing is a key component of software creation [29]. More than that, Burnstein states that software testing essentially assures users that they require that the program or system does what it is indicated to do based on their instructions [30].

It is essential that software testing be a part of the SDLC. As part of the SDLC, software testing is the process of ensuring the product works accurately before getting approval for it. Tests of software are essential for making sure individuals can use a properly developed software product. During this phase, the software product is inspected to ensure it is delivered to the users efficiently and without bugs. Beginning in the early 1970s, there was a noticeable rise in the level of skill associated with software testing. This marked the recognition of software test engineering as a growing specialized field [11].

1.2.1. Software Quality

Many people have different thoughts on what excellent software is, but most agree that it meets all of the customer's needs. Crosby states individuals must define quality by meeting standards to get it right [12]. Crosby also emphasizes the importance of clearly defining quality to enable effective measurement and management in business settings. Standards such as "that is close enough" are not sufficient; "zero defects" are. Crosby has pushed the idea that a company should try to make no mistakes since that is both possible and good. Crosby demonstrates that defining quality entails transforming people's desires into quantifiable characteristics, enabling manufacturers to produce goods that satisfy their requirements at a cost they can manage [13]. Using the phrase "meeting customer needs" to explain the author's idea of quality, Ishikawa makes a strong link between quality and how users' needs are always changing. When Juran asserts that quality can mean different things, Juran contradicts the opinions of other quality experts. The author argues against using the term quality solely to meet customer needs or specifications, highlighting the challenge of meeting such high standards. The author says that quality is "fitness for use," which means that it works well and meets the needs [14]. For the most part, excellent software meets users' desires at the right time, the right way, and without any errors, or bugs.

1.2.2. Software Test Methodologies

No matter how important it is, finding errors before software products' users notice them is very significant. These steps, called software testing, find errors in both the code layer and the user interface of a newly created product so that problems do not happen [11]. Anwar and Kar state that testing software cuts down on mistakes and the cost of making applications [15]. This is the methodical process of checking to see if the real results match the expected results [16].

Along the way, people have learned how to make testing more efficient and cheaper [11]. Create test cases as the first step in starting the testing process. Black box testing, white box testing, and gray box testing are some of the most common types of testing. These methods give different ideas about how to do tests.

White box testing is highly impactful due to its ability to assess both the functionality and internal structure of the software application. Proficiency in programming is essential for crafting test cases tailored for white box testing. Decision coverage, data flow testing, and control flow testing are some white box testing techniques.

In black box testing, the operation of a software product is closely tested without going into its deep dive structure. Checking all the product's features to make sure they meet customer needs. Black box testing has many forms, such as all-pairs testing, equality splitting, use case testing, decision table testing, and mistake guessing.

Gray box testing combines the best parts of both white box testing and black box testing. Given that test engineers know how programs work on the inside, they can do a more thorough check of how they work on the outside [17,18]. Figure 1.1 shows some of the testing methodologies under the testing types as white box, black box, and gray box.

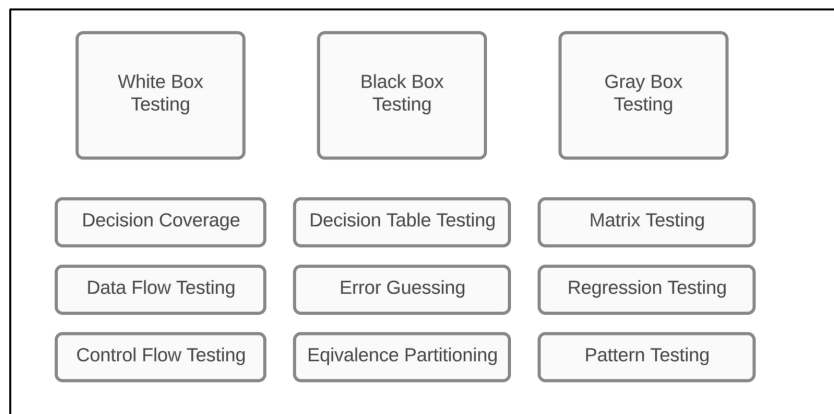


Figure 1.1. Some of the testing methodologies

1.2.3. Software Test Techniques

In the literature, there are many types of software test techniques. Each technique outlines specific methods for executing the test cases. We generally categorize these software test techniques into two categories: functional and non-functional testing. While functional tests concentrate on specific quality types and aim to answer "how" questions like performance, security, compatibility, and usability, functional tests concentrate on the behavior and functionality of the system, focusing on "what" the system should accomplish.

Unit, integration, system, and acceptance test techniques can be considered as functional testing. Security, performance, and usability test techniques can be considered non-functional testing [44].

Under the system testing, a testing technique called regression is the study's focused technique. In the Section 1.2.4, more detail will be given on what regression testing is and why it is being used. Figure 1.2 describes the testing techniques as functional and non-functional.

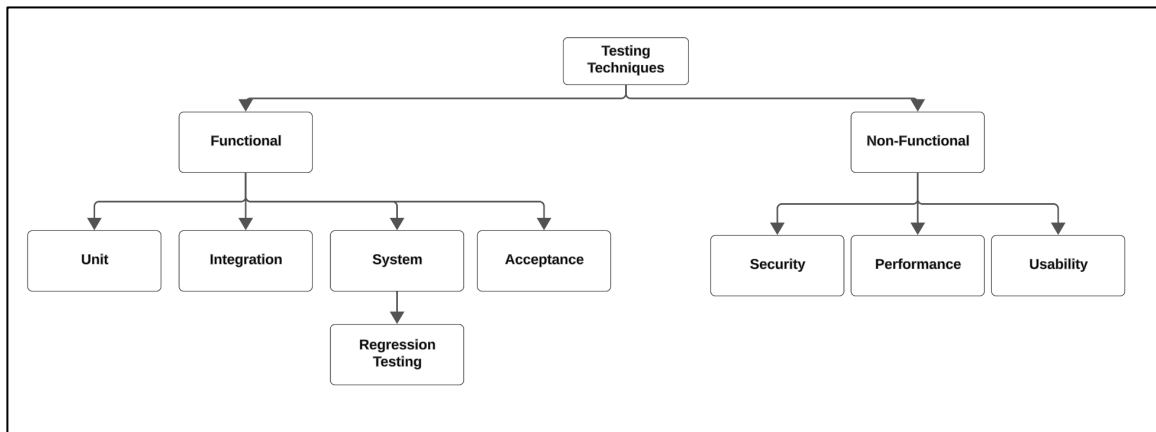


Figure 1.2. Testing techniques [44]

1.2.4. Regression Testing and Software Environments

Regression testing plays an important role in the testing process, ensuring that the software remains free of bugs following any new developments (features) to the requirements or software product [22]. These developments, features, or changes in a software product never end within the product's software development life cycle. The test process should reply to itself over and over again after the changes. This retesting process after new developments, features or any changes in the product is called regression testing [27].

The software product quality is improved by regression testing, and the test team and the other teams can get an output of effects after these new developments and changes. The test team can give verification of bugs before they are discovered by customers or users of that product [55].

Regression tests can be conducted at a predetermined time or scheduled for each version of a software release. Usually, regression tests are conducted after deployments to software environments such as development, staging, and production. From a tester's perspective, software environments can be summarized as follows.

The test environment (sometimes referred to as the development environment) is where changes to the software are initially deployed. It is typically not accessible to end users, or customers. In this environment, software developers are expected to deploy their newly developed code to test engineers (and other individuals such as business analysts, product owners, etc.) for testing.

The staging environment serves as the second and intermediate stage. Before moving to production (live environment), adaptations of developments to the product are tested once again. After thorough assurance and necessary steps, the deployment of developments for user access is approved.

The production environment is the live environment where end users interact with the application, demanding careful handling. Errors occurring in this environment are typically rectified through a hotfix. However, an abundance of such hotfixes is undesirable, as last-minute fixes do not guarantee that they will not disrupt other areas of the application.

In all these environment flows, it is expected that the test engineers execute their regression test cases and take action. While the numbers or names of these software environments may vary from product to product or company to company, the overview provided is generally applicable. Figure 1.3 represents an example flow chart of software environments under considering the scope of regression testing.

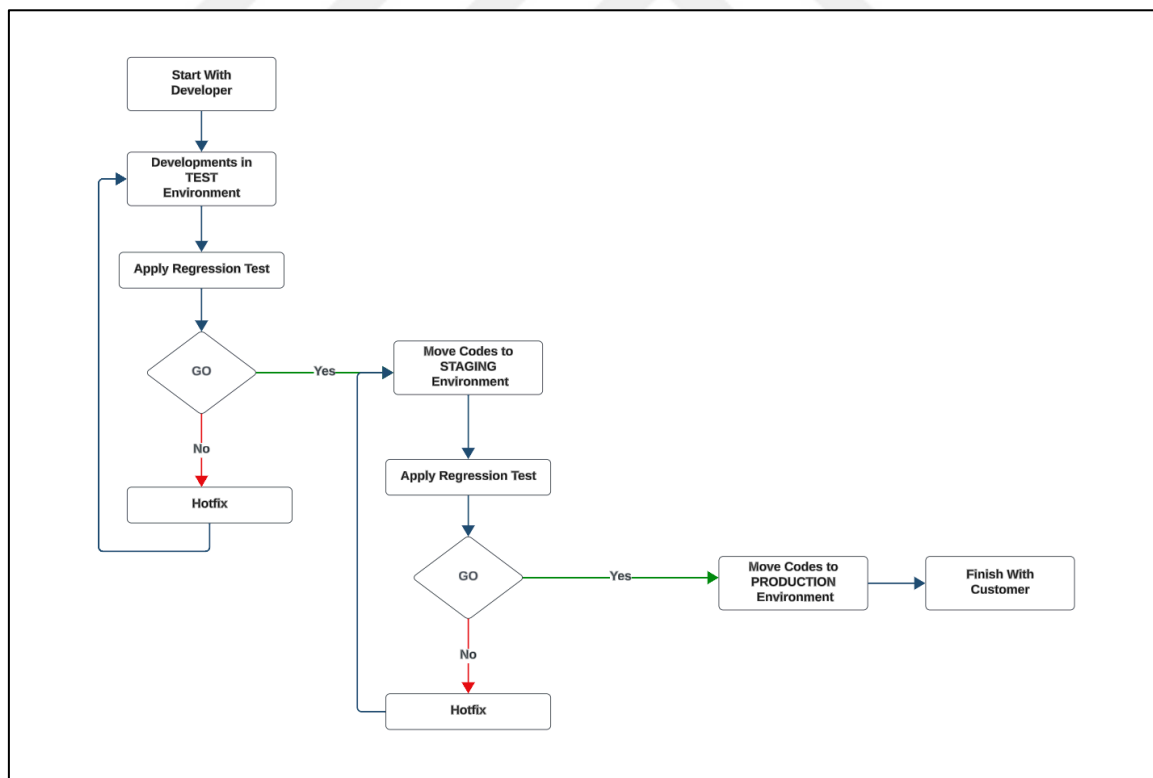


Figure 1.3. An example software release flow

The processes start with development. These developments include new features and bug fixes from previous versions. The packages containing this code are first sent to a test environment. In the test environment, tests of these new features and bug fixes are performed, and a regression test is executed. During the execution of the regression test, it is ensured that the new features do not break the existing ones. These tests, conducted by test engineers, may either receive a go-based on the situation or errors may be identified and resolved with fixes. The same processes are carried out for the staging environment. If no issues are detected in the staging environment, the new package is deployed to the production environment and made available to customers.

1.2.5. Software Test Case

Testing tasks include a series of activities, which are generating test cases, constructing test suites, executing the suites, analyzing the results, and preparing reports summarizing the testing process [19]. Software testing involves the utilization of established or newly created test cases to verify the functionality of software during its development phase and to validate its performance before deployment [20].

Kaner states that IEEE Standard 610 describes a test case as a set of test inputs, processing conditions, and expected results that were created for a specific purpose, like to run a certain program path or make sure that a certain requirement is met [21].

Based on this information, a test case can be specified as described in Table 1.1:

Table 1.1. Software test case structure

Test Case ID	Test Description	Pre-Condition	Test Steps	Test Data	Expected Results	Actual Results
ID1	Desc.	pre-condition if any	1. Step_1 2. Step_2	Data	Exp.Res.	Act.Res.
ID2	Desc.	pre-condition if any	1. Step_1 2. Step_2	Data	Exp.Res.	Act.Res.

The cumulative name of the test cases to be tested at regression time is called the regression test suite. Within this suite, changes and new features in each sprint (a methodology that its

period is generally carried out in 10-day periods) should be part of regression testing. This is necessary to ensure that new changes do not break existing functionality.

At the same time, not only the changes made in that sprint should be tested, but also all existing functionality of the software. This is important to maintain the overall stability of the software.

1.3. SOFTWARE TEST CASE SELECTION AND TECHNIQUES

The main topic of this study is how to select test cases for regression testing based on certain approaches. During the regression period, time is limited, so it is important to select the test cases. The ideas behind the suggested approaches are "clustering with similarity-based first and then applying nature-inspired selection algorithms" and "applying nature-inspired selection algorithms directly without clustering the test cases."

1.3.1. Software Test Case Selection

Software Test Case Selection (TCS) is a process of selecting software test cases that aims to choose the minimal number of test cases from test suites to achieve comprehensive testing with the lowest possible cost [45]. However, it has some challenges inside of it. Souza et al. say that TCS is a hard problem because there are so many different combinations to think about. When provided with a test suite, the goal of TCS is to identify a relevant subset of test cases that meet the chosen sufficient criteria. This allows for the removal of test cases that do not contribute to improving the coverage of the reduced test suite. The effectiveness of the selection relies on how well it covers the adopted competence criteria [46].

Test Suit Minimization (TCM) and Test Case Prioritization (TCP) are the other approaches that focus on regression testing processes [54]. Furthermore, there are many test case selection and prioritization techniques. In general terms, these approaches are cost-based, history-based, similarity-based, nature-inspired-based, risk-based, coverage-based, requirement-based, and search-based. Each approach comes with its own distinct algorithms.

In this study, combinations between "similarity-based clustering techniques" and "nature-based algorithms" are aimed to be created to reach the optimum combination for TCS. Figure 1.4 represents the concept of this study for TCS, which is analyzed under clustered nature-

inspired selection. According to this approach, the test cases will be clustered based on their similarities, and then a selection process will be conducted on these clustered test cases using the nature-inspired algorithms included in the study. More information can be found in the Section 4.

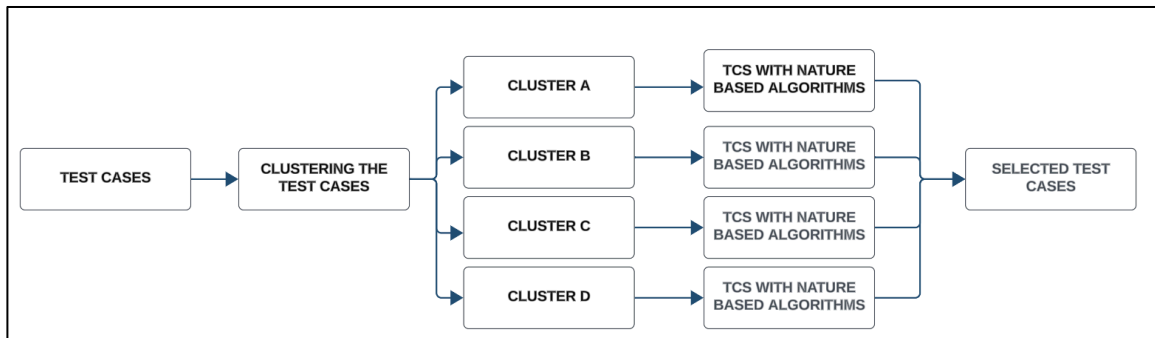


Figure 1.4. The idea of test case selection for the clustered nature-inspired selection

TCS is important in many ways, but the most important thing is that it can save time and development costs by picking the most important test cases.

It was shown by Chittimalli and Harrold that regression testing costs more, making up more than 33% of all the costs that come with making software [31]. As engineers, the testing costs need to be reduced by categorizing or ranking these cases.

Several methods in the literature aim to categorize test cases. TCS is only one of them, and there are also methods such as TSR and TSP as mentioned.

The duration allocated for testing is a crucial factor that is contingent upon the scope and complexity of the software. It has been observed that testing consumes more than 50% of the total software development time as it encompasses various activities [20]. Also, in software maintenance, research found that testing implementation costs could constitute as much as 80% of total maintenance expenses [34].

Due to the time constraints of regression testing, executing all test cases every time changes are made to the code becomes impractical. This underscores the importance of test case categorization in such cases [22].

Furthermore, choosing which test cases to run by TCS is crucial for saving time on testing while still ensuring the software's quality remains intact [59] Test engineers can focus on the

most critical areas during regression testing for that deployment. It allows better utilization of testing resources by eliminating unnecessary tests. If the number of test cases is reduced, updating and executing time for the test cases will be reduced.

Test engineers can target key functionalities and potential risk areas on products. Selecting test cases can cover the most critical parts of products. Test engineers can find their critical bugs quicker, and they can increase the efficiency of their regression time period.

1.3.2. Similarity-Based Clustering Techniques

Similarity-based algorithms are one of the focused algorithms that are developing in the literature. Jaccard, Cosine, Pearson Correlation Coefficient Euclidean, and their hybrid algorithms such as multi-level collaborative filtering and TMJ are a couple of examples of similarity-based techniques in the literature [47].

Jaccard similarity is a technique for comparing members of two sets to determine which members are common and which are unique [48]. There is an index in the Jaccard formula that ranges from zero, indicating no similarity, to one, indicating perfect similarity [49].

There is another method that is called cosine similarity. It uses cosine measurement between two rating vectors' angles [50].

Pearson Correlation Coefficient and its weighted version are the other methods. The coefficients focus on a threshold of co-rated items, and coefficients use these items to enhance the accuracy under specific conditions [51].

The proximity-impact-popularity approach evaluates aspects such as the proximity of users, the influence of their choices, and the popularity of items. However, its effectiveness may be limited for new users or items because of how it is structured [52].

At the higher dimensionality on the non-central t-distributions, Euclidean distances are very useful [53]. Euclidean distances measure the straight-line distance between two points in the space. Euclidean distances calculations are based on the Pythagorean theorem. Shortly, squares of the differences between corresponding coordinates of the points will be summed, and then it should take the square root of that sum.

1.3.3. Nature-Inspired Selection Algorithms

Due to the restricted time allocated for regression testing, the process becomes challenging in determining which test cases to select effectively.

Genetic Algorithms (GA) is one of them. The algorithms aim for natural selection and genetics. Chromosomes are represented by large randomly generated population members, and the chromosomes are created by genes. These genes can be assumed as binary digits in selected algorithms. Also, mutations and crossovers are the key terms that GAs focus on. The parents, which are selected as two chromosomes by the crossover operator, produce a new chromosome that is offspring. After fitness function usage, the selection of parents can be done. If the crossover algorithm is applied iteratively, new genes can be generated in the population. Furthermore, the length of chromosomes can create a rate of mutations that can happen to genes [55].

African Buffalo Optimization (ABO) is the other selection technique for test cases. The algorithm inspires African buffalos' migration behavior. However, there should be memory and alliance processes that track their movements after their generated sounds; alert and safe. Also, an intelligence process is generated due to any conflicts in their movements [56].

Grey Wolf Optimization (GWO) mimics the social hierarchy of wolves by arithmetical methods. The algorithm categorizes the wolves by hierarchical levels such as alpha, beta, delta, and omega. The best approach can be called alpha and respectively. While ABO focuses on migration behaviors, GWO focuses on hunting behaviors. The main idea of the algorithm is to consider a first location to prey and update the positions to reach their hunt [37].

Ant Colony Optimization (ACO) is a nature-inspired optimization technique that aims to investigate searching for food by ants' directions with their pheromones. In the optimization technique, there is no memory or hierarchy process like the other methods; rather that, it focuses on the decision points by the pheromones that ants produce. However, there are some other factors, such as the distance factor, that give negativity to this decision on food tracking [35].

Like ACO, the Bat Algorithm (BA) also aims to investigate searching for food by a bat's pulse ability. First, the bat population should be identified, and then the pulse frequency

should be initialized. Because BA simulates the pulse and loudness ability of bats, where bats emit sounds and listen to the echoes while they are searching for food. Secondly, this frequency and the positions should be updated in each iteration for potential solutions. The algorithm's certain parameters are changing by random walks to find the best local search solution. If a bat finds a better solution for reaching foods, this pulse rate increases and loudness decreases. If a selected random number is less than the loudness, the new solution should be selected [57].

Particle Swarm Optimization (PSO) is generated for complex optimization problems to solve them. PSO focuses on the movements of birds and fish swarms. A random position and velocities should be initialized for the population of swarms and then updated pbest (personal best) and gbest (global best) positions by comparing all particles' personal bests. After these updates, the velocity should get an update based on pbest and gbest positions by random factors in each iteration. The algorithm leverages the concept of social interaction to guide swarms by sharing information among them [35,58].

Dragonfly Algorithm (DA) is generated from social interactions between dragonflies to seek their foods and avoiding hunters through exploration and exploitation. An initial population is produced randomly, and the parameters indicate the current position, food, and hunting locations. The algorithm updates swarming weights, the separation between dragonflies, and their alignment. Another calculation is calculated by the Euclidian distance formula inside the algorithm for population update [38].

In all nature-inspired algorithms, there is a location update. While some methods aim to survive against enemies, some methods work to meet personal needs such as food. There is a hierarchy in GWO, but there is no hierarchy in some algorithms, but only a skill. (echo in BA or memory in ABO) However, despite all these differences, from a general perspective, these algorithms are designed based on the periodic movements of animals in nature and have found a place in the literature.

2. INTRODUCTION

Alike shown in Section 1, the testing phase in the SDLC is equally essential as all other phases. This phase is not allocated infinite time, particularly following regression periods. It is essential for test engineers engaged in software testing to conduct tests efficiently and deliver prompt feedback during time intervals. If there are any bugs in the package, test engineers must report them to developers, who should fix them as soon as possible. Business units are also expected to review these results and conduct their own assessments. At this point, test engineers face a substantial workload. This requires the greatest proficiency in time management and attentive attention. The best use of this constrained time is to systematically remove useless test cases, therefore exemplifying the adage of testing the appropriate tests in the shortest time. Test engineers can efficiently finish their evaluations and give input to the appropriate departments if the proper tests are carried out on time.

This section defines the author's motivation prior to starting this study. It includes a survey that supports the study's motivation and analyzes the results obtained from it. This section examines the study as a whole, explaining the main purpose and the objectives aimed at achieving this goal. It argues that the methods and approaches mentioned in Section 1 should be employed to reach the study's objectives. Furthermore, it outlines the report structure, guiding the reader through the upcoming sections.

2.1. MOTIVATION

As the main and most important job of test engineers, they need to find bugs in software products before users do and make sure that they highlight these problems to the right teams using clear language and the right reasons so that they can be fixed. The probability of discovering bugs is substantially reduced in software projects that are tested in a hurry or without proper planning, such as prioritizing or selecting test cases. Alternatively, users may discover these issues. Frequently, test engineers encounter difficulties in maintaining clear communication and completing duties thoroughly when they are under time constraints, such as regression testing.

The costs of regression testing can escalate rapidly if the most critical test cases are not selected. The productivity of other team members, such as developers or product managers,

can also be disrupted by a lack of time for regression testing, which can ultimately affect the end users.

Test engineers may use their own opinions to decide which test cases to execute so that they can make the best use of their limited time. But this can lead to very important mistakes. On the other hand, results may be more reliable when individuals use a structured method that is based on well-known techniques from the books. The resolution of these difficulties is expected to improve the overall quality of the software product. Thus, the company's reputation may be enhanced, perhaps yielding a competitive advantage through elevated customer satisfaction. This ultimately enhances the company's sustainability and success.

A survey was implemented to ascertain the challenges that evaluators encounter. Initially, participants were questioned regarding their primary challenges encountered during the assessment phase. This survey was instrumental in the identification of critical issues and provided the foundation for this investigation.

2.1.1. Understanding Testers' Perspectives - A Survey Study

The survey was conducted to identify the difficulties experienced by test engineers. Regardless of experience and field, the test engineers are involved in any software product that they are currently working on. The research questions were implemented before preparing the survey questions.

RQ I: How efficiently is time management carried out by teams during the testing phase?

RQ II: How many test cases are tested in the limited time during the regression?

RQ III: What is the scope of the test cases that are tested in that limited time?

RQ IV: What is the total regression testing time?

The exploratory research determined the survey study questions that provide insight into the answers to these questions. There are 20 questions in the survey, and the questions cover challenges in software testing and development. The questions provide insights into various topics such as time management, test coverage, complexity, risk identification, resource allocation, and documentation maintenance. Also, the participants are expected to answer

questions ranked from one to five. The related survey questions and their target topics are described in Appendix D.

2.1.2. Characteristics of Survey Participants

In this survey study, a tool called Google Survey was used, and participants participated via e-mail or link. The survey study was conducted with the participation of over 20 test engineers from the software industry, working in different private companies within the borders of Turkiye and on various projects, with a diverse range of experience levels from junior to senior. Junior participants were required to have at least one year of experience. All participants were expected to have experienced or be aware of the testing process in the SDLC. In this regard, after obtaining the necessary permissions, the relevant survey was conducted with test engineers who played a role in the manual and automation testing processes, and it was underlined that participation was completely optional.

The questions were prepared to focus entirely on the problems in software testing processes. In this regard, another sector, field, or similar person who is not familiar with these processes cannot answer the questions. To answer the survey questions, it is envisaged that people who have gained a certain experience in the testing processes will be answered.

2.1.3. Analysis of Survey Results

As a result of the following answers given to the question of how extensive the coverage of test cases during the limited testing time in the survey, it was observed that the majority of people provided moderate coverage. This analysis shows that tests performed at limited times, such as regression times, are not sufficient. It is possible to improve the coverage by doing a TCS study.

2.How extensive is the coverage of test scenarios during the limited testing time?
(Especially for regression test cases.)

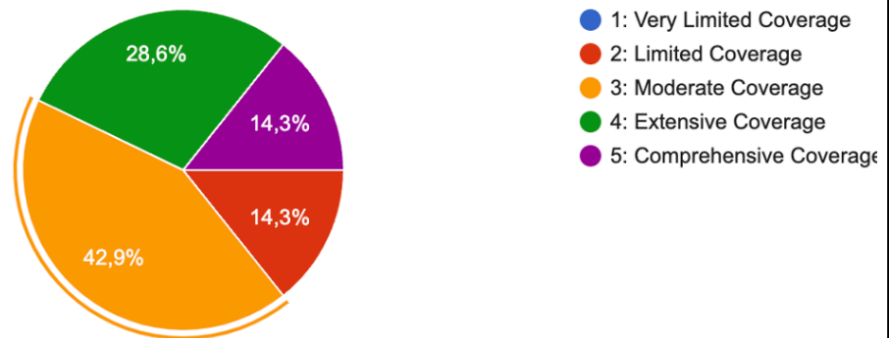


Figure 2.1. Question one

For the question of how confident are you in your team's ability to identify high-risk areas for testing in the survey, a high majority indicates that the team and the individual himself/herself know the risk areas in a product.

4.How confident are you in your team's ability to identify high-risk areas for testing?
Are they capable of identifying high-risk areas?

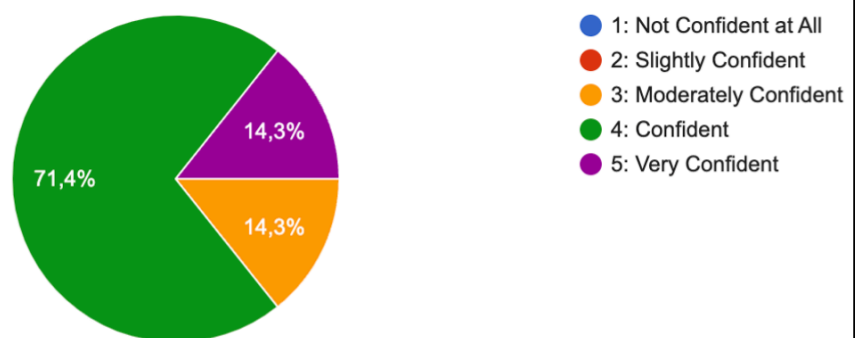


Figure 2.2. Question two

Knowing the risk areas in a product is important for a software test engineer. The person who selects the product tested based on experience may be beneficial for a limited time. But on the contrary, positive answers to this question carry a risk. Test selection studies that are based on experience and carried out completely away from the studies in the literature may

not always yield positive results. Risk areas in the product may have changed. There may be overlooked situations in a test selection study that are far from methodologies. This TCS study needs to be carried out with appropriate methodology and techniques.

The question of how time-consuming is regression testing in your software development life cycle is also referred to as slightly as a percentage.

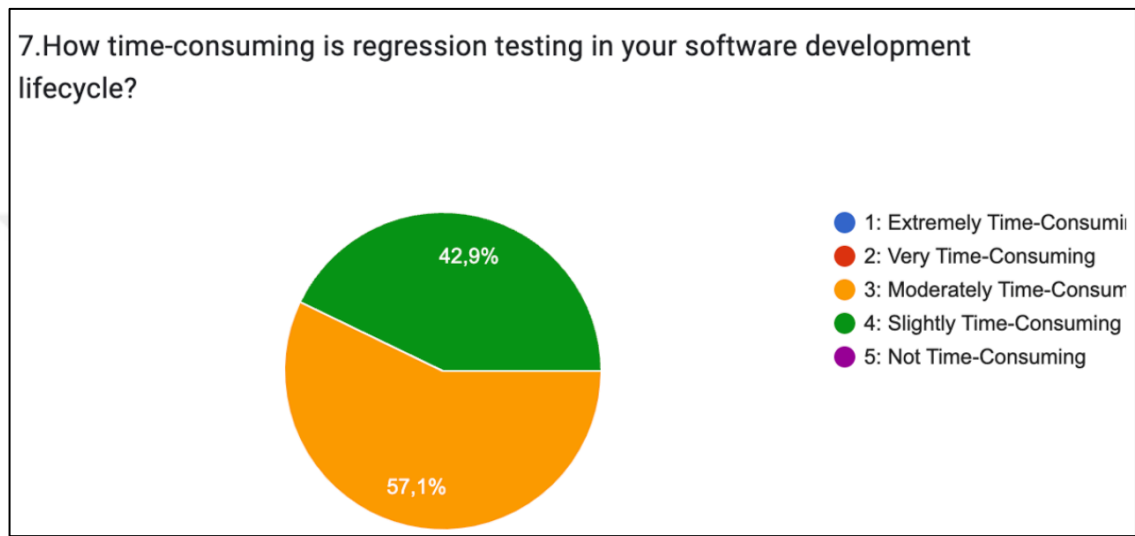


Figure 2.3. Question three

For the question of how challenging is it to manage dependencies between test cases during testing the answers are given. While developing a software product, many different teams work together, and there is always a commitment between these teams. Inter-team dependency poses a challenge not only for the testing team but also for all other teams as well.

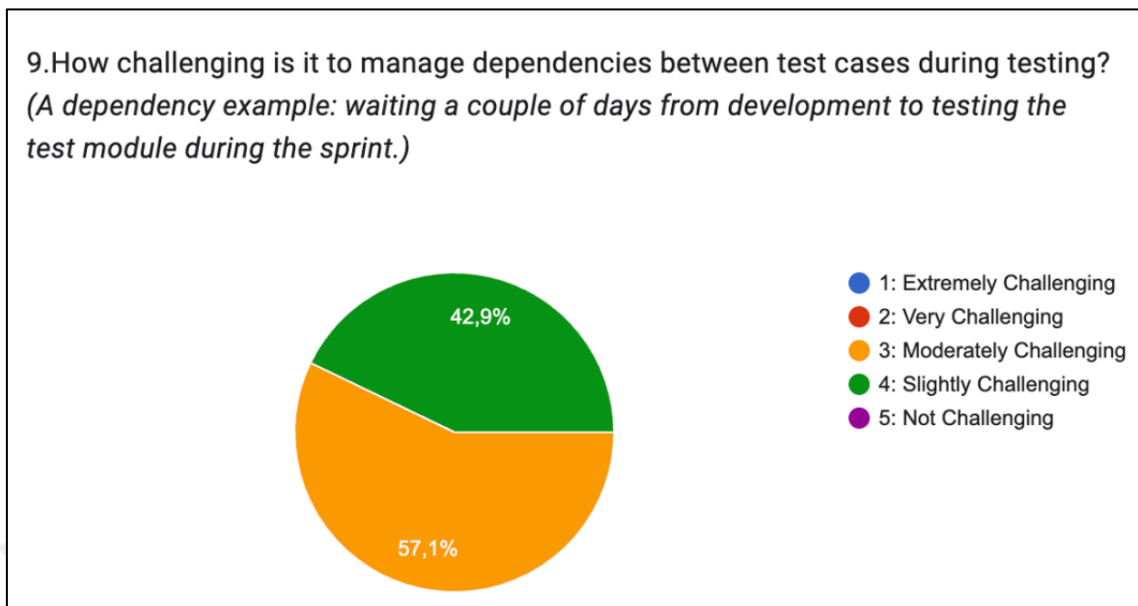


Figure 2.4. Question four

A software product is expected to constantly change and evolve within the scope of SDLC to survive among competing products. These changes and developments often require taking care of the test cases written in test teams. The objective was to ascertain the extent of the challenge associated with maintaining and updating test cases as the software develops. The query was acknowledged as very challenging by a significant number of respondents. Taking care of written test cases; considering that the time allocated to writing new test cases is limited, it is quite challenging, and the answers given support this. Time is limited, as for other teams, and the test engineers must both write new test cases and maintain existing tests in this limited time.

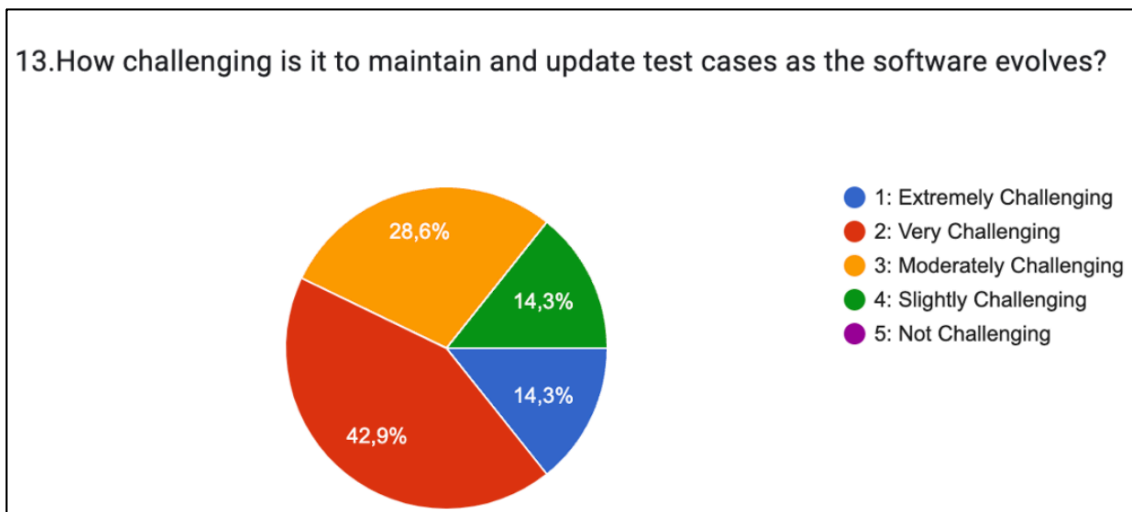


Figure 2.5. Question five

In addition to the question analyzed, it has been concluded that outdated and idle test cases are frequently found in the relevant test suite. There are also idle cases in the test cases that take place in a limited time, and starting to test these cases without selection or maintenance will create a lost-effort situation for the test team.

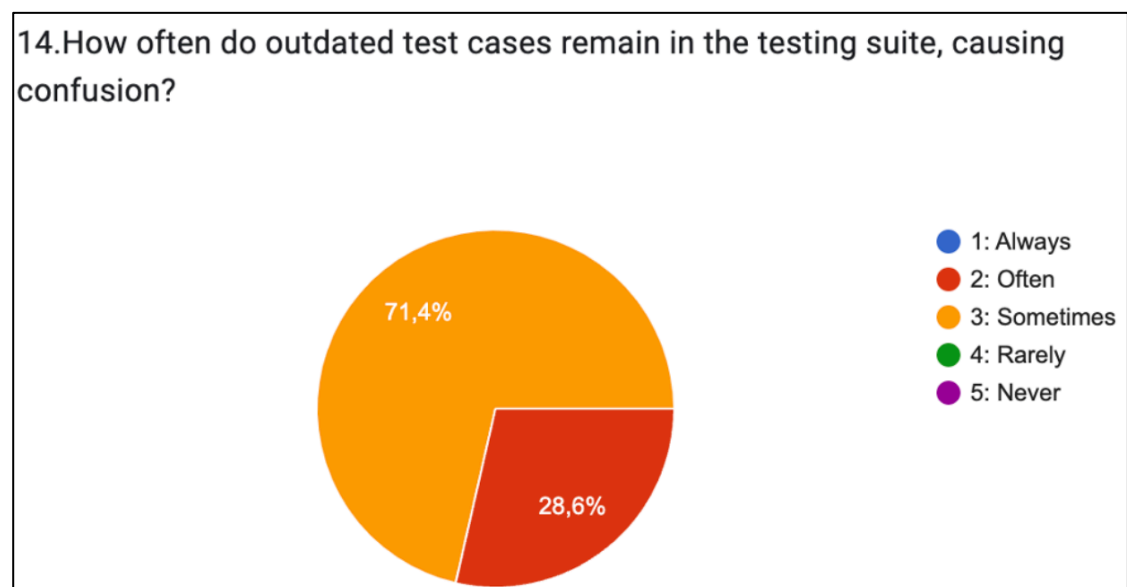


Figure 2.6. Question six

It is a healthy process for a test engineer to know the needs of the product from the relevant teams and write test cases according to these needs. However, by analyzing these needs, a

test engineer writes test cases and tracks defects in the product; it may be difficult to carry out these simultaneously. In the survey, it was observed that people had difficulties on this issue. According to the survey participants, analyzing the needs, writing test cases for them, analyzing the defects in the product, and re-testing them if necessary in a limited time is difficult according to the survey participants.

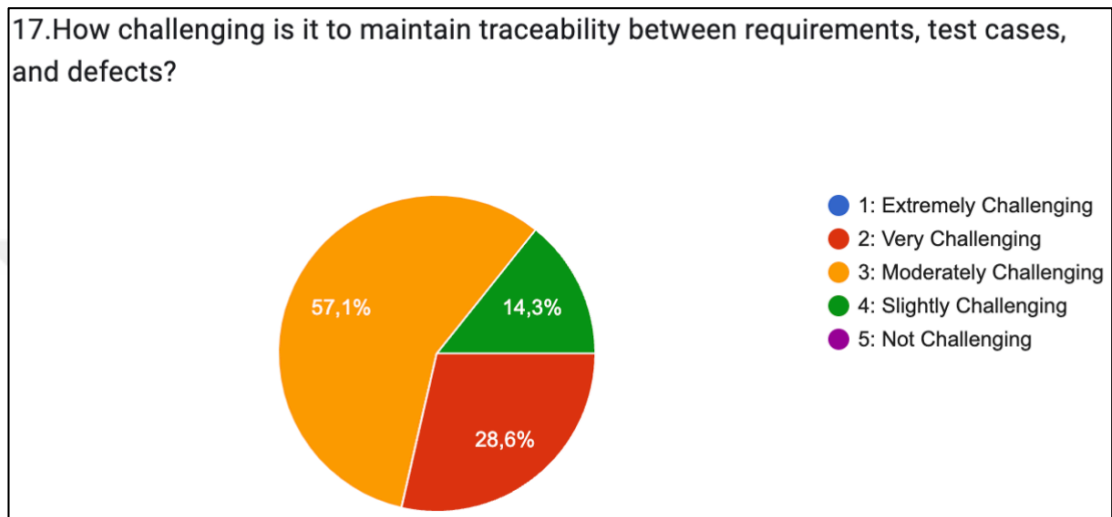


Figure 2.7. Question seven

Although there were different variations and periods, all survey participants reported that they had limited time to manage test cases. In the working methodology called sprint, the period is generally carried out in 10-day periods. Even for people who do not use this methodology but adopt another methodology or whose number of days has changed, testing the test cases should be carried out in a limited time.

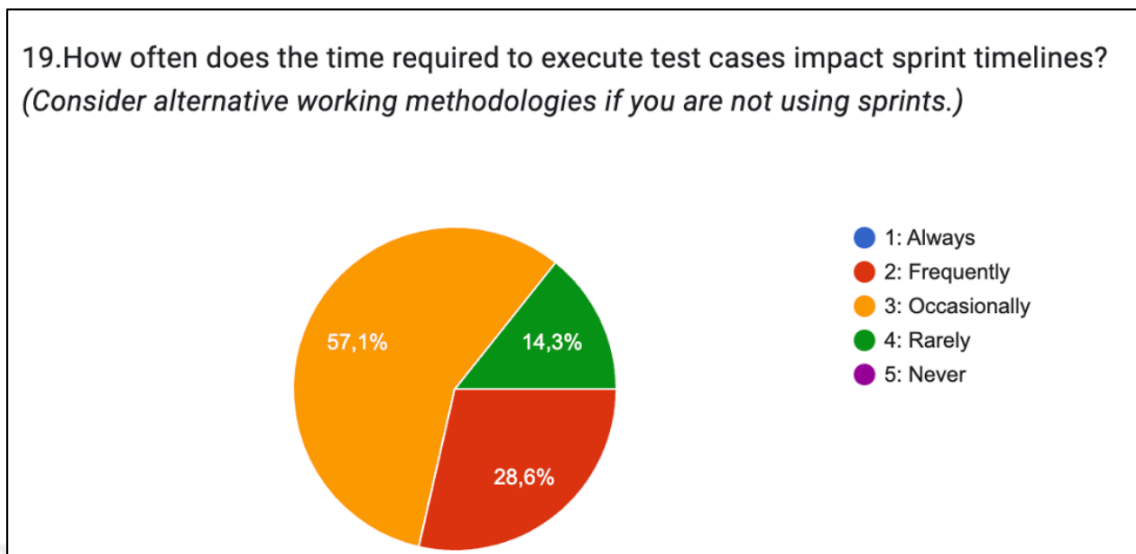


Figure 2.8. Question eight

One of the success criteria envisaged in the study is the faults that will be caught by the test cases tested in a limited period. It was included in the subsequent analysis that the survey participants did not carry out a selection study based on the literature. By taking advantage of this information, if a selection study using the approaches in the literature is carried out, the answer to the graph the error detection rate of the test cases is very frequent and is expected to increase.

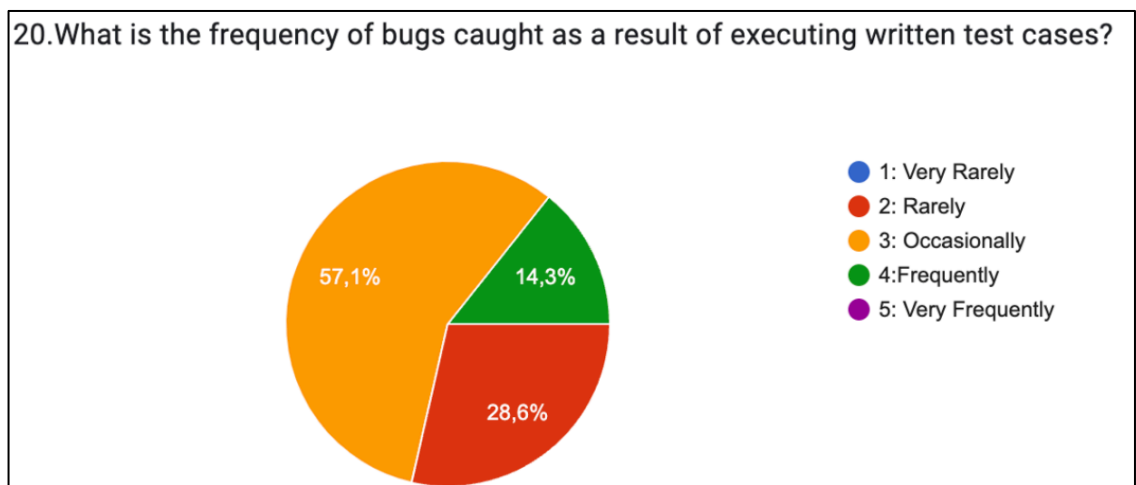


Figure 2.9. Question nine

One of the most important questions of the survey is, how often does bias influence (anticipation) the prioritization of test cases within your team. The answers are given. In line with the answers given, the selection of the tests performed depends on the person's thoughts and knowledge; it has been observed that it has been tested far from the approaches, techniques, and algorithms in the literature.

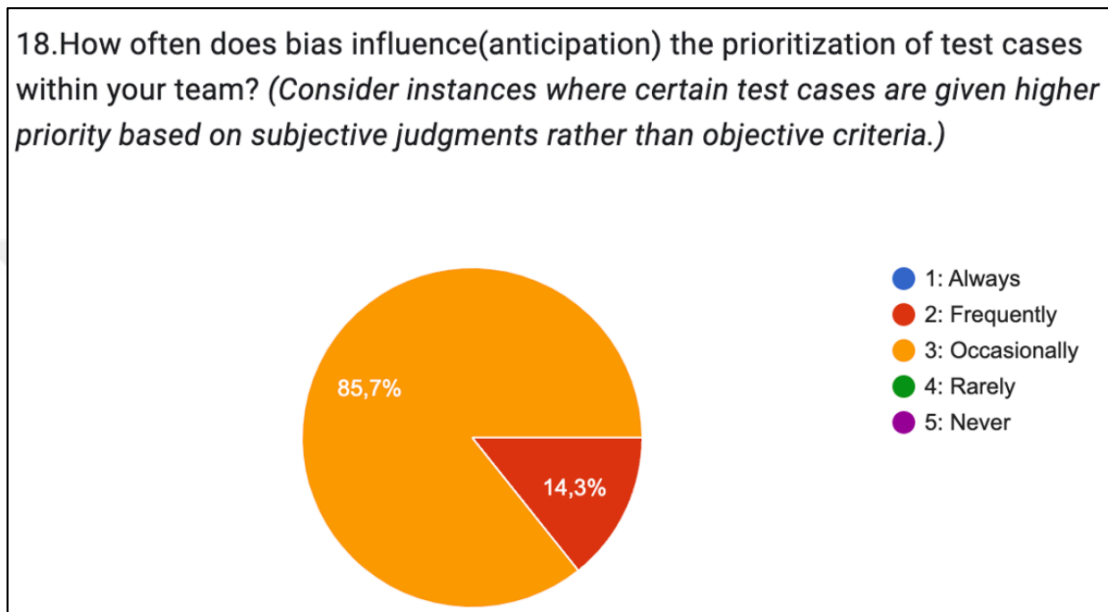


Figure 2.10. Question ten

It has been determined that the general issues include the inefficient execution of test case coverage within the limited time frame and particularly the difficulty in updating test cases during maintenance processes following changes and developments in the software product. Additionally, it was revealed TCP or TCS efforts occur not based on a specific methodology but rather entirely based on personal feelings. Following all these analyses, it was concluded that tests are not conducted efficiently and test engineers face challenges.

2.1.4. Discussions

Based on the survey results, it is evident that the frequency of catching bugs after a written test case is average. Furthermore, most of the participants should complete their test executions within a limited time, and they do not fully utilize test cases effectively, which is supporting the idea of the study's motivation. The participants do not have precise information about studies such as TCS or TCP, but they are not completely close to these

studies. The survey results have shown that testing and quality-related activities in the industry do not proceed smoothly, and time is not used effectively. Due to challenges in time management, the desired defect detection rate cannot be achieved, and, as a result, software products are not tested thoroughly.

During regression periods, some tensions may arise within teams, even if unintentionally. Time is limited, and it is crucial to identify and resolve potential issues during testing as quickly as possible. This is especially true for mobile applications, where the packages intended for customer release must be submitted for approval by the App Store on iOS and Google Play or other online stores for Android and other operating systems. These approval processes can take a long time, which may delay customers' access to the fixes or new features in the software product. This is an undesirable situation because competitors may be able to deliver their software products to shared customers earlier, increasing the likelihood of customers using their products instead. This situation, which creates a butterfly effect, necessitates improvement efforts in both development and testing areas.

After all these results, it is observed that there is a need for an optimization study on test cases that are expected to be carried out during regression testing times to reduce the difficulties experienced by test engineers and produce better quality tests and, as a result, high-quality products.

As mentioned in Section 3, there are already studies in this field in the literature. Upon analysis, it was observed that existing studies focus more on prioritization rather than selection [23,24,25,26]. While there are studies that focus on TCS, these works have not concentrated on nature-inspired algorithms [74,75,76]. The academic papers that focus on TCS using nature-inspired algorithms have primarily explored the differences between algorithms and approaches [59,63,64,65]. It is evident that there is a scarcity of multi-layered studies that hybridize the two different approaches in this study. Algorithms inspired by nature are used relatively less. Furthermore, it has been observed that a multilayered approach, created by hybridizing two different approaches (similarity-based and nature-inspired-based), is lacking in the literature.

This study aims to address this gap by conducting a multi-layered analysis. Thus, selected test cases will be tested in that limited time, the total regression time will decrease, and other

team units will be notified earlier. Test engineers will then be able to perform the remaining tests after the environment transitions (from staging to production) are completed.

2.2. AIM AND OBJECTIVES

This study aims to compare and analyze the test cases expected to be executed during regression periods with limited time using the average per execution cost (APXC) success factor for two approaches. The first approach named clustered nature-inspired selection focuses on clustering test cases based on their similarities, followed by selecting test cases from these clusters using nature-inspired algorithms. As opposed, the second approach named direct nature-inspired selection focuses on the application of nature-inspired algorithms to the test cases without any clustering process and focuses on their selection.

The objective of this study is to evaluate the effectiveness and differences of these approaches, as well as to examine the differences among the algorithms that are influenced by nature. It is the objective of the analysis to determine which algorithm is more effective in certain circumstances. Emphasizing the significance of TCS is the primary objective of this multi-layered study.

To achieve this goal, the first objective is to identify the challenges test engineers face during regression testing across different software environments, such as development, staging, and production.

The second objective is to determine the most relevant success criteria in relation to the problem. Given that time is the central focus of this study, a performance-based success metric is sought to measure this aspect. In this context, it is determined that the APXC success criterion contributes valuable insight into the project.

The third objective is to select the appropriate similarity-based algorithm for clustering and the nature-inspired algorithm for TCS. The chosen similarity-based algorithm should be well-suited for software testing cases and should be easy to implement due to time constraints. Additionally, the nature-inspired TCS technique is examined based on prior research from the literature. The selected algorithms must be adaptable to various environments and variables while also aligning with the success criteria.

The fourth objective is to design the test cases on which the clustering and selection processes are applied. In this regard, an Excel file contains 35 test cases, each of which includes an ID, steps, expected results, and actual results. The objective of these cases is to detect potential defects in a web-based e-commerce product by testing various displays and modules to ensure that they are suitable for retesting.

The fifth objective is to implement a case study for each approach. In the clustered nature-inspired selection, formulas are used to cluster the test cases into subcategories, followed by the application of the selected TCS technique to each subcategory. In the direct nature-inspired selection, the same TCS technique is applied to all test cases without clustering.

The sixth objective is to analyze the nature-inspired TCS techniques based on the approaches (whether clustered or not) using the success criteria. Here, each algorithm is analyzed individually, with its execution times recorded. For comparison purposes, the same TCS algorithms are applied without clustering the test cases, allowing for conclusions to be drawn both at the algorithm and approach levels.

Finally, the seventh objective is to develop a Graphical User Interface (GUI) support application for TCS using the chosen algorithms. The implementation of this application involves Java, HTML, CSS, JavaScript, and other supporting libraries such as Spring Boot.

After all these analyses and problem identifications, it has been determined both in the industry and in the literature that the tests carried out within limited time frames during regression periods are not efficient and, test engineers experience certain difficulties. In the literature, the focus is mostly on TCP rather than TCS. More detailed studies on this subject will be provided in Section 3. Moreover, it has been observed that studies are primarily conducted using algorithms such as history-based, risk-based, requirement-based, and customer-based methods. While the success and outcome criteria of these studies are based on average per fault detection (APFD), it has been found that there is insufficient research on the APXC criterion, which is aimed to be focused on in this study. A hybrid approach is proposed in this study; by mixing both similarity-based and nature-based algorithms, it is aimed to introduce a new perspective to the literature. By focusing on total test execution time with APXC, the success criterion, the following questions are aimed to be answered: What would the total testing time be if a TCS study was not performed? How would the total duration change if a TCS study is conducted? An efficiency ratio is planned to be presented

by comparing the differences before and after the study. Additionally, with the system developed, it is aimed to reduce the difficulties experienced by creating a software interface that can be easily used by test engineers.

2.3. REPORT STRUCTURE

A summary of important ideas, such as software engineering, the SDLC, software test engineering, software quality, software test methodologies, and regression testing, has been discussed up to this point in Section 1. The process of creating a software test case that describes a TCS and its methodology was given particular attention.

The study's motivations are summarized in Section 2 and supported by survey data. This section establishes the framework for the subsequent sections by defining the research questions, study motivations, and objectives. Additionally, Section 3, offers a thorough examination of the academic publications and thesis on TCP and TCS. This section, through an in-depth analysis of the subject matter, clarifies the distinctive features of this study in contrast to previous investigations.

Section 4 elaborates on the rationale behind the selection of algorithms that are inspired by nature and similarity. It also provides a comprehensive justification of the data as well as insights that were derived from it and used to implement and validate these ideas.

The two different approaches that are evaluated in the study are introduced in Section 5.

The implementation and demonstration of the suggested procedures utilizing the collected data are presented in Section 6, providing a practical example of how the algorithms and approaches were used. For the direct nature-inspired selection, the GUI application that has been created makes TCS easier using the chosen algorithms, showcasing the system's usefulness as an extra supporting tool.

3. RELATED WORK

Studies on test case optimization that have been done up to this point are often categorized into two groups in the literature: TCP and TCS. The literature covers papers on TCP, TCS, and both TCP and TCS separately. A few studies have also taken into account alternative strategies, such as TCM or TCR.

The main focus of this study is on TCS. However, it is believed that the findings from TCP studies can contribute to and apply parallel with TCS. Ultimately, the core objective of all these studies is to optimize the test cases within a test suite by success criteria such as APXC, APFD, and other criteria.

In this context, research has been categorized into TCP (in Section 3.1) and TCS (in Section 3.2). Both optimization approaches share common algorithms and success criteria. Thesis studies and academic papers on this subject have been reviewed, focusing on aspects such as the approach used (e.g., history-based, risk-based), success criteria (e.g., APFD, APXC), and the nature-inspired algorithms employed (e.g., BA, GWO, ACO). There are two reasons for analyzing these studies under these headings. The first reason is that the algorithms used form the core of the study. Among the nature-inspired algorithms found in academic papers, the algorithms used in this study have been examined. The second reason is to highlight how this study differs from other thesis studies in the literature.

3.1. RELATED WORK OF TEST CASE PRIORITIZATION

Ufuktepe discusses enhancing the error detection rate by implementing TCP using several probabilistic approaches. As a primary step, the author attempted to perform an impact analysis using Bayesian networks. The author carried out TCP studies with changing probability estimates by interpreting the changing codes on a graphical basis [23].

From the history of test case execution, Ying & Fan concentrate on risk-based TCP. The thesis's objectives are to first identify test case methods using history-based algorithms and subsequently concentrate on the distinctions between the algorithms. The selected test cases are then subjected to these algorithms in the concluding step. Adaptive TCP and similarity-based methods were the two methods that the authors ultimately settled on after meticulously selecting a specific number of literature papers to identify viable alternatives. The authors

can compare the results with those of non-prioritization test cases by applying these methodologies to the selected test cases. The results are calculated using APFD, which is a component of the authors' success criteria [24].

In the year 2000, Chu concentrates on TCP. The research focuses on a variety of techniques and empirical research to evaluate the pace at which faults can be identified in test suites. The pertinent study offered a wide range of statistical insights, with a particular emphasis on APFD, a critical success criterion. It also illustrated the effectiveness of TCP through the use of statistical data and graphics [25].

Omri addresses TCP studies in a data-driven manner and aims to provide resource-optimal feedback at continuous integration (CI) time intervals. The author's approach is followed by studies that predict the error detection rate by creating a regressive model with several code-based churn metrics [26].

The aforementioned and numerous other unmentioned studies tackle the problem of TCP from various perspectives, including risk-based, history-based, data-driven, and probabilistic predictions, and demonstrate the efficiencies achieved through TCP implementation. This thesis's nature-inspired algorithms are explained and adapted to TCP in research outside the thesis.

Ozturk concentrates on determining early fault detection rates in regression tests through prioritization. During this process, Ozturk also elucidated several other nature-inspired algorithms, including ACO and PSO, and identified their similarities and differences with the BA the author implemented. Also, the author's new algorithm approach focuses on high APFD on the individual cost of test cases rather than the total cost of test cases [35].

Bajaj and Sangwan concentrate on TCP using the Cuckoo Search Algorithm (CSA). The authors aimed to convert test case numbers into permutation sequences by presenting this algorithm with a new adaptation strategy from the perspective of asexual genetic reproduction. Thus, the efficiency of cuckoo search algorithms was increased by mutations of GA's [36].

Yue et. al. concentrate on a hybrid algorithm that utilizes both GWO and FA. The GWO emulates the hierarchical behaviors of a grey wolf, while the FA emulates the progression of fireworks for optimization purposes. The new algorithm mixes the FA's exploration

capabilities with the GWO's ability to exploit resources, achieving a balance through a coefficient. They use 16 established benchmark functions, each with diverse dimensions and complexities, to evaluate the effectiveness of this new hybrid algorithm. The hybrid algorithm outperforms the GWO and FA in terms of global search capability and convergence speed [37]. Despite not applying the algorithm to the selected test cases, this algorithm is still applicable to the test cases.

Bajaj and Abraham use a DA for minimizing and prioritizing the test cases. The authors evaluate the algorithm's performance using two success criteria, APFD and test minimization percentage (TMP), which are achieved by reducing the number of test cases. While the DA mimics the behavior of dragonflies in seeking meals, navigating, and avoiding predators, there are two stages of optimization. The authors also focus on a hybrid algorithm that shows better performance rather than the DA itself. However, Bajaj and Abraham also found that the GA produces superior results in terms of numerical values [38].

Another academic study by Bajaj et. al. concentrates on PSO. While DA is inspired by the hunting behavior of dragonflies, PSO is inspired by the social behavior of bird flocking or fish schooling. The research suggests quantum-behaved PSO in accomplishing regression testing. The algorithm enhances the efficiency of the fault detection rate by eliminating unnecessary test cases. Also, the results show the proposed new hybrid algorithm is better than GA, BA, GWO, and PSO in terms of performance [39].

ABO is another technique for TCP. Signal et. al. conducted a comparison between the ABO and the ACO, revealing that both algorithms yield similar results in terms of fault detection rate, depending on the size of the test case after it has been eliminated [56].

Ariffin et al. conducted a comparison between the ACO and the FA, taking into account factors such as APFD, execution time, and fault coverage of selected test cases. The authors observed that both of these algorithms have the same fault coverage; however, the FA has better performance regarding execution time [41].

Prioritization in cloud-based regression testing by Budanahatti et. al. is another approach that uses GWO. The test cases are divided by kernel fuzzy c clustering algorithm first, and then these cluster groups are served in the GWO utilized [42].

Table 3.1 summarizes all these studies by their focused fields and their parameters.

Table 3.1. TCP related works summarized

Related Work	Focused Field	Parameter I	Parameter II	Parameter III	Parameter IV	Parameter V
[23]	History-based	APFD	Law of Minimum	Law of Minimum Additional	Method Total Coverage	Method Additional Coverage
[24]	History-based	APFD	Test Execution Times	Code Coverage	Historical Fault Data and Fault Detection Effectiveness	Test Case Similarity
[25]	History-based	APFD	Total Edge Coverage	Additional Edge Coverage	Fault Exposing Potential	Total Statement Coverage
[26]	History-based	Test Execution Cost	Code Churn Metrics	Fault Density	Execution History Features	Graph Models
[35]	Bat (Nature Stimulated)	APFD	Fault Detection Capability	Cyclomatic Complexity	Execution Time	Individual Test Case Cost
[36]	Cuckoo (Nature Inspired)	Asexual Reproduction Algorithm	Permutation Sequences	Permutation Encoding	Random Seaching	Test Case Prioritization
[37]	Grey Wolf and Fireworks (Nature Inspired)	Exploration Capability	Exploitation Capability	Convergence Speed	Fitness Variance	Mean Fitness
[38]	Dragonfly (Nature Inspired)	APFD	Test Minimization Percentage	Swarming Weights (s,a,c,f,e)	Convergence Rate	Neighborhood Radius

[39]	Swarm (Nature Inspired)	APFD	Inclusivity	Test Selection Percentage	Cost Reduction Percentage	Fault Detection Loss Percentage
[56]	African Buffalo (Nature Inspired)	APFD	Execution Time	Buffalo Fitness	Exploration Weight	Exploitation Weight
[41]	Ant Colony (Nature Inspired)	APFD	Execution Time	Fault Coverage	Iterations	Path Cost (Algorithm by algorithm)
[42]	Grey Wolf (Nature Inspired)	Kernel Fuzzy C- Means (KFCM) Clustering	Memory Usage	Code Coverage Metrics	Execution Time	Test Case Prioritization

3.2. RELATED WORK OF TEST CASE SELECTION

Olofsson and Larsson focus on TCS. The authors' hybrid algorithm relies on recent code changes and risk factors associated with regression testing bugs. Previously, bug factors were important for the recent code change method. In the recent code change, these bugs are fixed, and it focuses on the commitment to solving these bugs. The risk factor method matches test cases to these commits through a text analysis of commit messages. As can be understood from the study, a risk-based study was carried out and is based on making risk-related predictions [74].

In the author's thesis, Cheruiyot focuses on TCS using a machine learning-driven approach, utilizing models such as logistic regression, Gaussian Naive Bayes, gradient boosting, and k-nearest neighbors. The author focuses on eliminating and categorizing unnecessary test data. This data converts to a numerical value. Next, the author applies the machine learning process to TCS, taking into account the characteristics of text values, such as test case titles. As a result, the author found that machine learning processes can be applied to TCS [75].

Valk focuses on TCS, TCP, and TCM in the thesis. The author investigates the prediction of test execution time through correlation. The author implements an asymmetric loss function approach and selects test cases based on historical runtimes. It can be said that the author focuses on a history-based approach. APFD is considered the main success criteria of the thesis [76].

Palak et al. have implemented a hybrid approach that combines ACO and GA. The proposed algorithm, Hybrid Ant Colony-Genetic Algorithm (HACGA), performs better than traditional approaches. A fault matrix is created by 15 test cases and their execution times. After the HACGA algorithm is applied, a reduced fault matrix with selected test cases is investigated. As a result, the new hybrid approach performed better in terms of fault coverage and execution time. From this, it is concluded that some hybrid algorithms exist in the literature; in fact, the authors can provide better results than traditional algorithms [59].

Neto et al. focus on TCS by using similarity-based approaches. The provided approach, Similarity for Regression Testing (SART), selects the test cases of a model. SART is not comparing models; it identifies modifications by similarity analysis among test cases. By doing that, SART reduces its dependency on other models. The authors use statistical techniques, such as p-values, to observe statistically significant differences between treatments concerning transition and modification coverage [63].

Raamesh et al. conducted a study where they developed an algorithm for TCS based on concepts from computer games. The authors provide an algorithm called the Hybrid Battle Royale-based Remora Optimization (HBR2O) algorithm for TCS with minimum prioritization and less memory consumption. The authors suggest that the Battle Royale Algorithm (BRO) is powerful, especially for execution times; however, due to the low convergence rate, its performance is reduced. The authors have implemented Remora Optimization (RO) to enhance the performance of the BRO algorithm by combining various algorithms. As a result, the proposed algorithm gives a higher detection rate with less execution time and memory consumption than other existing techniques [64].

Dalal et al. implement the hybrid optimization technique of Bee Colony Optimization (BCO) and GA. By comparing the rate of change in the Fitness Score (FS) and the variation in the number of Decision Nodes Covered (DNC), it was found that BCO-GA showed enhanced performance. This improvement led to a reduction in the time required for software testing

while also increasing the quality of the generated solutions. However, there is no automated method for fault rectification, which remains a limitation [64,65].

Table 3.2 summarizes all these studies all these studies by their focused fields and their parameters.

Table 3.2. TCS related works summarized

Related Work	Focused Field	Parameter I	Parameter II	Parameter III	Parameter IV	Parameter V
[74]	Risk-based	Code Commit Size	Keyword Extraction Score	Code Change Risk Score	Test Case to Commit Matching Score	Test Coverage Impact
[75]	Machine learning-driven	Textual Feature Extraction	Feature Selection Techniques	Classifier Models	Confusion Matrix Analysis	Cross Validation Optimization
[76]	History-based	APFD	Test Case Execution Time	Risk Score Of Test Cases	Historical Test Failure Data	Test Suite Execution Time
[59]	Nature-Inspired-based	Fault Matrix	Pheromone Intensity	Fitness Function	Mutation Rate	Execution Time
[63]	Similarity-based	Similarity Score	Modification Coverage	Transition Coverage	Fault Detection Capability	Execution Cost
[64]	Hybrid Battle Royale-based	Battle Royale Optimization	Remora Optimization	Fitness Function	Memory Usage	Execution Time
[65]	Nature-Inspired-based	Decision Nodes Coverage	Fitness Score	Nodular Path Sequences	Binary Representation	Swarm Parameters

3.3. SUMMARY

Current research includes investigations on TCP utilizing risk and history-based approaches described in Tables 3.1 and 3.2 [23,24,25,26,74,76]. These approaches often prioritize test cases by using past test results or potential risk factors. The APFD serves as the primary metric for success in these investigations. The studies [23,24,25,35,38,39,56,41,76] focus on APFD success criteria. The objective is to enhance the effectiveness of fault detection to maximize the rate of bug identification.

It is noted that Bayesian networks and other probabilistic models that significantly rely on historical data are often used in TCP thesis investigations [24,25,26,36,65,75]. An essential component of these investigations is statistical data. In the study [24], the parameter of historical fault data and fault detection effectiveness is an example parameter for statistical data or in the study [37], exploration capability, exploitation capability, convergence speed, fitness variance, and mean fitness are the example parameters of this statistical approach. However, scholarly publications are mostly concerned with creating algorithms inspired by nature and introducing new methods to the literature. Within the parameters of their research, these studies frequently evaluate the effectiveness of different algorithms inspired by nature. Some studies additionally take execution time into account, even though the APFD metric is frequently utilized as a success criteria [76]. Similar to TCP studies, history-based and risk-based techniques are extensively researched in TCS [74,76]. In these investigations, probabilistic elements are important, and multi-layered research that concurrently addresses selection, reduction, and priority is evident. Furthermore, several scholarly studies concentrate on combining two different nature-inspired algorithms to create novel techniques.

A gap in the existing studies is highlighted by the lack of studies on nature-inspired algorithms and APXC. Although research has been conducted using nature-inspired algorithms in existing studies [35,36,37,38,39,56,41,42], there are very few studies that hybridize the similarity-based and nature-based algorithms and approaches in this study and compare them based on the APXC success factor. This gap, as discussed in Section 2.1, prompted the development of a novel approach that combines algorithms inspired by nature with similarity-based methodologies. Bringing these different techniques together was intended to present a novel strategy that stands out. Furthermore, the goal of comparing nature-inspired algorithms among themselves was to enhance the value of previous research and offer insights that may help guide future investigations.

4. RESEARCH METHODOLOGY

Based on the motivation explained in detail in Section 2, it is expected that selecting the correct tests for execution within the limited timeframe of regression testing will minimize the issues encountered. In Section 3, it is identified that while existing studies in the literature address some gaps in this field, there is a lack of thesis studies that compare these algorithms individually. Furthermore, there is a shortage of studies focusing on a new approach that combines similarity-based and nature-inspired algorithms.

Section 4 focuses on the approaches included in this study. It emphasizes the similarity-based approaches between test cases and the new approaches using nature-inspired algorithms, which are the focus of the study. Accordingly, Figure 4.1 illustrates the approaches and techniques employed in the research. The clustering approach utilized in the study has been designated with Jaccard. The selection techniques are identified as BA, ABO, GWO, PSO, and ACO. The success criteria for the study are APXC.

Clustered nature-inspired selection encompasses both clustering and selection phases, whereas direct nature-inspired selection consists only of a selection process utilizing nature-inspired algorithms. Section 4.1 defines clustered nature-inspired selection, a clustering approach that categorizes test cases according to their similarities utilizing the Jaccard. Conversely, Section 4.2 defines direct nature-inspired selection, which excludes any clustering procedure. Section 4.3 defines the success metric of the study with the APXC formulation. Furthermore, Section 4.4 defines the data employed in the study, along with the data applied in the case study discussed in Section 6.

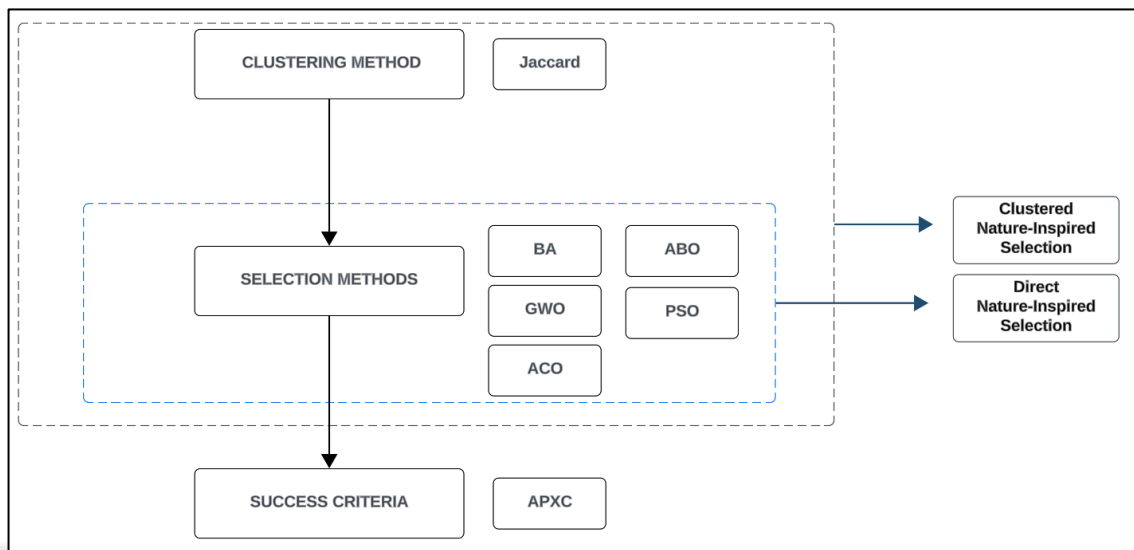


Figure 4.1. Flow chart of the research methodology

4.1. DETERMINING THE CLUSTERING METHODS

Including similar cases in the test suite and testing them within a limited timeframe during regression testing can lead to a decrease in tester efficiency. In this regard, by carrying out a similarity-based study, test cases within the test suite are grouped separately according to their similarities. An attempt was made to create a hybrid approach by performing a TCS study on these clustered cases.

When the studies in the literature are examined, it is seen that priority or selection studies have generally been carried out in a single method. A hybrid approach that utilizes multiple methods can yield more effective results within prioritization and selection processes. A different perspective is aimed to be brought to the literature by hybridizing two different approaches (similarity-based and nature-inspired, as in this study). Also, in situations where the number of test cases is large, performing TCS studies over all these cases can be challenging. In this regard, as a first step, the test cases are clustered according to their similarities, and then the algorithms are implemented through the clustered test suites; it is anticipated that the process will be managed more healthily and efficiently.

The Jaccard method was used within the scope of the project. It is simple to understand and implement. Following a comprehensive analysis, it was determined that the Jaccard method has a more comprehensible formulation and is easier to implement compared to other

methods. The study's focus is on nature-based algorithms rather than similarity-based approaches. Similarity-based approaches are considered supportive approaches to the study, helping to create a hybrid solution. Due to this, it was not desired to lose focus on this approach. Efforts were aimed to be directed toward nature-based algorithms.

The Jaccard method can be applied to software testing. Unlike Euclidean distance, which is better suited for continuous data, the Jaccard focuses on similarity based on sparse data [61]. Test cases can be sparse data because they typically include a series of tests covering various features or modules of software. Each test case tests a specific part of the software and does not cover all possible cases.

4.2. DETERMINING THE TEST CASE SELECTION METHODS

Zheng et al. state that most of the previous research on regression TCP has been based on greedy algorithms, which might not give the best results because they only find local minima in the search space. As a result, they claimed that evolutionary search algorithms and meta-heuristics work to avoid these problems. They offer the findings of an empirical investigation involving the use of several greedy, meta-heuristic, and evolutionary search algorithms [66].

Nature-inspired algorithms take a different approach to problem-solving by taking cues from the workings of the natural world. This strategy is particularly suited to handle challenging optimization challenges like TCS because it mimics the adaptive mechanisms seen in biological systems. While some algorithms date back to the early 1990s, others, such as BA, were created around the year 2010 [62]. Considering how new this breakthrough is, there are still a lot of problems to be solved. After reviewing the literature, the lack of academic research on this topic led us to use these strategies are decided.

Furthermore, algorithms with naturalistic inspiration show adaptability to changing conditions, which is highly beneficial for software testing. TCS needs to change as software systems do in order to ensure completion within the given time frame. Algorithms inspired by nature can be useful in this sense because, like living organisms in their natural environments, they are able to dynamically adjust their plans in response to changing conditions.

The study of animal behaviors and activities provides extra motivation for examining nature-inspired algorithms in the framework of TCS. Cooperative, communicative, and collective decision-making behaviors are seen in many animal species, and these behaviors offer valuable insights into the design and optimization of algorithms. Through the analysis and simulation of these organic processes, people may create algorithms that take advantage of the intrinsic intelligence and efficiency of biological systems. TCS benefits from increased operational efficiency.

The selected nature-based algorithms were used to be examined in this study because more academic articles on the algorithms were obtained. Furthermore, given the limited timeframe for completing this study, it is essential to concentrate on certain algorithms that exhibit the most potential and significance for the objectives of this inquiry. This focused approach ensures the study's quality by facilitating a more thorough analysis and in-depth investigation of the selected algorithms.

The TCS methods BA [57], GWO [37], ACO [72], ABO [56], and PSO [39] are used in this project.

4.3. DETERMINING THE SUCCESS CRITERIA

In the literature, success criteria are commonly used for TCS, TCP, and TCM. Every test case optimization technique shares common criteria such as APXC, APFD, Average Percentage of Code Coverage (APCC), and Average Percentage of Requirements Covered (APRC). Each of these criteria provides a way to measure the effectiveness and efficiency of the test cases in different aspects of software testing.

When all these criteria were examined, it was realized that they did not serve the success factor in this study, which is saving time by performing the right test in a limited time. In this regard, a success factor based directly on the subject of time was sought, and the APXC success factor was noticed. The APXC criteria are based on execution cost. For this condition, the execution cost can be time-related. Equation 4.1 identifies the APXC formulation.

$$APXC = \frac{\sum_{i=1}^k \text{Execution Time } (i)}{\sum_{j=1}^n \text{Execution Time } (j)} \quad (4.1)$$

Where,

- n: Total number of test cases.
- k: Number of selected test cases.
- Execution Time (*i*): Execution time of the *i*-th selected test case.
- Execution Time (*j*): Execution time of the *j*-th test case in the entire test suite, also known as, the total execution time.

4.4. DETERMINING THE TEST CASES

The concept of a test case's significance and how it is prepared is explained in detail under Section 1.2.5. A test case is one of the most essential tools for a test engineer and serves as a form of documentation. Test engineers who evaluate the software product can utilize these documents to conduct their tests. These test cases must be explained in as much detail as possible. Even a new team member is expected to be able to execute the test by simply following the documented steps.

In this context, the test cases to be used in the evaluation presented in Section 6 have been selected. These test documents are expected to serve as a basis and guide for this study.

4.4.1. Data Collection

The data selected as input for the algorithms to be used in the study are test cases. These test cases are prepared for a website that is an e-commerce site replica. Figure 4.2 displays this website. Figure 4.2 and Figure 4.3 are captured from this website and they are used as examples within the scope of an academic study; it is not intended for commercial purposes. All rights to the original website are reserved. A detailed analysis of the website was performed to identify all functionalities and features that need to be tested. Despite the website being an e-commerce site replica, most of the features are standard or common to every e-commerce application, such as sign-up, login, product search, cart operations, checkout process, and user account management. This involved exploring the user interface, understanding the workflows, and documenting the expected behaviors. First, an exploratory

test was conducted. The framework of the test data was determined by identifying the first modules that stood out when entering the User Interface (UI) layer of the application. After these modules, the application was examined module by module, and a scope was created. Figure 4.3 represents the modules of the website.

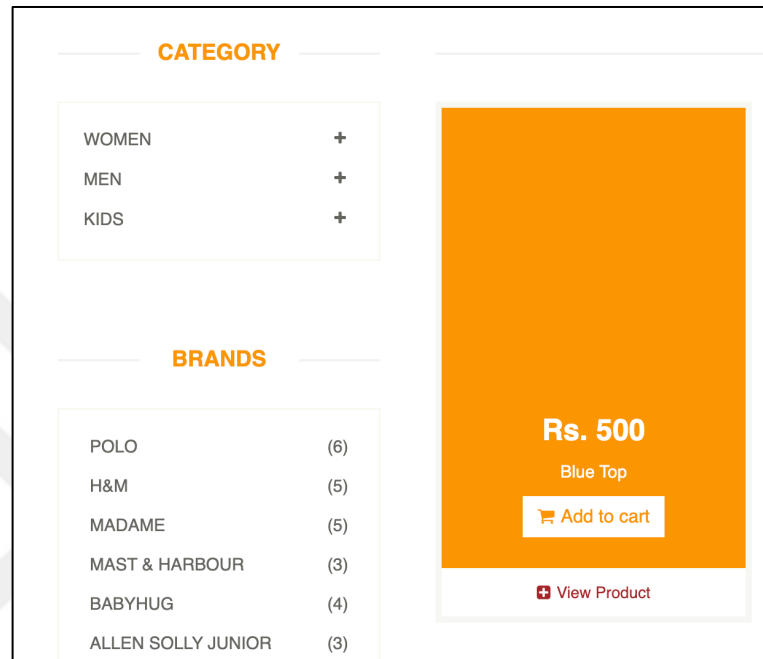


Figure 4.2. Web-based e-commerce site replica

By doing this exploratory test, the aim was to identify the potential edge cases. This helps in understanding the application from a user's perspective and identifying any hidden functionalities, such as the view product or any other submodule.

Thinking like a customer before testing the application makes it easier to create test cases. However, during the testing processes, thinking not only like the user but also considering the edge cases of the application is of enormous importance for the early detection of possible errors. In this regard, edge cases should always be considered. Entering only special characters or numbers in the sign-up e-mail field can serve as an example of an edge case. While collecting data, attention was paid to both potential user behavior and edge cases, as well as positive and negative ones.

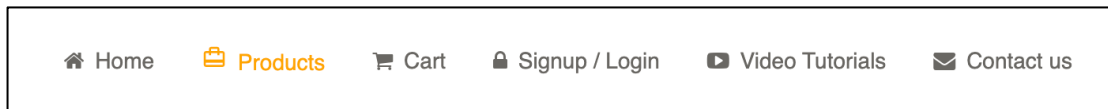


Figure 4.3. Web-based e-commerce site replica modules

4.4.2. Data Synthesis

As stated in Section 1.2.5, while preparing the test cases, care was taken to ensure that each case had headings such as a test ID, test title or description, pre-condition, test steps, expected result, and actual result. Test cases prepared according to this structure are given in the Appendix A section. Figure 4.4 represents an example of the synthesized test case data with the specified headings.

A	B	C	D	E	F	G
Test ID	Test Description	Pre-Condition	Test Steps	Test Data	Expected Results	Actual Results
TC1	User Registration with Valid Data - Sign Up	1. Email area should be contain @ character. 2. There should not be existed email used.	1. Open the website. 2. Navigate to "Sign Up". 3. Enter valid user details. 4. Click "Signup" button.	Name: test Email: test123@test.com	The user should successfully go to the "Account Information" page.	
TC2	User Registration with Invalid Data - Sign Up	1. Email area should not be contain @ character.	1. Open the website. 2. Navigate to "Sign Up". 3. Enter invalid user details. 4. Click "Signup" button.	Name: test Email: test123test.com	An error should be visible.	
TC3	Enter Account Information - Sign Up	1. All necessary areas should be completed. 2. Steps that TC1 should be completed.	1. Fill these areas: Title, Name, Email, Password, Date of Birth, First Name, Last Name, Company, Address, Country, State, City, Zipcode, Mobile Number. 2. Click "Create Account" button.	Title: Mr Name: TestName Email: test123@test.com Password: Test*123 Date of Birth: 1stJan1990 First Name: Test Last Name: Test Company: Test Address: Test Country: Tr State: Ege City: Izmir Zipcode: enter a 6 digit zipcode Mobile number: enter a dummy number	The user should successfully see the success message as "Account Created!"	
TC4	Enter Account Information - Missing Necessary Areas - Sign Up	1. All necessary areas should be completed. 2. Steps that TC1 should be completed.	1. Fill these areas: Title, Name, Email, Date of Birth, First Name, Last Name, Company, Address, Country, State, City, Zipcode, Mobile Number.		1. "Password should be entered!" information should be visible. 2. The user cannot create his/her account.	

Figure 4.4. Test cases synthesizing data

In the thesis, 35 test cases were used. These cases were designed to test every module of the application as much as possible. While some tests focused on positive results (successfully ordering a product), some tests focused directly on negative tests (focusing on receiving an error by entering invalid data when signing up). These test cases do not have an upper limit breakpoint. The number of test cases can be increased.

4.4.3. Data Analysis

Every test case in the study has a unique ID, which is observed through data analysis. For tracking reasons, these test IDs, which have a range of one to 35, must be unique. When creating defect reports based on test results, the test case ID can be utilized directly. Furthermore, having distinct IDs helps avoid misunderstandings if these test cases are migrated to a test automation solution.

A description is included for every test case. This summary serves as the case's title and makes it evident what the case entails and what topics it covers. Even if they are not the ones administering the test, it needs to paint a clear image for everyone who reads it.

The pre-condition section lists the steps that must be completed before executing the test case. Most of the test cases used in this study have a precondition. To minimize dependencies between test cases, it is important to keep the number of pre-conditions as low as possible.

The steps for each test case are provided in detail. By following these steps, the testing of the product can be performed. The names of buttons or fields that require action are enclosed in quotation marks (for example, click the sign-up button).

Each test case also has its own test data. The purpose of using these data sets is to ensure that any person other than the author of the test case can execute the test without the need for an additional data search.

The expected result is an important column in every test case. This result can be achieved by the tester, passing the evaluation. Otherwise, the test is to be declared unsuccessful if the actual result deviates from the expected one.

The expected and actual result sections are not considered when choosing test cases for this study. For the purpose of choosing test cases, these fields are irrelevant. The steps and explanation of the test should be the main focus instead.

5. PROPOSED APPROACH

In Section 2, the general motivation of the study was explained, stating that a test case optimization would be conducted by performing selection within the limited time during regression periods. In Section 3, the differences between this study and the existing ones in the literature were detailed. Section 4 described the approaches that form the focus of the study, and the analyses of the data to be used in the evaluation part in Section 6 were conducted.

In this study, two approaches are proposed. This section further elaborates on the previous sections and examines the approaches proposed for carrying out this TCS study under different headings. While Section 5.1 focuses on the first approach, which involves conducting TCS research through clustered nature-inspired selection, Section 5.2 provides details on the information used in direct nature-inspired selection, the second approach of this study.

To sum up, the first approach named clustered nature-inspired selection uses the similarity method to organize test cases into clusters based on similarities. Then, it applies TCS methods to test cases that have a lot of clusters, and the optimum test cases are chosen. In the second approach named direct nature-inspired selection, all test cases are subjected to TCS steps without performing a similarity-based clustering study.

In both approaches, a TCS study for regression test cases is conducted to ensure the efficient utilization of the limited time available during regression periods. However, following a detailed literature review, the idea of incorporating a new perspective by integrating clustering based on similarity, that is an alternative approach to test case optimization, emerged, making it a potential subject of academic research. This concept was considered interesting, and the potential consequences were worth investigating, which resulted in the creation of clustered nature-inspired selection.

This study concludes with a comparative analysis of these methodologies. Additionally, a multi-layered discussion is conducted to evaluate the effectiveness of each approach by contrasting the nature-inspired algorithms against one another, with the objective of identifying the variation that demonstrates the best results.

5.1. CLUSTERING PROCESS BEFORE TCS

The clustered nature-inspired selection approach focuses on clustering test cases based on their similarities first. For this, the Jaccard Distance method is used. Further information is described in Equation 5.2. With the Jaccard method, test cases are clustered according to similarities. This "clustering" process is carried out with the idea that if there are a large number of test cases, TCS can be performed more easily on these cases. It is carried out with the "divide and conquer" principle of software engineering, with the idea that dealing with less data can be more efficient and easier.

$$Sim(u, v)^{Jaccard} = \frac{(I_u \cap I_v)}{(I_u \cup I_v)} \quad (5.1)$$

The Jaccard similarity index primarily addresses overall ratings. It calculates the ratio of the number of items rated by both users to the total number of items rated by either user. The sets of items rated by users u and v , respectively, are represented by I_u and I_v . [71].

To cluster the test cases, each one is first represented as a set of features or keywords derived from their descriptions, steps, and expected results. The Jaccard similarity between any two test cases is then calculated using the formula that is described in equation 5.1.

For the study's case, I_u and I_v are the sets of features for test cases u and v . The numerator ($I_u \cap I_v$) represents the number of common features between the two test cases, while the denominator ($I_u \cup I_v$) represents the total number of features across both test cases.

Equation 5.2 shows a distance was calculated using the Jaccard similarity matrix to reduce the number of clusters.

$$Distance = 1 - Jaccard\ Similarity \quad (5.2)$$

The closer the distance value is to zero, the more similar the two test cases are. It makes sense to cluster these test cases. The closer the distance value is to one, the greater the difference between the two test cases, so it makes sense to place them in different clusters.

An example from the study's data for calculating the Jaccard distance is explained. First, the Jaccard similarity should be calculated and after that, the Jaccard distance can be observed:

TC1: User registration with valid data – sign up

TC2: User registration with invalid data – sign up

TC5: User login with valid credentials – login

TC1: {registration, valid data, sign up}

TC2: {registration, invalid data, sign up}

TC5: {login, valid credentials}

Calculating the Jaccard similarity:

Sim(TC1, TC2): “registration” + “sign up” / “registration” + “data” + “sign up”

$$2/3 = 0.67$$

Sim(TC1, TC5): no common features / “registration” + “data” + “sign up” + “login” + “credentials”

$$0/5 = 0$$

Sim(TC2, TC5): no common features / “registration” + “data” + “sign up” + “login” + “credentials”

$$0/5 = 0$$

Calculating the Jaccard distance:

A distance can be calculated for decreasing the number of clusters. For this, equation 5.2 can be applied.

Distance (TC1, TC2): 1-Sim(TC1, TC2)

$$1-0.67= 0.33$$

Distance (TC1, TC5): 1-Sim(TC1, TC5)

$$1-0=1$$

Distance (TC2, TC5): 1-Sim(TC2, TC5)

$$1-0=1$$

Lower distance values suggest that two test cases are more likely to be designated to the same cluster and are located closer to one another.

In this instance, TC1 and TC2 are more similar to TC5 in terms of Jaccard similarity and closer to TC5 in terms of Jaccard distance. This results in the classification of TC1, TC2, and TC5 as distinct entities.

Cluster I: TC1, TC2

Cluster II: TC5

Nature-inspired algorithms can be applied to select test cases for each cluster following the completion of this clustering process.

5.2. THE DIRECT APPLICATION OF TCS

Direct nature-inspired selection concentrates on the TCS process by directly applying nature-inspired algorithms to test cases without the need for a similarity-based clustering operation. This way, the potential advantages of clustered nature-inspired selection (separating test cases cluster by cluster based on their similarities) over direct nature-inspired selection can be analyzed. The system design of the project, which is detailed in Section 6.3, directly includes direct nature-inspired selection. All existing test cases are uploaded to the system, and TCS is carried out using the nature-inspired algorithm chosen by the user. Along with direct nature-inspired selection, the differences between nature-inspired algorithms will also be addressed to examine which algorithm provides a more optimal result based on the success criteria.

The study's success criterion, APXC, anticipates an increase in efficacy in both approaches, with a primary emphasis on the time factor. In both approaches, the test cases will be subjected to a selection process to prevent the necessity of testing all cases during the restricted regression period.

6. EVALUATION

This section contains a demonstration of clustered nature-inspired selection and direct nature-inspired selection, which are explained in detail in Sections 4 and 5. The previous sections described these approaches and their implementation. In this section, an example study is carried out in order to draw results related to the approaches. It includes a study conducted on 35 test cases (Appendix A), written to test the web-based product mentioned in Section 4.4.1. Table 6.1 shows this test cases.

The study aims to highlight the differences between clustered nature-inspired selection and direct nature-inspired selection approaches, as well as the differences among the nature-inspired algorithms. Section 6.3 contains an additional study for direct nature-inspired selection that is independent of the success criteria of the main study.

This evaluation was conducted on a machine with the following specifications:

- Chip: Apple M1
- Total Cores: 8 (4 Performance Cores, 4 Efficiency Cores)
- Memory: 8 GB
- Operating System: macOS Sonoma 14.5

Table 6.1. Test cases for the approaches

Test Case ID	Test Description
TC1	User Registration with Valid Data - Sign Up
TC2	User Registration with Invalid Data - Sign Up
TC3	Enter Account Information - Sign Up
TC4	Enter Account Information - Missing Necessary Areas - Sign Up
TC5	User Login with Valid Credentials - Login
TC6	User Login with Invalid Credentials - Login
TC7	Product Search with Valid Term
TC8	Product Search with Invalid Term
TC9	Adding a Product to the Cart
TC10	Removing a Product from the Cart

TC11	Updating Product Quantity in Cart
TC12	Logging Out
TC13	Login without Email
TC14	Login without Password
TC15	Register with Special Characters in Email
TC16	Proceeding to Checkout
TC17	Checkout with Empty Cart
TC18	Search with Empty Input
TC19	Searching for Out-of-Stock Items
TC20	Product Categories
TC21	Checking Cart After Login
TC22	Product Brands
TC23	Verify Scroll Up using 'Arrow' button
TC24	Verify Scroll Up without 'Arrow' button
TC25	Searching Results by Price
TC26	Navigating Product Details on the Cart
TC27	Verify "Contact Us" Function Works - Without Attachment
TC28	Verify "Contact Us" Function Works - With An Attachment
TC29	Account Deletion
TC30	Subscription Functionality
TC31	Subscribing with Invalid Email
TC32	Filtering by Product Brands
TC33	Entering Shipping Information
TC34	Entering Payment Information - Invalid Characters
TC35	Entering Payment Information - Valid Characters

6.1. CLUSTERED NATURE-INSPIRED SELECTION APPROACH

The clustered nature-inspired selection which is first clustering then TCS with nature-based algorithms is discussed in this section. A table of common words among the 35 written test cases was created. More information about test cases is included in Appendix A. Furthermore, Table 6.2 displays the keyword analysis for each test case. For example, for the TC1 registration with valid data sign up, the keywords are user registration, valid data, sign up, and email.

Table 6.2. Test cases separated by keywords

Test Case ID	Keywords
TC1	user registration, valid data, sign up, email
TC2	user registration, invalid data, sign up, email, error
TC3	account information, sign up, create account, details
TC4	account information, missing fields, sign up, error
TC5	user login, valid credentials, dashboard
TC6	user login, invalid credentials, error
TC7	product search, valid term, results
TC8	product search, invalid term, no results
TC9	add product, cart, success
TC10	remove product, cart
TC11	update product, cart
TC12	logout, redirect, homepage
TC13	login, no email, error
TC14	login, no password, error
TC15	register, special characters, email, error
TC16	checkout, cart, address details
TC17	checkout, empty cart, error
TC18	search, empty input, all products
TC19	search, out-of-stock items, no results
TC20	product categories, view categories
TC21	cart, retain products, login
TC22	product brands, view brands
TC23	scroll up, arrow button
TC24	scroll up, no arrow button
TC25	search, results by price
TC26	navigate, product details, cart
TC27	contact us, without attachment, submit
TC28	contact us, with attachment, submit
TC29	account deletion, confirm
TC30	subscription, verify
TC31	subscription, invalid email, error
TC32	filter, product brands
TC33	shipping information, checkout
TC34	payment information, invalid characters, error
TC35	payment information, valid characters, success

6.1.1. Clustering Process

Based on the Jaccard similarity-based approach, a similarity-based matrix was created. All details of the matrix can be found in Appendix B.

A	B	C	D	E	F	G	H	I	J	K
	TC1	TC2	TC3	TC4	TC5	TC6	TC13	TC14	TC15	TC31
TC1	1	0,5	0,1428571429	0,1428571429	0	0	0	0	0,1428571429	0
TC2	0,5	1	0,125	0,2857142857	0	0,1428571429	0,1428571429	0,1428571429	0,2857142857	0,1428571429
TC3	0,1428571429	0,125	1	0,3333333333	0	0	0	0	0	0
TC4	0,1428571429	0,2857142857	0,3333333333	1	0	0,1666666667	0,1666666667	0,1666666667	0,1428571429	0,1666666667
TC5	0	0	0	0	1	0,2	0	0	0	0
TC6	0	0,1428571429	0	0,1666666667	0,2	1	0,2	0,2	0,1666666667	0,2
TC13	0	0,1428571429	0	0,1666666667	0	0,2	1	0,5	0,1666666667	0,2
TC14	0	0,1428571429	0	0,1666666667	0	0,2	0,5	1	0,1666666667	0,2
TC15	0,1428571429	0,2857142857	0	0,1428571429	0	0,1666666667	0,1666666667	0,1666666667	1	0,1666666667
TC31	0	0,1428571429	0	0,1666666667	0	0,2	0,2	0,2	0,1666666667	1
TC9	0	0	0	0	0	0	0	0	0	0

Figure 6.1. The similarity-based matrix based on Jaccard

The diagonal always carries the value of one. The similarity of the same test case with itself will always be one, because the intersection and union contain the same set.

The Jaccard Similarity-Based approach in the initiative was primarily concerned with clustering based on distance. As previously stated, the objective was to minimize the number of clusters by employing distance. At the outset, the initiative employed similarity-based ratios; however, the presence of an increasing number of clusters necessitated the implementation of a distance-based approach.

The similarity-based distance matrix is the primary focus of Figure 6.1. Appendix C contains all the matrix's specifics. Furthermore, Figure 6.3 illustrates the heatmap visualization of the Jaccard matrix, while Figure 6.4 illustrates the heatmap of the Jaccard distance matrix. These visualizations are designed to simplify the representation process by depicting the ratios of similarity and dissimilarity among the test cases.

A	B	C	D	E	F	G	H	I	J	K
	TC1	TC2	TC3	TC4	TC5	TC6	TC13	TC14	TC15	TC31
TC1	0	0,5	0,8571428571	0,8571428571	1	1	1	1	0,857	1
TC2	0,5	0	0,875	0,7142857143	1	0,8571428571	0,85	0,85	0,714	0,8571428571
TC3	0,8571428571	0,875	0	0,6666666667	1	1	1	1	1	1
TC4	0,8571428571	0,7142857143	0,6666666667	0	1	0,8333333333	0,83	0,83	0,857	0,8333333333
TC5	1	1	1	1	0	0,8	1	1	1	1
TC6	1	0,8571428571	1	0,8333333333	0,8	0	0,8	0,8	0,833	0,8
TC13	1	0,8571428571	1	0,8333333333	1	0,8	0	0,5	0,833	0,8
TC14	1	0,8571428571	1	0,8333333333	1	0,8	0,5	0	0,833	0,8
TC15	0,8571428571	0,7142857143	1	0,8571428571	1	0,8333333333	0,83	0,83	0	0,8333333333
TC31	1	0,8571428571	1	0,8333333333	1	0,8	0,8	0,8	0,833	0
TC9	1	1	1	1	1	1	1	1	1	1
TC10	1	1	1	1	1	1	1	1	1	1
TC11	1	1	1	1	1	1	1	1	1	1

Figure 6.2. The similarity-based distance matrix based on Jaccard

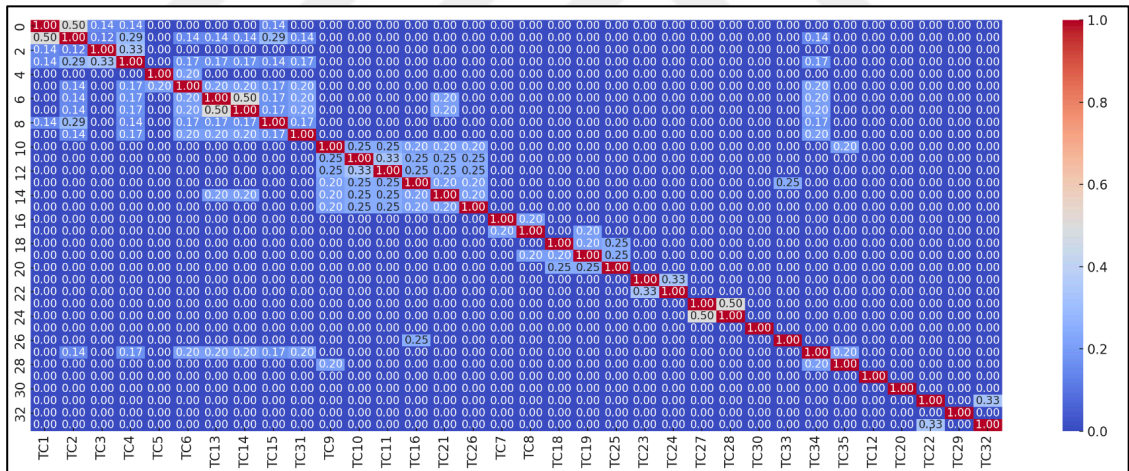


Figure 6.3. Heatmap of the Jaccard matrix

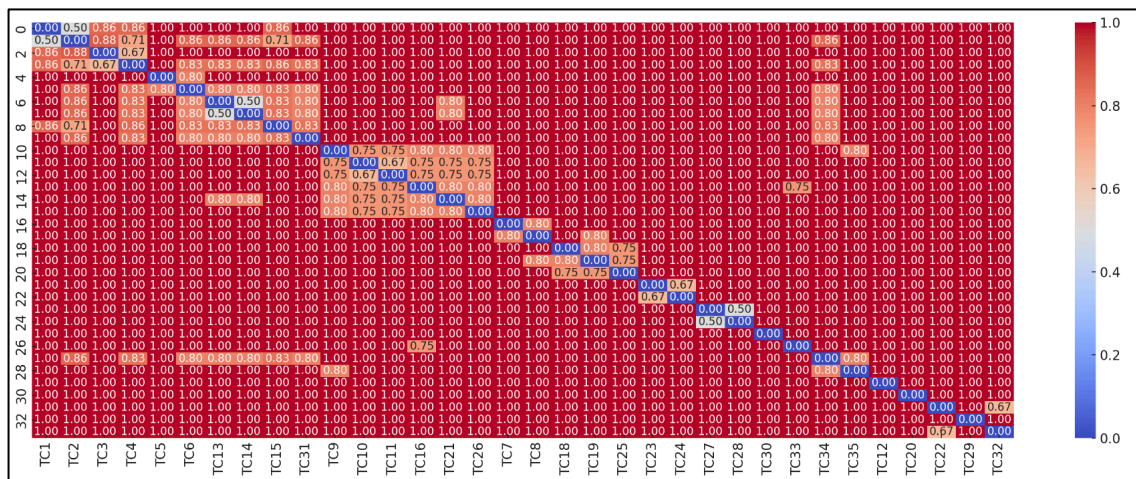


Figure 6.4. Heatmap of the Jaccard distance matrix

Each cell in the distance matrix represents the distance between two test cases. The diagonal values, such as the distance between TC1 and TC1, are set to zero, indicating that there is no distance between a test case and itself.

After that, the test cases are clustered based on their similarities (from the distance matrix).

Cluster I: Login and Sign Up Processes

TC1: User Registration with Valid Data - Sign Up

TC2: User Registration with Invalid Data - Sign Up

TC3: Enter Account Information - Sign Up

TC4: Enter Account Information - Missing Necessary Areas - Sign Up

TC5: User Login with Valid Credentials - Login

TC6: User Login with Invalid Credentials - Login

TC13: Login without Email

TC14: Login without Password

TC15: Register with Special Characters in Email

TC31: Subscribing with Invalid Email

Cluster II: Cart and Product Processes

TC9: Adding a Product to the Cart

TC10: Removing a Product from the Cart

TC11: Updating Product Quantity in Cart

TC16: Proceeding to Checkout
TC21: Checking Cart After Login
TC26: Navigating Product Details on the Cart

Cluster III: Search Operations

TC7: Product Search with Valid Term
TC8: Product Search with Invalid Term
TC18: Search with Empty Input
TC19: Searching for Out-of-Stock Items
TC25: Searching Results by Price

Cluster IV: Page Functionality

TC23: Verify Scroll Up using 'Arrow' button
TC24: Verify Scroll Up without 'Arrow' button
TC27: Verify "Contact Us" Function Works - Without Attachment
TC28: Verify "Contact Us" Function Works - With An Attachment

Cluster V: Payment and Subscription

TC30: Subscription Functionality
TC33: Entering Shipping Information
TC34: Entering Payment Information - Invalid Characters
TC35: Entering Payment Information - Valid Characters

Cluster VI: Unique

TC12: Logging Out
TC20: Product Categories
TC22: Product Brands
TC29: Account Deletion
TC32: Filtering by Product Brands

When the cluster results are evaluated by categorizing the similarity-based Jaccard distance method, it is realized that the test cases with the same or similar keywords have been clustered in the same clusters.

After this step, a TCS method was carried out with cluster-by-cluster algorithms inspired by nature. If there is a low number of test cases in a cluster, it indicates the uniqueness of these cases within the relevant test suite. Therefore, it can be decided to directly select and test these unique test cases.

6.1.2. Test Case Selection Process

Based on the test cases clustered by clusters according to the Jaccard-Distance Matrix, a TCS process was conducted cluster by cluster. Each cluster underwent a selection process according to the BA, GWO, ACO, ABO, and PSO algorithms. In the selection study, common values were used for the parameters of the algorithms.

With respect to each cluster, the objective was to select the three test cases for all algorithms.

In order to prevent the re-select of a selected test following code execution, additional condition checks were implemented. This eliminates the potential for selecting a test multiple times during the TCS process.

Table 6.3 shows the parameters used for the BA implementation:

Table 6.3. Cluster I bat algorithm parameters

Parameter	Description	Value
Seed	A seed value has been used for the random number generator variable rand. This way, the code will produce similar and consistent results each time it is run, preserving the varying random values.	42
Population Size (n)	The size of the bat population	10
Maximum Number of Iterations (maxIter)	The maximum number of iterations	3
Frequency (freq)	Frequency of bats, randomly initialized between 0 and 2	Random values between 0 and 2
Pulse Rate (r)	Pulse rate of bats, randomly initialized between 0 and 1	Random values between 0 and 1
Loudness (A)	Loudness of bats, randomly initialized between 1 and 2	Random values between 1 and 2
Positions	Initial positions of bats, randomly initialized between -5 and 5	Random values between -5 and 5

Velocities	Initial velocities of bats, set to 0	0
Fitness	Fitness values of bats, calculated using the objective function	Calculated using the objective function
Best Solution	The best solution found during optimization	The best position with the lowest fitness value

The selected test cases TC2, TC1, and TC4 are obtained, after executing the code with these parameters. Figure 6.5 shows the developed BA algorithm.

```

49     public static void updateBats() {
50         for (int t = 0; t < maxIter; t++) {
51             for (int i = 0; i < n; i++) {
52                 freq[i] = rand.nextDouble() * 2;
53                 for (int j = 0; j < 10; j++) {
54                     velocities[i][j] += (positions[i][j] - best[j]) * freq[i];
55                     positions[i][j] += velocities[i][j];
56                 }
57
58                 if (rand.nextDouble() > r[i]) {
59                     for (int j = 0; j < 10; j++) {
60                         positions[i][j] = best[j] + rand.nextDouble() - 0.5;
61                     }
62                 }
63
64                 if (rand.nextDouble() < A[i] && f(positions[i]) < f(best)) {
65                     best = positions[i].clone();
66                     r[i] *= 0.9;
67                     A[i] *= 0.9;
68                 }
69             }
70         }
71     }

```

BatAlgorithm x

/Users/baturaycaglasin/.sdkman/candidates/java/17.0.7-oracle/bin/java -javaagent:/Appli

Selected test cases:

TC2: User Registration with Invalid Data - Sign Up

TC1: User Registration with Valid Data - Sign Up

TC4: Enter Account Information - Missing Necessary Areas - Sign Up

Best solution found: [1.300744482263398, 0.9470417287682118, 0.5214104279791312, -1.181

Fitness value: 15.904271433472895

Figure 6.5. Bat algorithm Java code

Table 6.4 shows the parameters used for the GWO implementation:

Table 6.4. Cluster I grey wolf algorithm parameters

Parameter	Description	Value
Seed	A seed value has been used for the random number generator variable rand. This way, the code will produce similar and consistent results each time it is run, preserving the varying random values.	42
Population Size (n)	The size of the wolf population	10
Maximum Number of Iterations (maxIter)	The maximum number of iterations	3
Positions	Initial positions of wolves, randomly initialized between -10 and 10	Random values between -10 and 10
Velocities	Initial velocities of wolves, set to 0	0
Fitness	Fitness values of wolves, calculated using the objective function	Calculated using the objective function
Alpha Position	The best position found during optimization (alpha wolf)	The best position with the lowest fitness value
Beta Position	The second best position found during optimization (beta wolf)	The second best position with the second lowest fitness value
Delta Position	The third best position found during optimization (delta wolf)	The third best position with the third lowest fitness value
Coefficient (a)	Coefficient decreasing from 2 to 0	2 to 0
Coefficient (A)	Coefficient vector A, calculated using the formula $A = 2 * a * r1 - a$	Random values between -2a and 2a
Coefficient (C)	Coefficient vector C, calculated using the formula $C = 2 * r2$	Random values between 0 and 2

The selected test cases TC2, TC1, and TC5 are obtained, after executing the code with these parameters. Figure 6.6 shows the developed GWO algorithm.

```

51     public static void updateWolves(int numTestCases) {
52         for (int t = 0; t < maxIter; t++) {
53             a = 2 - 2 * ((double) t / maxIter);
54             for (int i = 0; i < n; i++) {
55                 for (int j = 0; j < numTestCases; j++) {
56                     double r1 = rand.nextDouble();
57                     double r2 = rand.nextDouble();
58                     double A1 = 2 * a * r1 - a;
59                     double C1 = 2 * r2;
60
61                     double D_alpha = Math.abs(C1 * alpha[j] - positions[i][j]);
62                     double X1 = alpha[j] - A1 * D_alpha;
63
64                     r1 = rand.nextDouble();
65                     r2 = rand.nextDouble();
66                     double A2 = 2 * a * r1 - a;
67                     double C2 = 2 * r2;
68
69                     double D_beta = Math.abs(C2 * beta[j] - positions[i][j]);
70                     double X2 = beta[j] - A2 * D_beta;

```

GreyWolfOptimizer x

/Users/baturaycaglasin/.sdkman/candidates/java/17.0.7-oracle/bin/java -javaagent:/Applicati

Selected test cases:

TC2: User Registration with Invalid Data - Sign Up

TC1: User Registration with Valid Data - Sign Up

TC5: User Login with Valid Credentials - Login

Alpha position: [1.5067040334249615, 0.6338326167571243, -1.8870740459264255, 3.25165222500

Fitness value: 29.119514034801334

Figure 6.6. Grey wolf algorithm Java code

Table 6.5 shows the parameters used for the ACO implementation:

Table 6.5. Cluster I ant colony algorithm parameters

Parameter	Description	Value
Seed	A seed value has been used for the random number generator variable rand. This way, the code will produce similar and consistent results each time it is run, preserving the varying random values.	42
Population Size (n)	The number of ants in the population	10
Maximum Number of Iterations (maxIter)	The maximum number of iterations	3
Evaporation Rate	The rate at which pheromone evaporates	0.5
Alpha (α)	Pheromone importance	1
Beta (β)	Heuristic importance	2

Pheromone Matrix	Initial pheromone values set to 1.0	1.0
Heuristic Matrix	Heuristic information set to $1.0 / (i - j + 1)$	$1.0 / (i - j + 1)$

The selected test cases TC1, TC2, and TC4 are obtained, after executing the code with these parameters. Figure 6.7 shows the developed ACO algorithm.

```

42     public static void updatePheromone() {
43         for (int t = 0; t < maxIter; t++) {
44             for (int i = 0; i < n; i++) {
45                 for (int j = 0; j < n; j++) {
46                     pheromone[i][j] *= 0.9; // Evaporation
47                     pheromone[i][j] += fitness[i] / 100.0; // Deposit
48                 }
49             }
50             for (int i = 0; i < n; i++) {
51                 int a = rand.nextInt(n);
52                 int b = rand.nextInt(n);
53                 if (pheromone[a][b] > rand.nextDouble()) {
54                     double[] temp = positions[a];
55                     positions[a] = positions[b];
56                     positions[b] = temp;
57                 }
58             }
59             for (int i = 0; i < n; i++) {
60                 fitness[i] = f(positions[i]);
61                 if (fitness[i] < f(best)) {
62                     best = positions[i].clone();
63                 }
64             }
65         }
66     }

```

AntColonyOptimization x

:/users/daturaycaglasin/.sakman/candidates/java/17.0.-oracle/bin/java -javaagent

Selected test cases:

TC1: User Registration with Valid Data - Sign Up

TC2: User Registration with Invalid Data - Sign Up

TC4: Enter Account Information - Missing Necessary Areas - Sign Up

Best solution found: [-0.7902829336897135, 1.3286979135327908, 1.99858645067172]

Fitness value: 38.07229046154121

Figure 6.7. Ant colony algorithm Java code

Table 6.6 shows the parameters used for the ABO implementation:

Table 6.6. Cluster I African buffalo algorithm parameters

Parameter	Description	Value
Seed	A seed value has been used for the random number generator variable rand. This way, the code will produce similar and consistent results each time it is run, preserving the varying random values.	42
Population Size (n)	The size of the buffalo population	10
Maximum Number of Iterations (maxIter)	The maximum number of iterations	3
Frequency (freq)	Frequency of buffaloes, randomly initialized between 0 and 2	Random values between 0 and 2
Pulse Rate (r)	Pulse rate of buffaloes, randomly initialized between 0 and 1	Random values between 0 and 1
Loudness (A)	Loudness of buffaloes, randomly initialized between 1 and 2	Random values between 1 and 2
Positions	Initial positions of buffaloes, randomly initialized between -5 and 5	Random values between -5 and 5
Velocities	Initial velocities of buffaloes, set to 0	0
Fitness	Fitness values of buffaloes, calculated using the objective function	Calculated using the objective function
Best Solution	The best solution found during optimization	The best position with the lowest fitness value

The selected test cases TC1, TC4 and TC2 are obtained, after executing the code with these parameters. Figure 6.8 shows the developed ABO algorithm.

```

25 public static void initialize() {
26     for (int i = 0; i < n; i++) {
27         for (int j = 0; j < 10; j++) {
28             positions[i][j] = rand.nextDouble() * 10 - 5;
29         }
30         fitness[i] = f(positions[i]);
31     }
32     best = positions[0].clone();
33     for (int i = 1; i < n; i++) {
34         if (f(positions[i]) < f(best)) {
35             best = positions[i].clone();
36         }
37     }
38 }
39 1 usage
40 public static void updateBuffalos() {
41     for (int t = 0; t < maxIter; t++) {
42         for (int i = 0; i < n; i++) {
43             for (int j = 0; j < 10; j++) {
44                 positions[i][j] += (rand.nextDouble() - 0.5) * 2;
45             }
46             fitness[i] = f(positions[i]);
47             if (fitness[i] < f(best)) {

```

AfricanBuffaloOptimization x

/users/paturaycaglasin/.sokman/candidates/java/1/.0./-oracle/bin/java -javaagent:/Applications/...

Selected test cases:

TC1: User Registration with Valid Data - Sign Up

TC4: Enter Account Information - Missing Necessary Areas - Sign Up

TC2: User Registration with Invalid Data - Sign Up

Best solution found: [0.5277844942243382, -3.158886976982598, -3.1339748975288657, -0.3360681391]

Fitness value: 33.55941851019855

Figure 6.8. African buffalo algorithm Java code

Table 6.7 shows the parameters used for the PSO implementation:

Table 6.7. Cluster I particle swarm algorithm parameters

Parameter	Description	Value
Seed	A seed value has been used for the random number generator variable rand. This way, the code will produce similar and consistent results each time it is run, preserving the varying random values.	42
Population Size (n)	The size of the particle population	10
Maximum Number of Iterations (maxIter)	The maximum number of iterations	3
Inertia Weight (w)	Inertia weight used to regulate the impact of prior velocity	0.5
Cognitive Coefficient (c1)	Cognitive coefficient used to adjust the attractiveness towards the particle's best-known position	1.5
Social Coefficient (c2)	Social coefficient used to adjust the attractiveness towards the global best-known position	1.5
Positions	Initial positions of particles, randomly initialized between -10 and 10	Random values between -10 and 10

Velocities	Initial velocities of particles, set to 0	0
Fitness	Fitness values of particles, calculated using the objective function	Calculated using the objective function
Personal Best	The personal best positions found during optimization	Calculated during optimization
Global Best	The global best position found during optimization	Calculated during optimization

The selected test cases TC1, TC3 and TC4 are obtained, after executing the code with these parameters. Figure 6.9 shows the developed PSO algorithm.

```

1 usage
47 public static void updateParticles() {
48     for (int t = 0; t < maxIter; t++) {
49         for (int i = 0; i < n; i++) {
50             for (int j = 0; j < 10; j++) {
51                 velocities[i][j] = w * velocities[i][j] +
52                     c1 * rand.nextDouble() * (pBest[i][j] - positions[i][j]) +
53                     c2 * rand.nextDouble() * (gBest[j] - positions[i][j]);
54                 positions[i][j] += velocities[i][j];
55             }
56
57             fitness[i] = f(positions[i]);
58             if (fitness[i] < pBestFitness[i]) {
59                 pBest[i] = positions[i].clone();
60                 pBestFitness[i] = fitness[i];
61             }
62             if (fitness[i] < gBestFitness) {
63                 gBest = positions[i].clone();
64                 gBestFitness = fitness[i];
65             }
66         }
67     }
68 }

```

ParticleSwarmOptimization x

/users/daturaycaglasin/.sakman/candidates/java/1/.0.7-oracle/bin/java -javaagent:/Applications/

Selected test cases:

TC1: User Registration with Valid Data - Sign Up

TC3: Enter Account Information - Sign Up

TC4: Enter Account Information - Missing Necessary Areas - Sign Up

Best solution found: [-0.5882980190388212, -0.29496319954426076, 1.289761230125218, 0.669706792

Fitness value: 8.993575227929142

Figure 6.9. Particle swarm algorithm Java code

Other clusters were also subjected to the TCS algorithms under the same cluster I headings, and the resulting selections are shown in Table 6.7.

Each algorithm's parameters are the same values as cluster I. Only the population size parameter is different from each other because the population size parameter should be equal to the number of test cases for each cluster.

No selection was made for cluster VI. Test cases within this cluster do not have similarities with each other. Test cases in such clusters will need to be evaluated individually during regression testing since these tests are unique. Making a selection in this case could compromise the quality of the tests performed.

Table 6.8. Results of the other clusters

Cluster Name	TCS Algorithm	Selected TCs
Cluster II	BA	TC16, TC26 , TC9
Cluster II	GWO	TC21, TC9, TC26
Cluster II	ACO	TC9, TC10, TC11
Cluster II	ABO	TC9, TC10, TC11
Cluster II	PSO	TC10, TC9, TC21
Cluster III	BA	TC18,TC8,TC25
Cluster III	GWO	TC25, TC8, TC18
Cluster III	ACO	TC18,TC8, TC25
Cluster III	ABO	TC8, TC7, TC19
Cluster III	PSO	TC8, TC7, TC19
Cluster IV	BA	TC24, TC27, TC28
Cluster IV	GWO	TC24, TC28, TC23
Cluster IV	ACO	TC28, TC23, TC24
Cluster IV	ABO	TC24, TC27, TC23
Cluster IV	PSO	TC23, TC24, TC27
Cluster V	BA	TC33, TC34, TC35
Cluster V	GWO	TC33, TC35, TC30
Cluster V	ACO	TC35, TC30, TC33
Cluster V	ABO	TC33, TC34, TC30
Cluster V	PSO	TC30, TC33, TC34

6.2. DIRECT NATURE-INSPIRED SELECTION

In clustered nature-inspired selection, six clusters were identified within the case study. The 6th cluster, named "others," contains test cases that do not belong to any specific group. It was deemed appropriate to select three test cases per cluster. Thus, for clustered nature-inspired selection, a total of 20 test cases were selected out of the 35 cases, 15 from the clusters and five from the "others" cluster. To maintain consistency with the results of clustered nature-inspired selection, it was also deemed appropriate to select 20 test cases out of the 35 cases in direct nature-inspired selection. The test cases were selected according to the evaluation. This issue will be examined in detail under Section 8.2. The number of selected test cases may vary depending on the data size and the desired or targeted number of tests.

The test cases, algorithms, parameters, and code blocks are identical to those that are specified and described in clustered nature-inspired selection.

The sole distinctions are that the maximum iteration number is increased from three to 200, and the population size in the algorithms is set to 20 (the total number of test cases). It was discovered that when the population size increased but the maximum number of iterations was set too low, the algorithm was unable to choose the necessary number of test cases. The maximum iteration parameter was increased from three to 200 to ensure proper selection. The other parameter selections remain unchanged.

6.3. SYSTEM PROTOTYPE FOR DIRECT NATURE-INSPIRED SELECTION

The developed system (hereafter referred to as "OptiTest," a name derived from the combination of the words "optimization" and "test") has been designed for the direct nature-inspired selection, with the primary goal of automatically selecting test cases based on the selected algorithm. Unlike clustered nature-inspired selection, the OptiTest does not contain a similarity-based clustering process. The OptiTest is developed by the author.

The reason OptiTest is included in this study and developed by the author is to demonstrate that TCS processes can be performed through a set of tools with a visual interface. Through this interface, test engineers can quickly select test cases using the nature-inspired algorithms

applied in the study, especially within the limited time frames of regression testing. It is an assisting tool that does not impact the success criteria.

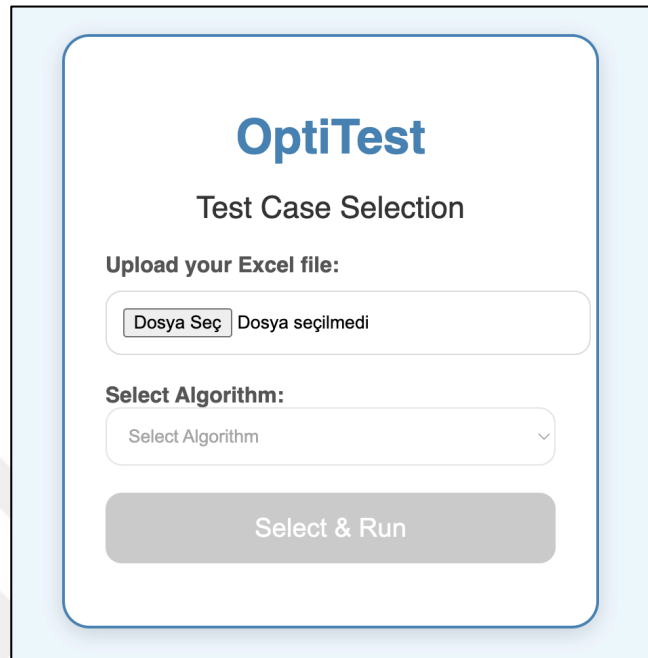
The image shows a web interface for 'OptiTest' titled 'Test Case Selection'. It features a light blue background with a white rounded rectangle in the center. Inside the rectangle, the title 'OptiTest' is in blue, followed by 'Test Case Selection' in black. Below this, the text 'Upload your Excel file:' is followed by a file selection button labeled 'Dosya Seç' and a status indicator 'Dosya seçilmedi'. Underneath, the text 'Select Algorithm:' is followed by a dropdown menu showing 'Select Algorithm' with a downward arrow. At the bottom of the white rectangle is a large grey button labeled 'Select & Run'. The entire interface is set against a background with a large, faint 'X' watermark.

Figure 6.10. Dashboard of OptiTest

Figure 6.10 shows the OptiTest's UI. The file format and extension for upload to OptiTest are critically important. The file must be an Excel document, according to the usual test case format.

After the user uploads the Excel file with the test cases to OptiTest, the user chooses an algorithm for TCS. Upon selecting the algorithm and clicking the select & run button, the designated algorithm executes in the background, and shortly thereafter, the user is presented with the requisite test cases for selection.

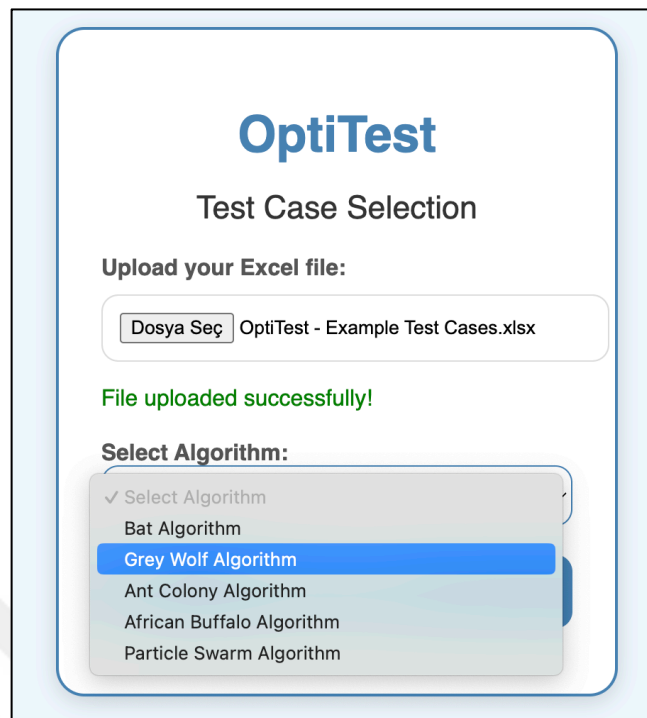


Figure 6.11. Algorithm selection in OptiTest

OptiTest's front-end is developed using HTML, CSS, and JavaScript, while the backend is developed in Java. Maven and Spring Boot have been used as the frameworks for back-end integration.

A Spring project was created using the Start.sprint.io website, and the following dependencies were added:

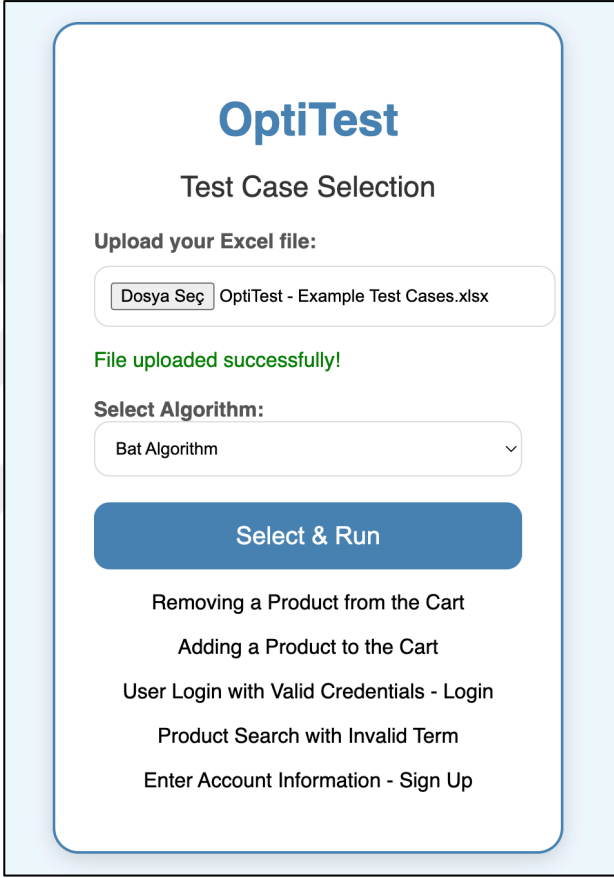
- Spring Web (to create REST APIs)
- Spring Boot DevTools (for automatic refresh during development)
- Apache POI (to process Excel files)

The number of test cases uploaded may vary. For the OptiTest the parameters':

- Total number of test cases: This refers to the number of test cases uploaded to OptiTest by the user.
- Number of test cases to be selected: This is 50% of the total number of uploaded test cases.
- Minimum selection rule: If the user uploads a small number of test cases (e.g., five test cases), a minimum of three cases will be selected.

In the example presented in Figure 6.11, the user has uploaded 10 test cases in Excel format to OptiTest. According to the selected Bat algorithm, five test cases were chosen and presented to the user.

Thus, the user can proceed, knowing which tests to select and execute based on the BA. OptiTest operates efficiently to deliver accurate results in a short time and plays a supportive role, particularly for test engineers during regression testing periods.



OptiTest

Test Case Selection

Upload your Excel file:

Dosya Seç OptiTest - Example Test Cases.xlsx

File uploaded successfully!

Select Algorithm:

Bat Algorithm

Select & Run

Removing a Product from the Cart

Adding a Product to the Cart

User Login with Valid Credentials - Login

Product Search with Invalid Term

Enter Account Information - Sign Up

Figure 6.12. Example output in OptiTest

In this example, it would be efficient for the user to select the following test cases based on the BA, according to OptiTest's recommendations as Figure 6.12 shows.

7. RESULTS

Following the evaluation presented in Section 6, different results emerged for both clustered nature-inspired selection and direct nature-inspired selection. These differing results have been examined under separate headings as 7.1. and 7.2. The challenges encountered during the execution of direct nature-inspired selection have been detailed under individual headings. Furthermore, in Section 7.3, where the results are assessed according to the study's success factor, APXC, the outcomes are evaluated based on the APXC criterion.

7.1. RESULTS FOR THE CLUSTERED NATURE-INSPIRED SELECTION

Generally, each algorithm seeks the optimum solution within a specified maximum number of iterations. This solution signifies the lowest fitness value for a certain optimization issue. The algorithms attain new places by modifying certain parameters during the iterations, including the frequency and velocity of the bats in the BA.

The algorithms' formulas are used to calculate the fitness values for these locations. [57,37,72,56,39]. The minimal fitness values are characteristics of the optimum solutions. A parameter, such as the total error rate or test case execution time, is decreased by the fitness value.

Table 7.1 shows the best fitness value and the range in which the best value is determined after running the algorithm codes. The best solution found column indicates the coordinates or values of the best solution found during the execution of each algorithm.

Table 7.1. Results of the clustered nature-inspired selection

Cluster	Algorithm	Selected TCs	Best Fitness Value	Best Solution Found
Cluster I	BA	TC2, TC1, TC4	15.90427143 3472895	[1.300744482263398, 0.9470417287682118, 0.5214104279791312, - 1.1810953751935198, 2.6105191651489754, 1.626399306059966, - 1.2451178039592414, - 0.26976778316694083, - 0.7198099848060389, 0.21768701945001312]
Cluster I	GWO	TC2, TC1, TC5	29.11951403 4801334	[1.5067040334249615, 0.6338326167571243, - 1.8870740459264255, 3.251652225004079, 0.4457720416438919, - 0.04241800140213451, - 2.859570584885431, - 1.5709663758512973, 0.6331241964201791, - 1.0329012659347547]
Cluster I	ACO	TC1, TC2, TC4	38.07229046 154121	[-0.7902829336897135, 1.3286979135327908, 1.9985864506717208, - 1.8467154994232704, 0.7162030552997667, - 1.2898771031623912, 3.7181459596483872, 3.0573094266199803, 1.1621368503517875, - 1.2564064391099574]
Cluster I	ABO	TC1, TC4, TC2	33.55941851 019855	[0.5277844942243382, - 3.158886976982598, - 3.1339748975288657, - 0.33606813917579736, 0.6348956904819343, - 1.1718493026299373, 3.088001386895743, - 0.4180780627920937, 0.7460986273042594, 1.150663431418753]
Cluster I	PSO	TC1, TC3, TC4	8.993575227 929142	[-0.5882980190388212, - 0.29496319954426076, 1.289761230125218, 0.6697067924858153, 1.5857005371059973, - 0.9649192014523131, 0.5014479251874915, 0.6556875131399083, 1.2673589276803499, 0.8458109210455116]
Cluster II	BA	TC16, TC26, TC9	11.19477299 343767	[-0.6355920907191991, - 0.2777764548910766, - 0.654631163619285, - 0.10588782279679787,

				0.35237270111062835, 3.1858617698944234]
Cluster II	GWO	TC21, TC9, TC26	9.223094938 855091	[-2.6495459884762798, 0.6568348464856453, - 1.2440554150838075, - 0.04078686868665297, 0.46708690687556026, 0.06372876285744684]
Cluster II	ACO	TC9, TC10,TC11	36.38164389 386044	[-0.9537725815134105, 0.8811046590067692, 1.6647536111931442, - 1.7739637496504148, 0.9273706585531218, - 0.9387665514110901, 3.5080381951259816, 3.0626535345906887, 1.637144434001348, - 1.6338741811587487]
Cluster II	ABO	TC9, TC10,TC11	19.06234750 189533	[-0.7790241159106701, 1.7646768133020339, 3.2842597161812797, 1.2823579320038223, - 0.07178118031653558, - 0.7720323658862469, - 0.9022298650383382, - 0.8688719447159176, 0.16698213977300141, - 0.8441292505929443]
Cluster II	PSO	TC10,TC9,TC21	17.74540482 783187	[1.1248515617742085, 2.264319615081905, - 0.6816821766276147, - 2.531552576947539, - 0.827936580981212, - 0.47140918460114734, - 0.5079681475746576, 0.7379734233735152, - 1.2006256528485961, 1.1522488709009382]
Cluster III	BA	TC18,TC8,TC25	9.105737744 916002	[2.179858217617402, 1.1833307702529134, - 1.3172554216579493, - 0.8257682265566894, 0.7325497759856927]
Cluster III	GWO	TC25,TC8, TC18	42.38536524 404361	[4.328284092963739, - 4.088735581901298, 1.6687908640095908, - 1.4157471339113286, 1.464363513938708]
Cluster III	ACO	TC18,TC8, TC25	24.73297127 5714696	[2.275636800328681, 1.8322347175984541, - 1.9128054466734024, - 2.2292150992586333, 2.751206959271756]
Cluster III	ABO	TC8,TC7, TC19	4.963518920 6758215	[-0.49119071956833865, 1.098668459078183, - 0.06711343940132518, 0.38650488145021755, 1.8333815690083197]

Cluster III	PSO	TC8,TC7, TC19	10.14701409 1806072	[1.812200188357036, - 0.7433191415987317, - 0.23069984260156406, - 2.2746024311825606, 1.0408566594587931]
Cluster IV	BA	TC24,TC27, TC28	3.593350872 719433	[0.06974576187311277, 0.18762135514039602, 0.8009606116251835, 1.706384108909968]
Cluster IV	GWO	TC24,TC28, TC23	86.03701837 344181	[5.105155505453044, - 5.02757353663515, - 5.8065694074562115, - 0.9907884177279435]
Cluster IV	ACO	TC28,TC23, TC24	20.01795430 7745516	[3.2596587188782102, 0.8742738178629565, - 1.9128054466734024, - 2.2292150992586333]
Cluster IV	ABO	TC24,TC27, TC23	2.095323329 550548	[-0.4903496491990076, 0.5507215382743629, - 1.2035510019650835, 0.3210160806416416]
Cluster IV	PSO	TC23,TC24, TC27	4.852992721 628342	[-0.9395279749404457, - 0.746276363637917, 0.7891088259510974, 1.6705264906053108]
Cluster V	BA	TC33,TC34, TC35	3.593350872 719433	[0.06974576187311277, 0.18762135514039602, 0.8009606116251835, 1.706384108909968]
Cluster V	GWO	TC33,TC35, TC30	86.03701837 344181	[5.105155505453044, - 5.02757353663515, - 5.8065694074562115, - 0.9907884177279435]
Cluster V	ACO	TC35,TC30, TC33	20.01795430 7745516	[3.2596587188782102, 0.8742738178629565, - 1.9128054466734024, - 2.2292150992586333]
Cluster V	ABO	TC33,TC34, TC30	2.095323329 550548	[-0.4903496491990076, 0.5507215382743629, - 1.2035510019650835, 0.3210160806416416]
Cluster V	PSO	TC30,TC33, TC34	4.852992721 628342	[-0.9395279749404457, - 0.746276363637917, 0.7891088259510974, 1.6705264906053108]

Unlike clustered nature-inspired selection, direct nature-inspired selection focuses on TCS operations using nature-inspired algorithms without any clustering of the data. The absence of data clustering, as seen in the clustered nature-inspired selection, has introduced certain challenges which are detailed in Section 7.2.1 and 7.2.2.

7.2. RESULTS FOR THE DIRECT NATURE-INSPIRED SELECTION

Table 7.2 contains the selected test cases and other relevant information for each algorithm. It is important to note that the test case IDs in the best solution found section should be adjusted by adding one due to Java's array indexing, which starts at zero. In other words, seeing “[3]” in the best solution found section corresponds to “test case ID 4”.

Table 7.2. Results of the direct nature-inspired selection

TCS Algorithm	Selected TCs	Best Fitness Value	Best Solution Found	Number of Selected Test Case
BA	TC4, TC21, TC23, TC19, TC24, TC20, TC5, TC27, TC25, TC29, TC14, TC10, TC22, TC32, TC6, TC30, TC18, TC7, TC34, TC13	595.0	[3, 20, 22, 18, 23, 19, 4, 26, 24, 28, 13, 9, 21, 31, 5, 29, 17, 6, 33, 12, 8, 0, 32, 14, 7, 34, 16, 10, 27, 25, 2, 1, 30, 11, 15]	20
GWA	TC10, TC8, TC1, TC13, TC2, TC3, TC19, TC5, TC4, TC6, TC7, TC9, TC11, TC12, TC14, TC15, TC16, TC17, TC29, TC18	297.0	[9, 7, 0, 12, 1, 2, 18, 4, 3, 5, 6, 8, 10, 11, 13, 14, 15, 16, 28, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 32, 33, 34]	20
ACO	TC21, TC6, TC16, TC9, TC19, TC20, TC14, TC4, TC7, TC5, TC15, TC18, TC34, TC24, TC31, TC10, TC1, TC25, TC17, TC12	66.47723665173352	[20, 5, 15, 8, 18, 19, 13, 3, 6, 4, 14, 17, 33, 23, 30, 9, 0, 24, 16, 11, 1, 12, 7, 31, 25, 10, 32, 34, 2, 28, 22, 27, 21, 29, 26]	20
ABO	TC12, TC1, TC25, TC6, TC5, TC4, TC13, TC17, TC11, TC16, TC31, TC3, TC26, TC24, TC20, TC21, TC27, TC15, TC10	393.0	[11, 0, 24, 5, 4, 5, 3, 12, 16, 10, 15, 5, 30, 4, 11, 2, 25, 23, 19, 0, 20, 26, 4, 10, 0, 5, 14, 11, 3, 11, 9, 30, 12, 5, 9]	19
PSO	TC3, TC7, TC10, TC8, TC2, TC4, TC12, TC26, TC6, TC1, TC15, TC16, TC5, TC14, TC29, TC30, TC9, TC18	268.0	[2, 6, 9, 6, 7, 2, 1, 2, 2, 3, 11, 25, 5, 7, 6, 2, 0, 3, 14, 15, 4, 6, 13, 6, 7, 28, 29, 4, 0, 7, 3, 8, 3, 5, 17]	18

7.2.1. Stochastic to Deterministic Transition

Direct nature-inspired selection specifically includes 35 test cases within the scope of this case study. The maximum number of iterations for the algorithms has been increased from three to 200. This increase was necessary due to the occurrence of array index out-of-bounds errors, even with the higher number of iterations. As the population (number of data points or test cases in the study) increases, the management and application of the algorithms become increasingly difficult.

This difficulty has brought about certain issues. Unlike in clustered nature-inspired selection, the best solution positions had to be determined in a more deterministic manner. In clustered nature-inspired selection, the best solution positions were determined in a more stochastic manner. This was feasible because the data set was smaller and more manageable, allowing the nature-inspired algorithms to explore various solutions more freely.

Clustered nature-inspired selection result (BA):

Selected test cases:

- TC2: User Registration with Invalid Data - Sign Up
- TC1: User Registration with Valid Data - Sign Up
- TC4: Enter Account Information - Missing Necessary Areas - Sign Up

Best solution found: [1.300744482263398, 0.9470417287682118, 0.5214104279791312, -1.1810953751935198, 2.6105191651489754, 1.626399306059966, -1.2451178039592414, -0.26976778316694083, -0.7198099848060389, 0.21768701945001312]

Fitness value: 15.904271433472895

However, as previously indicated, controlling the algorithms got more difficult in direct nature-inspired selection as the number of test cases expanded dramatically. The implementation of a purely stochastic technique becomes challenging due to the bigger data set's new complications. In order to guarantee that the algorithms could manage the higher volume of data, a more deterministic method was used to determine the optimal solutions.

Direct nature-inspired selection results (BA):

Selected test cases:

- TC4: Enter Account Information - Missing Necessary Areas - Sign Up
- TC21: Checking Cart After Login
- TC23: Verify Scroll Up using 'Arrow' button
- TC19: Searching for Out-of-Stock Items
- TC24: Verify Scroll Up without 'Arrow' button
- TC20: Product Categories
- TC5: User Login with Valid Credentials - Login
- TC27: Verify 'Contact Us' Function Works - Without Attachment
- TC25: Searching Results by Price
- TC29: Account Deletion
- TC14: Login without Password
- TC10: Removing a Product from the Cart
- TC22: Product Brands
- TC32: Filtering by Product Brands
- TC6: User Login with Invalid Credentials - Login
- TC30: Subscription Functionality
- TC18: Search with Empty Input
- TC7: Product Search with Valid Term
- TC34: Entering Payment Information - Invalid Characters
- TC13: Login without Email

Best solution found: [3, 20, 22, 18, 23, 19, 4, 26, 24, 28, 13, 9, 21, 31, 5, 29, 17, 6, 33, 12, 8, 0, 32, 14, 7, 34, 16, 10, 27, 25, 2, 1, 30, 11, 15]

Fitness value: 595.0

If the goal was to maintain the methods' dependability and accuracy, a change from a stochastic to a more deterministic strategy was required as the data set grew and the complexity of the solution space increased. This change is crucial for achieving consistent outcomes for this study and properly managing the extensive data collection.

7.2.2. Repeated Test Case Situations

There were various difficulties in the given ABO and PSO algorithm implementations. Though both techniques apply the same ideas and guidelines as clustered nature-inspired selection, repeated case selections were noted during execution. This repeating affected the intended choice of test cases.

Particle velocities and locations are continuously updated in the PSO algorithm. Nevertheless, the algorithm might find the ideal solution difficulty without appropriate changes to these velocities and placements. Analogous to this, the ABO method depends on random initialization for coordinates and velocities; so, as the number of iterations rises, obtaining a stable solution is progressively difficult, especially in view of increasing population size.

Under comparable population sizes and repetition limits, the ACO and other algorithms did not present this difficulty. Using pheromones, chemical markers left on solution routes during the search for ideal test cases, ACO achieves stability. Stronger pheromone trails from more successful solutions increase the probability of these paths being chosen in next searches. This technique lets ACO concentrate on right answers over time, hence strengthening stability. The good reinforcement of successful answers enables the algorithm to concentrate more regularly on certain results. ACO preserves effective routes from the past even as it keeps looking for fresh ideas. Both local and global optimal solutions are sought for by ABO and PSO algorithms. These algorithms by their very nature repeat like or identical solutions. In Figure 7.1, repeating test cases for ABO can be observed.

ABO	TC12, TC1, TC25, TC6, TC5, TC4, TC13, TC17, TC11, TC16, TC31, TC3, TC26, TC24, TC20, TC21, TC27, TC15, TC10	393.0	[11] 0, 24, 5, 4, 5, 3, 12, 16, 10, 15, 5, 30, 4, [11] 2, 25, 23, 19, 0, 20, 26, 4, 10, 0, 5, 14, [11] 3, [11] 9, 30, 12, 5, 9]	19
-----	--	-------	--	----

Figure 7.1. Repeated test cases in African buffalo optimization

7.3. RESULT BY SUCCESS CRITERIA

The success criterion for this study has been defined as APXC, a time-based metric. As previously mentioned in Section 4.3, the APXC ratio is calculated by dividing the total duration of test cases where TCS occurs by the total duration of test cases without TCS. This measure provides insight into the effectiveness of TCS in terms of time performance.

Table 7.3 shows the estimated durations of the test cases used in the evaluation. Effort was made to ensure these durations are not only consistent but also realistic. Any potential delays or data issues that may occur during the application process have been considered outside the scope of this study, as have edge cases like testing boundary values. (such as entering eleven characters or just one character in a 10-character password field or using special characters in fields where they are not allowed).

Table 7.3. Evaluation of the success criteria

Test Case ID	Test Name	Estimated Test Duration (Minute)
TC1	User Registration with Valid Data - Sign Up	2
TC2	User Registration with Invalid Data - Sign Up	2
TC3	Enter Account Information - Sign Up	3
TC4	Enter Account Information - Missing Necessary Areas - Sign Up	2
TC5	User Login with Valid Credentials - Login	2
TC6	User Login with Invalid Credentials - Login	2
TC7	Product Search with Valid Term	3
TC8	Product Search with Invalid Term	3
TC9	Adding a Product to the Cart	4
TC10	Removing a Product from the Cart	3
TC11	Updating Product Quantity in Cart	4
TC12	Logging Out	1

TC13	Login without Email	1
TC14	Login without Password	1
TC15	Register with Special Characters in Email	2
TC16	Proceeding to Checkout	4
TC17	Checkout with Empty Cart	2
TC18	Search with Empty Input	3
TC19	Searching for Out-of-Stock Items	4
TC20	Product Categories	2
TC21	Checking Cart After Login	2
TC22	Product Brands	3
TC23	Verify Scroll Up using 'Arrow' button	2
TC24	Verify Scroll Up without 'Arrow' button	2
TC25	Searching Results by Price	3
TC26	Navigating Product Details on the Cart	3
TC27	Verify "Contact Us" Function Works - Without Attachment	4
TC28	Verify "Contact Us" Function Works - With An Attachment	5
TC29	Account Deletion	3
TC30	Subscription Functionality	3
TC31	Subscribing with Invalid Email	4
TC32	Filtering by Product Brands	3
TC33	Entering Shipping Information	4
TC34	Entering Payment Information - Invalid Characters	3
TC35	Entering Payment Information - Valid Characters	3

These data yield the following results for the APXC success criterion. For clustered nature-inspired selection, an additional 12 minutes have been added to the total algorithm durations. This is because the test cases in the "unique" cluster, referred to as "cluster VI," were not

subjected to any selection process (TC12, 20, 22, 29, and 32). To ensure the results are more consistent and fair, the durations of these test cases have also been included in the algorithm calculations. This situation does not apply to direct nature-inspired selection, as these test cases have already undergone the selection procedures.

Table 7.4. Execution durations by the approaches and algorithms

Clustered nature-inspired selection – Cluster I - BA	6 Minutes
Clustered nature-inspired selection – Cluster I - GWO	6 Minutes
Clustered nature-inspired selection – Cluster I - ACO	6 Minutes
Clustered nature-inspired selection – Cluster I - ABO	6 Minutes
Clustered nature-inspired selection – Cluster I - PSO	7 Minutes
Clustered nature-inspired selection – Cluster II - BA	11 Minutes
Clustered nature-inspired selection – Cluster II - GWO	9 Minutes
Clustered nature-inspired selection – Cluster II - ACO	11 Minutes
Clustered nature-inspired selection – Cluster II - ABO	11 Minutes
Clustered nature-inspired selection – Cluster II - PSO	9 Minutes
Clustered nature-inspired selection – Cluster III - BA	9 Minutes
Clustered nature-inspired selection – Cluster III - GWO	9 Minutes
Clustered nature-inspired selection – Cluster III - ACO	9 Minutes
Clustered nature-inspired selection – Cluster III - ABO	10 Minutes
Clustered nature-inspired selection – Cluster III - PSO	10 Minutes
Clustered nature-inspired selection – Cluster IV - BA	11 Minutes
Clustered nature-inspired selection – Cluster IV - GWO	9 Minutes
Clustered nature-inspired selection – Cluster IV - ACO	9 Minutes
Clustered nature-inspired selection – Cluster IV - ABO	8 Minutes
Clustered nature-inspired selection – Cluster IV - PSO	8 Minutes
Clustered nature-inspired selection – Cluster V - BA	10 Minutes

Clustered nature-inspired selection – Cluster V - GWO	10 Minutes
Clustered nature-inspired selection – Cluster V - ACO	10 Minutes
Clustered nature-inspired selection – Cluster V - ABO	10 Minutes
Clustered nature-inspired selection – Cluster V - PSO	10 Minutes
Clustered nature-inspired selection – BA (All Clusters)	47 Minutes + 12 Minutes = 59 Minutes
Clustered nature-inspired selection – GWO (All Clusters)	43 Minutes + 12 Minutes = 55 Minutes
Clustered nature-inspired selection – ACO (All Clusters)	45 Minutes + 12 Minutes = 57 Minutes
Clustered nature-inspired selection – ABO (All Clusters)	45 Minutes + 12 Minutes = 57 Minutes
Clustered nature-inspired selection – PSO (All Clusters)	44 Minutes + 12 Minutes = 56 Minutes
Direct nature-inspired selection – BA	51 Minutes
Direct nature-inspired selection – GWO	51 Minutes
Direct nature-inspired selection – ACO	51 Minutes
Direct nature-inspired selection – ABO	48 Minutes
Direct nature-inspired selection – PSO	46 Minutes
Average testing duration time of clustered nature-inspired selection	56.8 Minutes
Average testing duration time of direct nature-inspired selection	49.4 Minutes
Total testing duration time (All Cases)	97 Minutes

Based on these data, the APXC calculations in seconds are detailed in Table 7.5 and 7.6. These comparisons have been conducted both on an approach basis and an algorithm basis.

Table 7.5. APXC evaluation of the clustered nature-inspired selection

APXC - BA	0.4845
APXC - GWO	0.4433
APXC - ACO	0.4639
APXC - ABO	0.4639
APXC - PSO	0.4536

Table 7.6. APXC evaluation of the direct nature-inspired selection

APXC - BA	0.5258
APXC - GWO	0.5258
APXC - ACO	0.5258
APXC - ABO	0.4948
APXC - PSO	0.4742

Using APXC data, two different approaches have been investigated for evaluating algorithm performance. The APXC data in clustered nature-inspired selection demonstrated that the BA selected the test cases with the maximum efficiency, with an APXC score of 0.4845. However, the GWO showed the lowest efficiency, with an APXC value of 0.4433, indicating less effective performance in the TCS.

The BA, GWO, and ACO algorithms had the greatest APXC value of 0.5258, which suggests superior performance in this method, according to recalculated APXC values for direct nature-inspired selection. In contrast, the APXC values of ABO and PSO were similarly lower, at 0.4948 and 0.4742. All things considered, direct nature-inspired selection showed better APXC values than the other methods, suggesting a more effective TCS and execution plan.

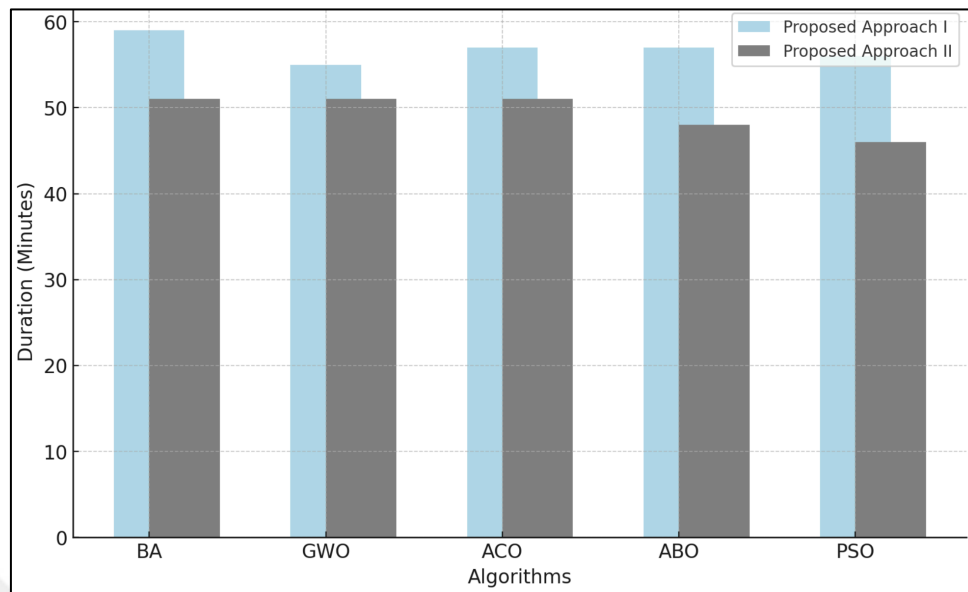


Figure 7.2. Comparison of the algorithms by durations

Figure 7.2 contains a graph displaying the durations between the algorithms. Accordingly, the light blue bars represent the durations after the clustered nature-inspired selection (approach I) was applied, while the gray bars represent the durations after the direct nature-inspired selection (approach II) was applied. Based on this, the durations in clustered nature-inspired selection are longer compared to those in direct nature-inspired selection. Specifically, it is noted that BA, GWO, and ACO have consistently demonstrated higher efficacy in both approaches, highlighting their success in TCS optimization and running time reduction. It is also acknowledged that direct nature-inspired selection is a more efficient method for improving algorithm performance in controlling test durations.

8. DISCUSSION

Efforts have been made to optimize the use of limited time during regression periods by selecting a subset of the numerous test cases that need to be tested, thereby utilizing this constrained time more efficiently.

In the study, "nature-inspired" approaches, such as the BA, GWO, ACO, ABO, and PSO algorithms, which are already established in the literature, were implemented. However, unlike previous studies, a novel approach was introduced. In this new approach, termed clustered nature-inspired selection, test cases were first clustered based on the Jaccard similarity algorithm, and then TCS was performed for each cluster.

In direct nature-inspired selection, on the other hand, all test cases were directly selected using traditional methods combined with nature-inspired algorithms. In general, it was observed that clustering in the clustered nature-inspired selection, allowed for easier management by completing operations in a shorter time. The code implementation was more straightforward. When the MaxIter parameter (the number of algorithm iterations) was set to three, no issues were observed, and results were produced according to the algorithms.

However, in direct nature-inspired selection, the large number of test cases entering the selection process without any clustering made management more challenging. When the MaxIter value was set to three, issues arose in the code, requiring an increase in this iteration parameter. Best position selections had to be made in a more deterministic manner. As the number of test cases increased, boundary errors began to occur during code execution, highlighting the need for a more deterministic approach in direct nature-inspired selection.

In clustered nature-inspired selection, the best solution output is provided numerically, and positions are obtained from arrays. Thus, a small number of test cases were uncovered using more stochastic methods. However, in direct nature-inspired selection, the best solution was directly stored as the indices of the test cases, and the results were presented as a complete numerical array. This was a more deterministic approach, but it sometimes led to the algorithm repeatedly selecting the same test cases, despite all constraints.

In summary, the differences between the algorithms in terms of success criteria in this study are not vast. Each algorithm provides not identical but similar results. Additionally, clustered nature-inspired selection is easier to manage. However, direct nature-inspired selection

produced better results in terms of the success factor, APXC. Nevertheless, it is anticipated that as the number of test cases increases, issues may arise. Additionally, the implementation of the direct nature-inspired selection system was simplified due to the absence of an intermediary layer that relies on similarity-based approaches. The user can directly perform TCS using nature-inspired algorithms. Overall, the efficiency of conducting TCS was measured at 45% higher compared to a scenario where no such selection process was implemented (based on the evaluation in this work).

8.1. LESSONS LEARNED

The divide and conquer strategy, one of the cornerstones of software engineering, might not always yield the greatest results. Based on the study, the clustered nature-inspired selection mostly aligns with this idea by grouping test cases based on how similar they are. However, direct nature-inspired selection had shorter execution times for TCS because the test cases were not clustered together. (If direct nature-inspired selection is applied, the test cases are tested in 49.4 minutes, whereas in clustered nature-inspired selection, the testing takes 56.8 minutes.) Although this result is surprising, it is thought to be a consequence of the deterministic approaches used in direct nature-inspired selection.

While the clustered nature-inspired selection yields lower results in terms of APXC compared to direct nature-inspired selection, it is generally easier to manage and the implementation of the algorithms on the test cases was more seamless. Furthermore, there are no significant differences among nature-inspired algorithms in terms of both implementation and the project's success criteria, such as APXC. While it is clear that these algorithms assist with selection, the expected level of variation has not been observed. Increasing efficiency may be achieved by combining several TCP and TCS methodologies to produce hybrid solutions. This can have a lot of benefits, particularly when it comes to using algorithms. When it comes to software testing, utilizing variants of these many methodologies might produce more varied and significant outcomes if the objective is to optimize test cases.

Nature-inspired algorithms carry significant randomness. Their implementation can be challenging, but they tend to deliver efficient results. Even though they might be challenging to implement, the results are frequently successful. As examined in detail in Section 4 and

in the existing literature, certain hybrid algorithms can yield much better results [35,37,39,59,64,65]. However, in addition to these studies, AI-assisted approaches, such as using reinforcement learning, have the potential to optimize TCS by learning from the execution history of test cases. This approach may reduce the potential issues caused by stochastic methods and improve the development of algorithms that address these challenges. The study's direct nature-inspired selection required a deterministic approach. This technique may have been stochastic, like the clustered nature-inspired selection, had AI been employed.

Another important lesson is that applying a TCS procedure may be a stress-relieving and efficiency-boosting element when pressure is high. While the evaluation study resulted in a 45% gain in efficiency, it is expected that this kind of strategy would always boost efficiency in subsequent case studies. This is due to the obvious differences between evaluating all test cases versus testing a subset of test cases. This study has demonstrated that if a test case optimization like TCS is performed, test engineers will be positively impacted, and their productivity will increase. Since test engineers will be able to complete their test suites more quickly after an optimization process, they will be able to report potential issues to the relevant developers more swiftly. As developers receive feedback on these issues sooner, they will be able to resolve them faster, allowing the relevant packages to reach post-regression environments (staging, production, or package/application releases) more quickly. This means that the updates and fixes in the application will reach customers earlier. The development of software products is a race against time, and those who act quickly will outpace their competitors, indirectly boosting the company's efficiency.

8.2. THREATS

The clustered nature-inspired selection may encounter difficulties in filtering comparable occurrences when they involve different displays or procedures, thereby complicating bug detection. The study did not focus on algorithms that were "proposed" in the literature; instead, it employed conventional algorithms. Nevertheless, the literature contains optimized versions of these traditional algorithms that frequently produce more optimal results [59]. Furthermore, the selection of test cases during the selection process may differ in real-world scenarios based on the specific situation, team, or module.

The number of test cases selected for evaluation in this study was limited, with iterations set to three for clustered nature-inspired selection and 200 for direct nature-inspired selection. This adjustment was made because, as the population size (the number of test cases) increased, it was observed that lower MaxIter values led to inaccurate selections and a failure to meet the desired number of test cases. Furthermore, nature-inspired algorithms inherently exclude certain important test cases, as documented in the literature. These excluded cases can be addressed after the regression phase. Another key consideration is that all the algorithms used in this study contain an element of randomness, which leads to different results each time they are executed in both clustered nature-inspired selection and direct nature-inspired selection. This uncertainty is a result of the fact that these algorithms generate random allocations. In order to address this issue and guarantee consistent results, the code implemented a seed value, which enabled the utilization of identical initial values and resulted in more consistent results.

9. CONCLUSION

Regression periods are managed with considerable attention by both quality assurance teams and other departments. During these periods, the test cases that need to be executed should be tested as quickly as possible to ensure efficient use of time. In this study, two different approaches were applied using nature-inspired algorithms such as BA, GWO, ACO, ABO, and PSO on sample test cases. The clustered nature-inspired selection relies on the divide-and-conquer idea, in which test cases are clustered based on their commonalities prior to selection. The alternative approach is the direct selection of test cases utilizing these algorithms, devoid of any clustering.

The results indicate that while clustering and dealing with a smaller number of test cases may offer managerial advantages, it does not always guarantee faster results. However, processing a large number of test cases through selection algorithms is generally more challenging than managing a smaller set. There are some differences between the nature-inspired algorithms in terms of the study's success factor, APXC; these differences are not significant. All algorithms provide a certain degree of efficiency.

It should be remembered that testing all available test cases without performing any test case selection would take much longer and lead to inefficient use of the limited and critical time available. If the goal is to use time efficiently, efforts should be made to streamline the process. This would enhance usability for other users and stakeholders, such as through the development of systems like OptiTest, which provide rapid and accurate outputs.

10. FUTURE WORKS

Both the approaches and the comparisons between nature-inspired algorithms in this study have been evaluated using a single success criterion, APXC. However, the literature includes a variety of other success factors. One of the future directions for this research will be to conduct comparisons using these additional success factors.

It is important to remember that there are more algorithms influenced by nature than those in this study. In addition to ACO, ABO, PSO, GWO, and BA, there are more algorithms. Including these techniques would strengthen the case for the study.

Finally, only direct nature-inspired selection is currently supported by the established system, OptiTest. However, an extra layer that operates similarly to clustered nature-inspired selection may be put into operation. In this layer, test cases from the uploaded Excel file are first grouped according to how similar they are, and TCS is then carried out using the selected algorithms.

REFERENCES

1. Denert E. Software Engineering. In: Felderer M, et al. Ernst Denert Award for Software Engineering 2019. Cham: Springer; 2020. Available from: https://doi.org/10.1007/978-3-030-58617-1_2.
2. Sommerville I. Software Engineering. 10th ed. Istanbul: Nobel; 2016. ISBN: 978-605-320-817-4.
3. Khan F, Kumar RL, Kadry S, Nam Y. The future of software engineering: Visions of 2025 and beyond. International Journal of Electrical and Computer Engineering. 2021;11(4):3443-50. Available from: <https://doi.org/10.11591/ijece.v11i4.pp3443-3450>.
4. Rastogi V. Software Development Life Cycle Models- Comparison, Consequences. International Journal of Computer Science and Information Technologies. 2015;6(1):168-72.
5. Malik S, Nigam C. A Comparative study of Different types of Models in Software Development Life Cycle. International Research Journal of Engineering and Technology. 2017. Available from: <https://www.irjet.net/archives/V4/i10/IRJET-V4I1078.pdf>.
6. Jevtic G. What is SDLC? Phases of Software Development, Models, & Best Practices. PhoenixNAP Blog. 2019. Available from: <https://phoenixnap.com/blog/software-development-life-cycle>.
7. Öztürk V. Selection of appropriate software development life cycle using fuzzy logic. Journal of Intelligent & Fuzzy Systems. 2013;25(3):797-810.
8. Hicdurmaz M. A fuzzy multi criteria decision making approach to software life cycle model selection. In: Proceedings of the Euromicro Conference on Software Engineering and Advanced Applications (SEAA). Cesme, Izmir: IEEE; 2012. p. 384.
9. Abdelsalam HME, Bao HP. A simulation-based optimization framework for product development cycle time reduction. IEEE Transactions on Engineering Management. 2006;53(1):69-85.
10. Albalawi FO, Maashi MS. Selection and optimization of software development life cycles using a genetic algorithm. Intelligent Automation and Soft Computing. 2021;28(1). Available from: <https://doi.org/10.32604/iasc.2021.015657>.

11. Gelperin D, Hetzel B. The growth of software testing. Communications of the ACM. 1988;31(6). Available from: <https://doi.org/10.1145/62959.62965>.
12. Crosby PB. Quality is free: the art of making quality certain. New York: McGraw-Hill; 1979.
13. Deming WE. Out of the crisis: quality, productivity and competitive position. Cambridge: Cambridge University Press; 1988.
14. Ramulu K, Murhtyr BVR. Importance of software quality models in software engineering. International Journal of Engineering Technologies and Management Research. 2020;5(3). Available from: <https://doi.org/10.29121/ijetmr.v5.i3.2018.192>.
15. Anwar N, Kar S. Review Paper on Various Software Testing Techniques & Strategies. Global Journal of Computer Science and Technology. 2019. Available from: <https://doi.org/10.34257/gjcstcvol19is2pg43>.
16. Taley DS, Pathak B. Comprehensive Study of Software Testing Techniques and Strategies: A Review. International Journal of Engineering Research & Technology. 2020;9(08). Available from: <http://www.ijert.org>.
17. Arumugam AK. Software Testing Techniques & New Trends. International Journal of Engineering Research & Technology. 2019;8(12). Available from: <http://www.ijert.org>.
18. Sneha K, Malle GM. Research on software testing techniques and software automation testing tools. In: 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS). IEEE; 2018. Available from: <https://doi.org/10.1109/ICECDS.2017.8389562>.
19. Baresi L, Pezze M. A Primer on Software Testing. Electronic Notes in Theoretical Computer Science. 2006;148(1):89-111. Available from: <https://doi.org/10.1016/j.entcs.2005.12.014>.
20. Verma AS, Choudhary A, Tiwari S. Software Test Case Generation Tools and Techniques: A Review. International Journal of Mathematical, Engineering and Management Sciences. 2023;8(2). Available from: <https://doi.org/10.33889/ijmems.2023.8.2.018>.
21. Kaner C. What is a Good Test Case? In: Proceedings of the Software Testing Analysis & Review Conference (STAR East). 2003.
22. Qasim M, Bibi A, Hussain SJ, Jhanjhi NZ, Humayun M, Sama NU. Test case prioritization techniques in software regression testing: An overview. Int J Adv Appl Sci. 2021;8(5). Available from: <https://doi.org/10.21833/ijaas.2021.05.012>

23. Ufuktepe E. Test case prioritization for regression testing using change impact analysis. Doctoral Thesis, Izmir Institute of Technology, 2019.
24. Ying P, Fan L. Methods for test case prioritization based on test case execution history: A systematic literature review and an experiment. Master Thesis, Blekinge Institute of Technology, 2017.
25. Chu C. Test Case Prioritization. Master Thesis, Oregon State University, 1999.
26. Omri S. Quality-aware learning to prioritize test cases. Doctoral Thesis, Karlsruhe Institute of Technology, 2022.
27. Rothermel G, Untch RH, Chu C, Harrold MJ. Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering. 2001;27(10). Available from: <https://doi.org/10.1109/32.962562>.
28. Uddin A, Anand A. Importance of Software Testing in the Process of Software Development. International Journal for Scientific Research & Development. 2019;6. Available from: www.ijssrd.com.
29. Bertolino A. Software testing research: Achievements, challenges, dreams. In: 2007 Future of Software Engineering. IEEE Computer Society; 2007. p. 85-103.
30. Burnstein I. Practical software testing: a process oriented approach. New York: Springer Science & Business Media; 2006. p. 61-95.
31. Chittimalli PK, Harrold MJ. Recomputing coverage information to assist regression testing. IEEE Transactions on Software Engineering. 2009;35:452-69.
34. Rahmani A, Ahmad S, Jalil IEA, Herawan AP. A Systematic Literature Review on Regression Test Case Prioritization. International Journal of Advanced Computer Science and Applications. 2021;12(9). Available from: <https://doi.org/10.14569/IJACSA.2021.0120929>.
35. Öztürk MM. A bat-inspired algorithm for prioritizing test cases. Vietnam Journal of Computer Science. 2017. Available from: <https://doi.org/10.1007/s40595-017-0100-x>.
36. Bajaj A, Sangwan OP. Discrete cuckoo search algorithms for test case prioritization. Applied Soft Computing. 2021. Available from: <https://doi.org/10.1016/j.asoc.2021.107584>.
37. Yue Z, Zhang S, Xiao W. A Novel Hybrid Algorithm Based on Grey Wolf Optimizer and Fireworks Algorithm. Beijing: University of Science and Technology Beijing; 2020.

38. Bajaj A, Abraham A. Prioritizing and minimizing the test cases using the dragonfly algorithms. ResearchGate. 2021. Available from: <https://www.researchgate.net/publication/362279068>.
39. Bajaj A, Abraham A, Ratnoo S, Gabralla LA. Test Case Prioritization, Selection, and Reduction Using Improved Quantum-Behaved Particle Swarm Optimization. Sensors. 2022;22:4374. Available from: <https://doi.org/10.3390/s22124374>.
41. Ariffin MA, Ibrahim R, Ibrahim IS, Wahab JA. Test Cases Prioritization Using Ant Colony Optimization and Firefly Algorithm. International Journal of Engineering Trends and Technology. 2022;70(3):22-28. Available from: <https://doi.org/10.14445/22315381/IJETT-V70I3P203>.
42. Badanahatti S, Rama Murthy YSS. Optimal Test Case Prioritization in Cloud Based Regression Testing with Aid of KFCM. International Journal of Intelligent Engineering and Systems. 2017;10(2). Available from: <https://doi.org/10.22266/ijies2017.0430.11>.
44. Hooda I, Singh Chhillar R. Software Test Process, Testing Types and Techniques. International Journal of Computer Applications. 2015;111(13). Available from: <https://doi.org/10.5120/19597-1433>.
45. Jia H, Zhang C, Wu S. Multi-objective software test case selection based on density analysis. In: ACM International Conference Proceeding Series. 2022. Available from: <https://doi.org/10.1145/3569966.3570010>.
46. de Souza LS, Cavalcante Prudêncio RB, de Barros FA. A hybrid particle swarm optimization and harmony search algorithm approach for multi-objective test case selection. Journal of the Brazilian Computer Society. 2015;21(1). Available from: <https://doi.org/10.1186/s13173-015-0038-8>.
47. Widiyaningtyas T, Hidayah I, Adji TB. User profile correlation-based similarity (UPCSim) algorithm in movie recommendation system. Journal of Big Data. 2021;8(1). Available from: <https://doi.org/10.1186/s40537-021-00425-x>.
48. Jalal AA, Jasim AA, Mahawish AA. A web content mining application for detecting relevant pages using Jaccard similarity. International Journal of Electrical and Computer Engineering. 2022;12(6):6461-71. Available from: <https://doi.org/10.11591/ijece.v12i6.pp6461-6471>.
49. Bobbitt Z. A simple explanation of the Jaccard Similarity Index. Statology. 2020 Dec 23. Available from: <https://www.statology.org/jaccard-similarity-index>.

50. Patra BK, Launonen R, Lee N, Zhang Y. New approaches to cosine similarity in collaborative filtering. *Journal of Information Science*. 2021. Available from: <https://journals.sagepub.com/doi/abs/10.1177/0165551520924489>.
51. Polatidis N, Georgiadis CK, Pimenidis E, Mouratidis H. A multi-level recommendation method to improve the quality of recommendations. *Computers in Human Behavior*. 2022. Available from: <https://www.sciencedirect.com/science/article/pii/S0747563222000521>.
52. Liu X, He Q, Tian Y, Lee W, McPherson J, Li J. Improving recommendation accuracy by combining popularity and similarity. *Information Processing & Management*. 2021. Available from: <https://www.sciencedirect.com/science/article/pii/S0306457321000151>.
53. Charu C. The behavior of Euclidean distance in high-dimensional spaces. *Journal of Big Data*. 2023.
54. Vescan A, et al. On the use of evolutionary algorithms for test case prioritization in regression testing considering requirements dependencies. In: *Proceedings of the 1st ACM International Workshop on AI and Software Testing/Analysis*. 2021.
55. Muhammad H, Khushi T, Nazir F, Yar A, Masood T. Regression Test Case Selection Using a Genetic Algorithm. *Journal of Xidian University*. 2023;17(11).
56. Singhal S, Jatana N, Subahi AF, Gupta C, Khalaf OI, Alotaibi Y. Fault Coverage-Based Test Case Prioritization and Selection Using African Buffalo Optimization. *Computers, Materials and Continua*. 2023;74(3). Available from: <https://doi.org/10.32604/cmc.2023.032308>.
57. Yang XS. A new metaheuristic bat-inspired algorithm. In: *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*. Berlin, Heidelberg: Springer; 2010. p. 65-74. Available from: https://doi.org/10.1007/978-3-642-12538-6_6.
58. Eberhart R, Kennedy J. A new optimizer using particle swarm theory. In: *Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS95)*. IEEE; 1995. p. 39-43. Available from: <https://doi.org/10.1109/MHS.1995.494215>.
59. Palak, Gulia P. Hybrid swarm and GA based approach for software test case selection. *International Journal of Electrical and Computer Engineering*. 2019;9(6):4898-4903. Available from: <https://doi.org/10.11591/ijece.v9i6.pp4898-4903>.
60. Choi SS, Cha SH, Tappert CC. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*. 2010;8(1):43-8.

61. Tan PN, Steinbach M, Kumar V. Introduction to Data Mining. Boston: Addison-Wesley; 2005.
62. Yang XS. Bat Algorithm: Literature review and applications. International Journal of Bio-Inspired Computation. 2013;5(3):141-9. Available from: <https://doi.org/10.1504/IJBIC.2013.055093>.
63. de Oliveira Neto FG, Torkar R, Machado PDL. Full modification coverage through automatic similarity-based test case selection. Information and Software Technology. 2016;80: Available from: <https://doi.org/10.1016/j.infsof.2016.08.008>.
64. Raamesh L, Radhika S, Jothi S. A cost-effective test case selection and prioritization using hybrid battle royale-based remora optimization. Neural Computing and Applications. 2022;34(24). Available from: <https://doi.org/10.1007/s00521-022-07627-1>.
65. Dalal S, Solanki K. Performance analysis of BCO-m-GA technique for test case selection. Indian Journal of Science and Technology. 2018;11(9). Available from: <https://doi.org/10.17485/ijst/2018/v11i9/110048>.
66. Li Z, Harman M, Hierons RM. Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering. 2007;33(4):225-37.
67. Anuar MA, Sahid MZ, Zainal N. Comparative Analysis of Test Case Prioritization Using Ant Colony Optimization Algorithm and Genetic Algorithm. Journal of Soft Computing and Data Mining. 2023;4(2). Available from: <https://doi.org/10.30880/jscdm.2023.04.02.005>.
68. Lima JAP, Vergilio SR. Test Case Prioritization in Continuous Integration environments: A systematic mapping study. Information and Software Technology. 2020;121:106268.
69. Gupta PK. K-step crossover method based on genetic algorithm for test suite prioritization in regression testing. Journal of Universal Computer Science. 2021;27(2). Available from: <https://doi.org/10.3897/jucs.65241>.
70. Panda N, Mohapatra DP. Test scenario prioritization from user requirements for web-based software. International Journal of System Assurance Engineering and Management. 2021;12(3). Available from: <https://doi.org/10.1007/s13198-021-01056-4>.

71. Bag S, Kumar SK, Tiwari MK. An efficient recommendation generation using relevant Jaccard similarity. *Information Sciences*. 2019;483: Available from: <https://doi.org/10.1016/j.ins.2019.01.023>.
72. Akila TK, Malathi A. Test case prioritization using modified genetic algorithm and ant colony optimization for regression testing. *International Journal of Advanced Technology and Engineering Exploration*. 2022;9(88):384. Available from: <http://dx.doi.org/10.19101/IJATEE.2021.874727>.
73. Odili JB, Noraziah A. African buffalo optimization for global optimization. *Current Science*. 2018;114(3):627-36. Available from: <https://doi.org/10.18520/cs/v114/i03/627-636>.
74. Olofsson A, Larsson C. Test case selection based on code changes and risk of regression. Bachelor Thesis, Lund University, 2022.
75. Cheruiyot V. Machine learning driven software test case selection. Master Thesis, Concordia University of Edmonton, 2021.
76. Valk MV. Test selection, minimization and prioritization. Master Thesis, University of Groningen, 2022.

APPENDIX A: TEST CASES

Test ID	Test Description	Pre-Condition	Test Steps	Test Data	Expected Results	Actual Results
TC1	User Registration with Valid Data - Sign Up	1. Email area should be contain @ character. 2. There should not be existed email used.	1. Open the website. 2. Navigate to "Sign Up". 3. Enter valid user details. 4. Click "Signup" button.	Name: test Email: test123@test.com	The user should successfully go to the "Account Information" page.	
TC2	User Registration with Invalid Data - Sign Up	1. Email area should not be contain @ character.	1. Open the website. 2. Navigate to "Sign Up". 3. Enter invalid user details. 4. Click "Signup" button.	Name: test Email: test123test.com	An error should be visible.	
TC3	Enter Account Information - Sign Up	1. All necessary areas should be completed. 2. Steps that TC1 should be completed.	1. Fill these areas: Title, Name, Email, Password, Date of Birth, First Name, Last Name, Company, Address, Country, State, City, Zipcode, Mobile Number. 2. Click "Create Account" button.	Title: Mr Name: TestName Email: test123@test.com Password: Test*123 Date of Birth: 1stJan1990 First Name: Test Last Name: Test Company: Test Address: Test Country: Tr State: Ege City: Izmir Zipcode: enter a 6 digit zipcode Mobile number: enter a dummy number	The user should successfully see the success message as "Account Created!"	
TC4	Enter Account Information -	1. All necessary areas should be completed.	1. Fill these areas: Title, Name, Email, Date of Birth, First Name, Last	No test data	1. "Password should be entered!" information should be visible.	

	Missing Necessary Areas - Sign Up	2. Steps that TC1 should be completed.	Name, Company, Address, Country, State, City, Zipcode, Mobile Number. 2. Do not fill the Password area.		2. The user cannot create his/her account.	
TC5	User Login with Valid Credentials - Login	1. The user should completed TC1 and TC3 (An account should be created.)	1. Open the website. 2. Navigate to "Sign Up". 3. Enter valid user details. 4. Click "Login" button.	Email: test123@test.com Password: Test*123	1. The user should be on the "Dashboard" page. 2. The user should see the "Delete Account" button. 3. The user should see the "Logout" button.	
TC6	User Login with Invalid Credentials - Login	1. The user should completed TC1 and TC3 (An account should be created.)	1. Open the website. 2. Navigate to "Sign Up". 3. Enter invalid user details. 4. Click "Login" button.	Email: test123@test.com Password: Test*1234	1. The user should see the "Your email or password is incorrect!" message on the "Login" page.	
TC7	Product Search with Valid Term	1. The user should be on the "Products" page. Dashboard > Products	1. Enter a valid product name in the search bar. 2. Click "Search".	Men Tshirt	1. The user should see the related product on the screen.	
TC8	Product Search with Invalid Term	1. The user should be on the "Products" page. Dashboard > Products	1. Enter an invalid product name in the search bar. 2. Click "Search".	Man T shirt	1. The user should not see the related product on the screen. 2. An empty page should be visible under the "Searched Products" section.	
TC9	Adding a Product to the Cart	1. The user should be on the "Products" page. Dashboard > Products 2. A valid product should be searched.	1. Search for a product. 2. Select a product from the search results. 3. Click "Add to Cart".	Men Tshirt	1. The "Added!" success message should be visible on the screen.	

TC10	Removing a Product from the Cart	TC9 should be completed. (A product should be added to the cart)	1. Go to Cart section. 2. Click "X" button.	No test data	1. The product should not be visible.	
TC11	Updating Product Quantity in Cart	TC9 should be completed. (A product should be added to the cart)	1. Add a product to the cart once again. 2. Navigate to the cart. (By clicking Cart button)	No test data	1. The user should see the quantity as 2	
TC12	Logging Out	The user should completed TC5 (The user should be logged in to the system)	1. Click the "Logout" button.	No test data	The user should be logged out and redirected to the homepage. (Login to your account page)	
TC13	Login without Email	1. The user should completed TC1 and TC3 (An account should be created.)	1. Open the website. 2. Navigate to "Sign Up". 3. Enter password. 4. Click "Login" button.	Email: Password: Test*1234	1. The user should see the "Your email or password cannot be empty!" message on the "Login" page.	
TC14	Login without Password	1. The user should completed TC1 and TC3 (An account should be created.)	1. Open the website. 2. Navigate to "Sign Up". 3. Enter email. 4. Click "Login" button.	Email: test123@test.com Password:	1. The user should see the "Your email or password cannot be empty!" message on the "Login" page.	
TC15	Register with Special Characters in Email	There is no pre-condition for this test case.	1. Open the website. 2. Navigate to "Sign Up". 3. Enter a name with special characters. 4. Enter a valid email. 5. Click "Signup" button.	Name: !@#\$\$%^&* Email: test123@test.com	An error message should be displayed indicating invalid email format.	
TC16	Proceeding to Checkout	The user should be logged in on the system.	1. Open the website. 2. Add products to the cart. (You can add any products to the cart.) 3. Navigate to the cart.	No test data	1. The "Address Details" screen should be opened. 2. The user should see the products that added to the cart.	

			4. Click "Proceed to Checkout".			
TC17	Checkout with Empty Cart	1. The user should be logged in on the system. 2. Ensure the cart is empty.	1. Open the website. 2. Navigate to the cart.	No test data	1. The user should not be able to see the "Proceed to Checkout" button. 2. The user should see the "Cart is empty! Click here to buy products." message on the screen.	
TC18	Search with Empty Input	The user should be logged in on the system.	1. Open the website. 2. Go to the Products screen. 3. Leave the search bar empty. 4. Click the "Search" button.	No test data	All products should be visible.	
TC19	Searching for Out-of-Stock Items	The user should be logged in on the system.	1. Open the website. 2. Go to the Products screen. 3. Enter the name of an out-of-stock item in the search bar. 4. Click the "Search" button.	A product that does not exists.	Under the "Searched Products" there are no visible items.	
TC20	Product Categories	The user should be logged in on the system.	1. Open the website. 2. Check the product categories.	No test data	The user should be able to see these product categories: 1. Women 2. Men 3. Kids	
TC21	Checking Cart After Login	1. The products should be carted before re-login.	1. Open the website. 2. Navigate to the cart.	"Email: test123@test.com Password: Test*123"	The cart should retain the added products after re-login.	
TC22	Product Brands	The user should be logged in on the system.	1. Open the website. 2. Check the product brands.	No test data	The user should be able to see these product categories: 1. Polo 2. H&M 3. Madame	

					4. Mast & Harbour 5. Babyhug 6. Allen Solly Junior 7. Kookie Kids 8. Biba	
TC23	Verify Scroll Up using 'Arrow' button and Scroll Down functionality	The user should be logged in on the system.	1. Open the website. 2. Scroll down page to bottom.	No test data	Verify 'Subscription' is visible	
TC24	Verify Scroll Up without 'Arrow' button and Scroll Down functionality	The user should be logged in on the system.	1. Open the website. 2. Scroll up page to top.	No test data	Verify 'Home' is visible	
TC25	Searching Results by Price	The user should be logged in on the system.	1. Open the website. 2. Go to the Products screen. 3. Enter the price of a product. 4. Click the "Search" button.	400	Under the "Searched Products" there are no visible items.	
TC26	Navigating Product Details on the Cart	TC9 should be completed. (A product should be added to the cart)	1. Open the website. 2. Go to the Cart screen 3. Click the products' description.	A product that is exists.	The user should be able to see that: 1. "Write Your Review" section 2. Product details like "availability, condition and brand"	
TC27	Verify "Contact Us" Function Works - Without Any Attachment.	1. The user should be logged in on the system. 2. The user should be enter the valid email.	1. Open the website. 2. Go to the Contact Us screen. 3. Fill in these areas: "Name, Email, Subject, Message". 4. Click the Submit button.	Name: test Email: test@test.com Subject: test Message: test	The user should be able be able to see "Success! Your details have been submitted successfully." message.	
TC28	Verify "Contact Us" Function Works - With An Attachment.	1. The user should be logged in on the system. 2. The user should	1. Open the website. 2. Go to the Contact Us screen. 3. Fill in these areas:	Name: test Email: test@test.com Subject: test Message: test	The user should be able be able to see "Success! Your details have been submitted successfully." message.	

		enter a valid email. 3. The attached document should be valid.	"Name, Email, Subject, Message". 4. Upload a valid document. 5. Click the Submit button.			
TC29	Account Deletion	The user should be logged in on the system.	1. Open the website. 2. Click the Delete Account section.		"Account Deleted! Your account has been permanently deleted! You can create new account to take advantage of member privileges to enhance your online shopping experience with us." message should be visible.	
TC30	Subscription Functionality	The user should be logged in on the system.	1. Open the website. 2. Scroll down page to bottom. 3. Under the Subscription area, enter a valid email address. 4. Click the Submit button.	Email: test@test.com	"You have been successfully subscribed!" message should be visible.	
TC31	Subscription Functionality - invalid mail	1. The user should be logged in on the system. 2. The user should enter an invalid mail address.	1. Open the website. 2. Scroll down page to bottom. 3. Under the Subscription area, enter an invalid email address. 4. Click the Submit button.	Email: test	"Please enter a valid email address!" message should be visible.	
TC32	Filtering By Product Brands	The user should be logged in on the system.	1. Open the website. 2. Click a brand on the left side.	Polo	The user verifies that only products that are under Polo is visible.	
TC33	Entering Shipping Information	1. The user should be logged in on the system. 2. TC9 should be completed. (A product should be added to the cart)	1. Open the website. 2. Go to the Cart screen. 3. Click the Proceed to checkout button. 4. Click "Place Order" button.		1. Verify that the address and product information are correct. 2. The payment screen should be opened.	

TC34	Entering Payment Information - invalid characters	The user should completed the TC33.	1. Enter invalid characters under: Name on card Card number CVC Expiration Date	Name on card: Test Card number:Test CVC:Test Expiration Date:Test	The user should not be able to enter the string data to card number, CVC and expiration date	
TC35	Entering Payment Information - valid characters	The user should completed the TC33.	1. Enter invalid characters under: Name on card Card number CVC Expiration Date	Name on card: Test Card number:1231231231313 CVC:123 Expiration Date:08/29 <i>A valid credit card should be entered.</i>	Congratulations! Your order has been confirmed! message should be completed.	

APPENDIX B: SIMILARITY MATRIX BY JACCARD

	TC1	TC2	TC3	TC4	TC5	TC6	TC13	TC14	TC15	TC31	TC9	TC10	TC11	TC16	TC21	TC26	TC27	TC28	TC31	TC39	TC23	TC24	TC27	TC28	TC30	TC33	TC34	TC35	TC31	TC32	TC22	TC29	TC32	TC35		
TC1	1	0,5	0,1428 57142 9	0,1428 57142 9	0	0	0	0	0,1428 57142 9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
TC2	0,5	1	0,125	0,2857 14285 7	0	0,1428 57142 9	0,1428 57142 9	0,1428 57142 9	0,2857 14285 7	0,1428 57142 9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,1428 57142 9	0	0	0	0	0	0	0			
TC3	0,1428 57142 9	0,125	1	0,3333 33333 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
TC4	0,1428 57142 9	0,2857 14285 7	0,3333 33333 3	1	0	0,1666 66666 7	0,1666 66666 7	0,1666 66666 7	0,1428 57142 9	0,1666 66666 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,1666 66666 7	0	0	0	0	0	0	0			
TC5	0	0	0	0	1	0,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
TC6	0	0,1428 57142 9	0	0,1666 66666 7	0,2	1	0,2	0,2	0,1666 66666 7	0,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,2	0	0	0	0	0	0	0	0	0	
TC13	0	0,1428 57142 9	0	0,1666 66666 7	0	0,2	1	0,5	0,1666 66666 7	0,2	0	0	0	0	0,2	0	0	0	0	0	0	0	0	0	0	0,2	0	0	0	0	0	0	0	0	0	
TC14	0	0,1428 57142 9	0	0,1666 66666 7	0	0,2	0,5	1	0,1666 66666 7	0,2	0	0	0	0	0,2	0	0	0	0	0	0	0	0	0	0	0,2	0	0	0	0	0	0	0	0	0	
TC15	0,1428 57142 9	0,2857 14285 7	0	0,1428 57142 9	0	0,1666 66666 7	0,1666 66666 7	0,1666 66666 7	1	0,1666 66666 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,1666 66666 7	0	0	0	0	0	0	0	0	0	0,2

T C 3 1	0	0,1428 57142 9	0	0,1666 66666 7	0	0,2	0,2	0,2	0,1666 66666 7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,2	0	0	0	0	0	0	0
T C 9	0	0	0	0	0	0	0	0	0	0	1	0,25	0,25	0,2	0,2	0,2	0	0	0	0	0	0	0	0	0	0,2	0	0	0	0	0	0
T C 1 0	0	0	0	0	0	0	0	0	0	0	0,2 5	1	0,3333 33333 3	0,2 5	0,2 5	0,2 5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T C 1 1	0	0	0	0	0	0	0	0	0	0	0,2 5	0,3333 33333 3	1	0,2 5	0,2 5	0,2 5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T C 1 6	0	0	0	0	0	0	0	0	0	0	0,2	0,25	0,25	1	0,2	0,2	0	0	0	0	0	0	0	0	0,2 5	0	0	0	0	0	0	0
T C 2 1	0	0	0	0	0	0	0,2	0,2	0	0	0,2	0,25	0,25	0,2	1	0,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T C 2 6	0	0	0	0	0	0	0	0	0	0	0,2	0,25	0,25	0,2	0,2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T C 7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0,2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T C 8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,2 1	0	0,2	0	0	0	0	0	0	0	0	0	0	0	0
T C 1 8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,2 1	0,2 5	0	0	0	0	0	0	0	0	0	0	0	0
T C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0,2 1	0,2 5	0	0	0	0	0	0	0	0	0	0	0	0

APPENDIX C: SIMILARITY DISTANCE MATRIX BY JACCARD

APPENDIX C: SIMILARITY DISTANCE MATRIX BY JACCARD

T C 3 1		0,8571 42857 1		0,8333 33333 3					0,8333 33333 3																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
------------------	--	----------------------	--	----------------------	--	--	--	--	----------------------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

APPENDIX D: SURVEY QUESTIONS

No	Target Topic	Question	Ranks
1	Time Management	How effectively does your team manage time during the testing phase? (This can include various testing phases such as sprint testing and release execution.) (Consider alternative working methodologies if you are not using sprints.)	1: Extremely Inefficient 2: Inefficient 3: Neutral 4: Efficient 5: Extremely Efficient
2	Test Coverage	How extensive is the coverage of test cases during the limited testing time? (Especially for regression test cases.)	1: Very Limited Coverage 2: Limited Coverage 3: Moderate Coverage 4: Extensive Coverage 5: Comprehensive Coverage
3	Test Case Complexity	How complex are the test cases that you currently test the module? (This can include various testing types such as regression, smoke, etc.)	1: Very Simple 2: Simple 3: Moderate Complexity 4: Complex 5: Very Complex
4	Risk Identification	How confident are you in your team's ability to identify high-risk areas for testing? Are they capable of identifying high-risk areas?	1: Not Confident at All 2: Slightly Confident 3: Moderately Confident 4: Confident 5: Very Confident
5	Test Coverage Maintenance	How challenging is it to maintain adequate test coverage throughout the testing phase? (This can include both of outdated test cases and new ones.)	1: Extremely Challenging 2: Very Challenging 3: Moderately Challenging 4: Slightly Challenging 5: Not Challenging
6	Balancing Quality and Deadlines	How difficult is it to balance between delivering high-quality software and meeting project deadlines?	1: Extremely Difficult 2: Very Difficult 3: Moderately Difficult 4: Slightly Difficult 5: Not Difficult
7	Regression Testing	How time-consuming is regression testing in your software development lifecycle?	1: Extremely Time-Consuming

			2: Very Time-Consuming 3: Moderately Time-Consuming 4: Slightly Time-Consuming 5: Not Time-Consuming
8	Resource Allocation	How challenging is it to allocate testing efforts effectively across different modules or features?	1: Extremely Challenging 2: Very Challenging 3: Moderately Challenging 4: Slightly Challenging 5: Not Challenging
9	Test Case Dependencies	How challenging is it to manage dependencies between test cases during testing? (A dependency example: waiting a couple of days from development to testing the test module during the sprint.)	1: Extremely Challenging 2: Very Challenging 3: Moderately Challenging 4: Slightly Challenging 5: Not Challenging
10	Manual vs. Automated Testing	How difficult is it to create a balance between manual and automated testing efforts? (Consider communication efforts if your position involves only one of them.)	1: Extremely Difficult 2: Very Difficult 3: Moderately Difficult 4: Slightly Difficult 5: Not Difficult
11	Test Data Management	How challenging is it to manage and maintain test data for various testing cases?	1: Extremely Challenging 2: Very Challenging 3: Moderately Challenging 4: Slightly Challenging 5: Not Challenging
12	Dependencies Impact	How often do delays caused by dependencies impact testing timelines? (A dependency example: waiting a couple of days from development to testing the test module during the sprint.)	1: Always 2: Often 3: Sometimes 4: Rarely 5: Never
13	Test Case Maintenance	How challenging is it to maintain and update test cases as the software evolves?	1: Extremely Challenging 2: Very Challenging 3: Moderately Challenging

			4: Slightly Challenging 5: Not Challenging
14	Outdated Test Cases	How often do outdated test cases remain in the testing suite, causing confusion?	1: Always 2: Often 3: Sometimes 4: Rarely 5: Never
15	Documentation Maintenance	How challenging is it to maintain up-to-date documentation for test cases?	1: Extremely Challenging 2: Very Challenging 3: Moderately Challenging 4: Slightly Challenging 5: Not Challenging
16	Test Case Reusability	How often are test cases reused across different projects or releases?	1: Always 2: Frequently 3: Occasionally 4: Rarely 5: Never
17	Traceability Management	How challenging is it to maintain traceability between requirements, test cases, and defects?	1: Extremely Challenging 2: Very Challenging 3: Moderately Challenging 4: Slightly Challenging 5: Not Challenging
18	Bias in Prioritization	How often does bias influence(anticipation) the prioritization of test cases within your team? (Consider instances where certain test cases are given higher priority based on subjective judgments rather than objective criteria.)	1: Always 2: Frequently 3: Occasionally 4: Rarely 5: Never
19	Execution Time Impact	How often does the time required to execute test cases impact sprint timelines? (Consider alternative working methodologies if you are not using sprints.)	1: Always 2: Frequently 3: Occasionally 4: Rarely 5: Never
20	Bug Detection Rate	What is the frequency of bugs caught as a result of executing written test cases?	1: Very Rarely 2: Rarely 3: Occasionally 4: Frequently 5: Very Frequently