Test Design Optimisation of factors and levels by Covering double and triple mode Combinations using Orthogonal Array test strategies and Random Forest Algorithm

S. Malathi[1], M. Sangeetha[2,*], Faiyaz Ahmad[3], Saravanan M. S.[4], T. Kalachelvi[4]

[1]Department of AI & DS, Panimalar Engineering College, Chennai, India.
[2]Department of Computer Science and Engineering, Panimalar Engineering College, Chennai, India
[3]Department of Computer Engineering, Jamia Millia Islamia, New Delhi, 110025, India
[4]Department of Computer Science and Engineering, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, Tamilnadu, India.

Emails: malathi.raghuram@gmail.com; sangeetharemi@yahoo.co.in; ahmad.faiyaz@gmail.com; sarvananms@saveetha.com; ai_dshod@pit.ac.in

Abstract

Testing is a process of trying to find out every believable fault or weakness in a project. In today's world, software products and components play a vital part in our life. Software testing is a world, it contains its own life cycle consists of the following stages – Requirements, Test Plan, Test Design, Test Execution, Defect reporting/tracking. The core of software testing lies in writing test cases based on specifications. Software testers play a vital role writing the test cases during test design phase of software testing life cycle. Research have proved that writing test cases is the most time killing and challenging activity among other testing life cycle phases. It is very crucial to sequence and write optimized test cases to increase the rate of fault identification during test design phase as early as possible. There are various proven test design techniques available which focuses on optimizing test cases in different test stages. Our key focus in this paper is to identify the optimized test cases minimizing the actual number of test cases with minimal effort using OATS (Orthogonal Array Test Strategy) techniques covering double mode and triple mode test combinations and Random Forest algorithm.

Keywords: defects; testsuite; sequence; Equivalance Partitioning; Boundaryvalue analysis; optimization;

1. Introduction

Software testing is an activity aimed at evaluating an attribute or capability of a program or a system. Defect causes the software to perform in a not deliberate manner.

Identifying defects early as possible reduces enormous rework, ultimately saving plenty of time and effort ensuring better quality of the work product. The core software development phases are Requirements, Design, Coding, Testing and Deployment. It is very critical identifying the defects in the right stage where it got introduced. Say for example: Defects that got created in requirements phase; we must identify in requirements stage itself [3]. In software testing, we call this process as "Stage Containment" The excellence of the software product lays in the effectiveness of Software testing. The amount of costing that we spend to ensure quality in the work product is called as "Cost of Quality – COQ". Cost of Quality mainly involves 3 main testing costs – Prevention, Appraisal and Rework costs. The costs that are incurred for preventing defects in organizations are termed as Prevention costs. Example of prevention costs includes implementing process standards; imparting training to resources etc., the costs that are incurred for appraising a defect is called Appraisal costs [4]. Example of Appraisal costs includes the salary paid to every individual

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023

software tester for identifying defects on day-to-day basis. The costs that are incurred for any kind of rework that a tester is doing in any of the software development phases is called as Rework costs. Example of Rework costs includes the rework that a tester is doing in finding the exact root cause of a defect that he has identified in test stage. When he started backtracking, he concluded that the origin of the defect is from design stage. The time and effort that he has put in to discover this defect has to get recognized. This cost is termed as Rework cost. When we term anything as quality costs, mainly these three costs will come into picture. It is very important to ensure better software quality before we deliver the product to the customer. At the same time, customer will not be happy or satisfied if he spends more to achieve the same [5-9]. So, aim of software testing is to make sure better quality and at the same time minimizing software quality costs. It tells the major aim of software testing is to ensure better quality work product minimizing cost of quality. We cannot be able to compromise on prevention and appraisal costs. But we can think of minimizing or reducing rework costs. Practically rework costs can be minimized if we find out a way where there is no or negligible rework in the work product. This can be achieved if the correct defects are identified and fixed at the right stage, then living it to propagate and identify the same in the later stage. This is what we call as "Stage Containment". A very popular approach used in organizations to identify deviations at the earlier in the testing life cycle includes finding out the correct test cases that reveals the important defects at the earliest in the testing life cycle. This approach is called as Risk based testing. This approach is used to prioritize the test cases based on criticality and priority, so that the high severe defects are identified and fixed early which minimizes risk in the work product[10]. Experience has proved that the most time bound activity in testing lies in designing the test cases according to the requirements. We have a testing principle that states, "Exhaustive Testing is not possible". It means, it is practically impossible for someone to test all possible combinations and derive test cases accordingly. So, testing is to design optimized test suites which will minimize the time as well us uncover maximum test coverage with minimal test cases.

2. Existing Work

Various studies related to test design optimization has been proposed and analyzed by various test analysts before which analyses different test stages. Previous method, used to make the similarity evaluation less difficult, is to document and analyze the execution hint of test cases rather the test cases themselves [1]. The drawbacks of this approach are the instrumentation of the production code, and the issue is to get consistent traces from guide test cases.

Test case prioritization:

Considerable research effort has been dedicated to prioritizing test cases with test execution statistics, that includes the full exposure of code components, the coverage of code components not previously covered [2]. An optimization technique called Equivalence partitioning focuses on partitioning the input space into small number of partitions according to the requirements. For every partition only one test case is needed. This technique defines the number of test cases by identifying the test partitions (equivalence classes). This technique is also called as domain analysis. There are 2 types of classes identified – Valid and Invalid. Valid corresponds to inputs deemed valid from requirements and Invalid corresponds to input that is erroneous from the requirements. In this technique, groups are formed as partitions based on the condition that they all test the same thing, if one test can identify a defect, the other test cases will also identify the same. If one test is not capable of identifying the defect, the other test cases in the group will also not identify. Equivalence classes are formed when they involve the same input variables, they result in similar operation, they affect the same output variables etc., the next technique is called as Boundary Value Analysis and in short, we call as BVA [11-12]. This technique leads to selecting the test cases that exercise the boundaries of the partition. This technique complements Equivalence partitioning. Instead of selecting one value in the partition, BVA leads to select the test cases at both left and right edges of the partition. Experiences have proved that more errors form as a cluster in the boundaries.

Orthogonal Array Test Strategy (OATS) method is very well suited for large real-world applications when the test inputs are huge. Exhaustive testing is practically difficult testing each test combination here. Considering an example of Tatkal train ticket booking in our Indian Railways website, the input combinations that we need to test may include the following: Source station, Destination station, Date of travel, Number of passengers, Quota, Class of travel etc., So, there are plenty of input combinations that we can provide and based on every possible combination, the IRCTC application will respond and provide output accordingly. These varying inputs are called as "Factors". Testing every input combination is really challenging and time consuming and for every factor there can be "n" number of levels. For example: the factor class can have various levels like "General", 3rd AC, 2nd AC, 1st AC etc., Most of the real-world problems are of factors-levels type. These problems are called as "Combinatorial problems". OATS techniques mainly focus on defining optimized test combinations of factors and levels and it is mainly preferred for combinatorial

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023

problems type. Defect types are classified into various modes like Single mode defect, double mode defect etc., Suppose, we try to combine 10 factors and as a result we are getting an unexpected defect [13]. If this defect is because of one single input, it is classified as a Single mode defect. Example: When working with an editor application and trying to test Verdana font throws an "unsupported format error". Let's assume other than Verdana; the application supports all other font names. This is an example of a single mode error caused by a single input called "font name". Considering another example when trying to combine font name "Verdana" with size 12 we get a "unsupported format error". Let's assume in this case, font name Verdana works with all other sizes except "12" and font size "12" works with all other font names except "Verdana" [14-16]. This is an example of a double mode defect where this error gets introduced when 2 different input combinations combine. Likewise, applications will have triple mode, multi-mode errors and so on. Various research has proved that most of the deviations/defects in software projects are mainly because of double mode combinations of inputs. This has been taken into consideration and Orthogonal Array test strategy algorithms have proposed creating optimized test design that uncovers 100% of all double mode combination of inputs. This ensures maximum coverage with minimal effort. However, there are no techniques so far that focus optimizing other mode input combinations like single mode, triple mode, or multi-mode combinations. Also, there are no proven techniques that provide a well-defined plan during test plan phase itself on effectively test sequence the modules for testing. This will enable a proper sequence for testers to decide which modules to test first and which modules to test next ensuring maximum test coverage with minimal effort. Once the test sequence has been decided and signed off, the key role of testers is to optimize the test cases created according to the test sequence decided in the test plan phase.

3. The Proposed Work

Once effective test sequence methods have been decided using dependency structure algorithms, it is very important to create an optimized test design considering the factors and levels effectively and define a capable test suite to identify all critical defects with minimal effort. The overall aim of testing is to ensure that all risks in the software are eliminated, and the software attains satisfied quality in terms of conformance to requirements as well as fitness for use. It is also extremely important that the quality factors are achieved with negligible or minimal rework ensuring less cost of quality. As we have seen already, test design is a stage which is more challenging to define the test logic covering all the possible combinations that should satisfy the real needs of the customer and the actual users. It is practically impossible to create test logic for each possible state and its really time consuming. OATS is an optimization technique that ensures test optimization with minimal test cases ensuring maximum coverage. It generates both positive and negative test suite optimizations. It allows deciding infeasible combinations where combinations are omitted during test case generation and decide on high priority combinations during optimized suite generation. Orthogonal Array Testing Strategy (OATS) is a systematical, measurable method for testing pair-wise associations by inferring appropriate little arrangement of experiments from countless. The testing technique can be utilized to lessen the quantity of test mixes and furnish most extreme scope with a base number of experiments. OATS use a variety of qualities speaking to variable factors that are joined match shrewd as opposed to speaking to all blends of components and levels. Considering distinctive imperfection modes are incorporates – Single mode blame, where the most straightforward sort of issue to identify is one that is activated by a solitary variable in a solitary state. This is known as solitary mode blame. In Double-mode blame, the imperfection that relies on two conditions (even though two things work without anyone else's input, they fall flat when combined (associated) together)). The most troublesome sort of issue to discover by discovery testing is one in which at least three things in blend don't cooperate [17]. This is known as triple-mode blame, or even for the most part as a multi-mode blame. The heritage frameworks have no viable following instruments for following the test advance when the quantity of emphasis of testing is high. The actual test case forecasting is really challenging. Design of over test cases will result in redundancy or duplicate cases. Test design challenges include test case forecasting, over test cases, relevant test coverage and optimum proving. All existing OATS optimization tools focus only on pair-wise testing combinations. There are no options for omission of factors/levels for certain realistic for-sure combinations (Infeasible combinations) and no options to decide on high priority levels that should appear a greater number of times in the pair-wise combinatorial output samples. The test cases planning to create using OATS are a systematic and statistical method of pair-wise combinations of factors across their levels. It creates an optimized test suite with lesser test cases. It detects all single mode, double mode, and triple mode defects. It provides an option for generating full factorial set (exhaustive combination) that allows a provision to decide whether to go for test suite optimization. It increases confidence in the system by executing a concise set of tests and uncovering most of the bugs. Negative test case optimization can also be generated. Infeasible combinations can be removed for meaningful scenarios. Precedence can be given for factors and levels that

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023

should appear a greater number of times in the combinatorial output. The OATS optimization mechanism shows in fig.1 consists of 3 tiers – Tier1: also called application layer (front end) that can be created using HTML and JavaScript, defining the user interface for providing the inputs and generating the outputs. Tier2: It consists of the entire business logic of the proposed tool, and it can be designed in C# or Java. Tier3: It consists of the database layer that uses MS-Access/Oracle as the backend to store all the outcomes of the statistical inputs i.e., combinatorial outputs [18-20].

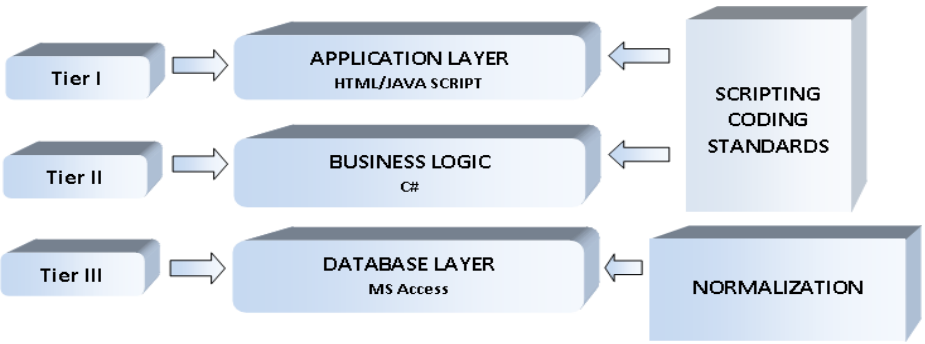


Figure 1: OATS optimization mechanism.

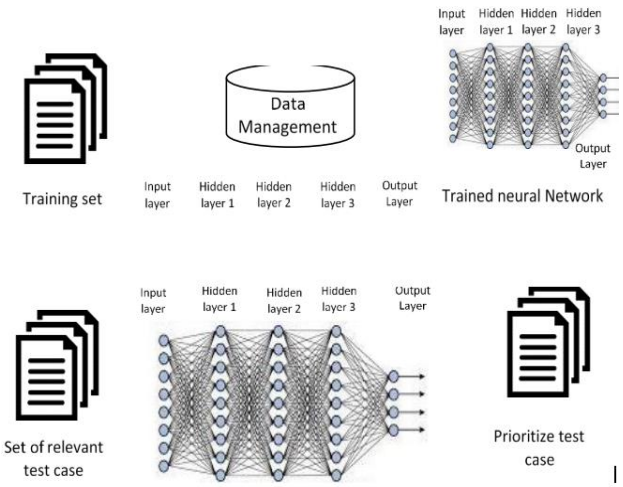
4. Deep Learning-Random Forest Algorithm (Dl-Rl) For Black Box Test Case Selection:

4.1 Features of DL-RL:

Increase the rate of fault identification during the test design phase as early as possible. It is based on training and testing phase of test case model. A large set of test cases is manually executed. Test case selection has proposed with DL-RL algorithm to solve the problem of TC selection with multiple criteria. The basic deep learning algorithm starts its search process with a random set of test data stored in the management unit. Each test case represents a candidate solution for the problem being solved.

4.2 Steps involved in DL-RL:

There are mainly two operations - training phase, and testing phase. The data we used to perform test case prioritization is stored in a data management system. In training phase, a test expert must select a training set for the DL algorithm. Test case requires the expert to select a set of positive test cases, i.e., test cases which are of high importance, for the current version under test. In testing phase, the deep learning approach uses a static data, requirement for test case and failures which shows in below fig 2.



Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023

Figure 2:Traning and testing phase

The following fig.3 shows the system architecture of test design phase phase after effective module sequencing using dependency structure matrix [21] in planning phase.

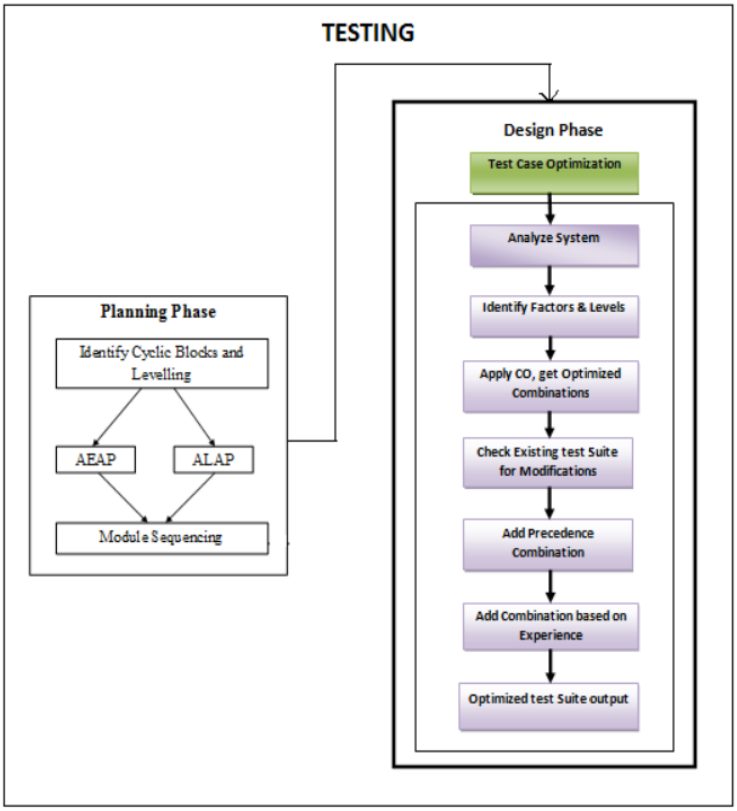


Figure 3: Architecture Diagram

One of the main challenges we face during test design phase is to manage huge number of test cases that we need to create matching the customer requirements and to execute all of them considering the constraint factors like time, budget etc.[22].

4.3 Application – 1:

We used a small-scale application which we need to test for compatibility with different browsers and operating systems we must consider the support of the application with various browsers like IE, Chrome, Firefox etc., We also need to consider the operating systems like Windows, Mac, Linux etc., Consider the application can be connected through LAN, ISDN etc. Considering the factors – Web browser, Operating System, and Connection type we have various levels as follows: 1. Web Browser (IE, Chrome, Firefox) 2. Operating System (Windows, Mac, Linux) and 3. Connection Type (LAN, ISDN). For this simple example, if we need to test all possible combinations, we will require 3 * 3 * 2 – 18 combinations. It means we must write 18 independent test cases that will test this example exhaustively. Considering this example, identifying, and executing 18 test cases is not a big deal, it's a matter of time and effort. But today's real-world problems are not these simple as our example. In today's world organizations experience very complex combinations of factors and levels that would result in creating huge number of test combinations.

4.4 Application – 2:

Another application is used here.i.e.car insurance software that would require quite a few details before issuing insurance. In general, there are many policy types like third party, fire, theft etc., Storage modes include garage,

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023

driveaway and road. Certain vendors also offer no claim bonus types like for 0 years, 1 year, 2 years and 3+ years. Insurance license type includes provisional and full. Insurance benefits for policy holders mainly depend on their age and there can be several age categories like 17-21, 22-30, 31-40, 41-50 and 50+. Insurance value also varies based on the engine sizes of the vehicles like <1000 cc, 1000cc – 1600 cc, 1601 cc – 2000 cc, 2001 cc – 2999 cc, 3000 cc+ etc., Considering this example, if we list out the factors and respective values it would come as below:

1. Factor 1 – Policy Type (third party, fire, theft)
2. Factor 2 – Storage Mode (garage, driveaway and road)
3. Factor 3 – No claim bonus type (0,1,2,3+ years)
4. Factor 4 – License Type (Provisional, Full)
5. Factor 5 – Age of customer (17-21, 22-30, 31-40, 41-50 and 50+)
6. Factor 6 – Engine Size (<1000 cc, 1000cc – 1600 cc, 1601 cc – 2000 cc, 2001 cc – 2999 cc, 3000 cc+)

If we want to do exhaustive testing for all these 6 factors it would come to 3 * 3 * 4 * 2 * 5 * 5 = 1800 test combinations. If we do a rough estimate for one test case that it takes at least half day to build the test case, setup the test environment, execute and validate, it takes approximately 900-man days to complete testing. Considering the mandatory holidays, weekends it would equate to 900 days of testing for this simple example that we have considered. It clearly says that no testing team or customer would spend almost 4 years (900 days) for testing this insurance software functionality. If we see here in one hand exhaustive testing, testing all possible permutations and combinations is impractical and impossible having the constraints of time, budget etc., On the other hand, if we try to ignore certain test combinations, there is a huge risk of missing very important defects. Also, if we test all possible combinations there is a possibility of testing infeasible combinations. So, the main research lies here in identifying the right way to select the best possible combinations.

Orthogonal array test strategy (OATS) provides a way to select the test cases that guarantees all pair-wise combinations of inputs are tested. This strategy creates an efficient and small test set with minimal test cases instead of testing all exhaustive combinations. This strategy also creates an even distribution of all pair-wise combinations. Dr. Genichi Taguchi was one of the first one who proposed orthogonal array concepts in test design stage of testing. This strategy provides a statistical way of testing pair-wise interactions of factors and level combinations. This technique mainly focuses on defects that are mainly depends upon two conditions when they are connected. We call this as double mode defect. The most difficult kind of problem to find in functional testing using black box technique is one in which three or more things in combinations doesn't work together leading to defects. This is called as triple mode defect or generally we can call as multi-mode defect. Our main aim here is to cover all single, double, and triple mode combinations of test cases 100% ensuring optimized coverage going one step ahead of the usual OATS strategy covering pair-wise combinations.

5. Terminology of Orthogonal Array

The main terminology in working with OA concepts lies on 3 main parameters – Runs, Factors and Levels. Runs are specifying the number of rows in the array. Factors define the number of columns in the array. Levels are defining the maximum number of values that a factor can take. Orthogonal arrays follows the pattern $\mathbf{L_{runs}(Levels^{Factors})}$. Choosing the best fit orthogonal array using $\boldsymbol{L_x}(n^y)$ is extremely important after identifying the factors and levels.

$\boldsymbol{x}$ = runs = number of rows = number of test cases we need
$\boldsymbol{y}$ = factors = number of columns = number of categories
$\boldsymbol{n}$ = levels=maximum number of choices for each category

Referring to Application – 1, we discussed above, to construct an orthogonal array, we would need to construct $2^1 3^2$ array as one factor with two levels and two factors with three levels. Unfortunately, we cannot create such an array. So, the best-fit array would be a 3^3 array and we can create $L_9 3^3$ matrix as below table 1:

Table 1: Before Mapping Factor.

OA before mapping factors			
	Factor 1	**Factor 2**	**Factor 3**
Run 1	1	1	1
Run 2	1	2	2
Run 3	1	3	3

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023

Run 4	2	1	2
Run 5	2	2	3
Run 6	2	3	1
Run 7	3	1	3
Run 8	3	2	1
Run 9	3	3	2

If we map the factors and levels of Example-1 in the above array, our matrix will look as below table.2:

Table 2: After Mapping Factor

OA after mapping factors			
	Factor 1	**Factor 2**	**Factor 3**
Run 1	IE	Windows	LAN
Run 2	IE	Mac	ISDN
Run 3	IE	Linux	~
Run 4	Chrome	Windows	ISDN
Run 5	Chrome	Mac	~
Run 6	Chrome	Linux	LAN
Run 7	Firefox	Windows	~
Run 8	Firefox	Mac	LAN
Run 9	Firefox	Linux	ISDN

The next step is to convert the runs into test cases. We can also add any other test combinations that covers triple mode combinations if we have a gutt feel that our added combinations will reveal a new defect. So, if we convert Row 1 into a test case, the combination will be to test the small scale application to run in IE browser using Windows operating system and LAN connection. In the same way the second test case will be to run the small scale application in IE browser using Mac operating sytem and ISDN connection. In this example, we have reduced 18 test cases that tests all possible combinations to 9 test cases testing all possible pairwise combinations. If we deeply look into this example, we are not achieving huge savings in terms of numbers.

Referring to Application – 2, we have already seen that to test all possible combinations we need 1,800 test cases. We have considered 6 variables in this example: Policy types, Storage modes, Claim types, License types, Age range, Engine sizes. So we have to look for a $2^1 3^2 4^1 5^2$ array, but we cannot create such an array. So, the best-fit array would be a 5^6 array since the maximum number of choices we have is 5 and the variables we have is 6. The number of rows = (6-1) * 5 = 5 * 5 = 25 and so we need to have $L_{25}5^6$ array. So we need to create 25 test cases to cover all possible pair-wise combinations. Including the other multi mode combinations based on the gutt feel, even if we increase the test cases to a maximum of 10, our overall test cases will be a maximum of 30. This is a significant reduction comparing the exhaustive test set of 1800 test cases.

V.COMBINATORIAL TEST OPTIMIZER PROCESS

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023

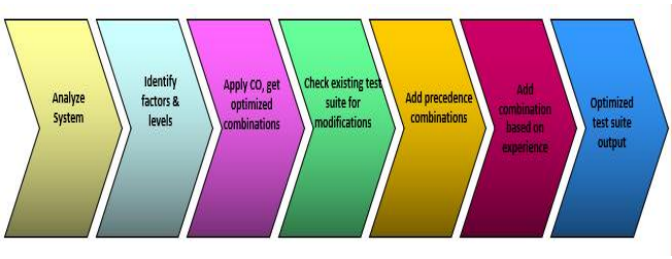


Figure 4: Combinatorial test optimizer process.

Fig 4 shows proposed work. First analyze the existing system that a combinatorial situation exists. As we discussed earlier, any system that has factors and levels can be considered as a combinatorial problem. The next step is to apply our algorithms to find the optimized test combinations using combinatorial optimizer. There are plenty of tools that exists using OA strategies, covering pair-wise combinations. The concept used here covers single, double, triple mode combinations as well as take care of infeasible combinations. No tool will exactly identify the test combinations that we need. We must manually analyze and add few more test cases based on our gutfeel that would reveal defects directly. That forms our optimized test combinations. A typical real time project always has its own top priorities considering the risk factors in the organization analyzing the factors and levels. We have a provision to provide the prioritized factors and levels and ask to give us a customized combination of test suites. There can be one or more needed test combinations that default OA strategies may omit which we may need, or our client would have asked that combination specifically. Such combinations we can be able to add having a proven expertise and inquire the tool to account the same while creating the optimized combinations.

6. Results and Discussion

6.1 Application – 3:

We have analyzed a real retail supermarket project billing possibility. A customer who purchases products in this supermarket must answer the following questions during billing at the counter. The factor "Article category" indicates the type of purchased item. Meaning assumptions include 2 levels – Custom (item given in loose based on buyer's need) and Box up (packaged item/sealed item). The next factor is "Price Cut". Is price cut or discount applicable for the item purchased? It accepts 2 levels as answers – Yes and No. The third factor is "Excise Type". What kind of tax is applicable for the buying item? Possible answers can be the identified levels like: VAT, Duty, Sales etc., The last factor can be "Payment Mode". What is the possible mode of payments accepted in the supermarket for the item purchased? The applicable levels could be Cash, Credit Payment, Debit Payment, Sodexo meal vouchers.

Table 3: Factors & Level Abstraction

S.No	Factor Name	No. of Levels	Level Name			
1	Article category	2	Custom	Box up		
2	Price Cut	2	Yes	No		
3	Excise Type	3	VAT	Duty	Sales	
4	Payment Mode	4	Cash	Credit	Debit	Sodexo

:

Once factors and levels are identified, selecting the precedence factor is quite important. At times, customer may prefer certain factors of top priority and need those factors to appear a greater number of times in the optimized test suite. Precedence factor selection decides the level chosen as priorities, will appear a greater number of times in the optimized output shows in table.3

6.2. Optimized Outputs with Custom Precedence Factors:

The sample output shows table.4. The optimized output generated for "Custom" article category with greater precedence. We can be able to see more test case combinations have "Custom" category in it.

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023

Table 4: Optimized Output- Custom

SELECTED PRECEDENCE FACTOR: Article Category, LEVEL: Custom				
S.No	**Article Category**	**Price Cut**	**Excise Type**	**Payment Mode**
1	Custom	Yes	VAT	Cash
2	Box up	No	Duty	Cash
3	Custom	Yes	Sales	Cash
4	Custom	No	Sales	Credit
5	Box up	Yes	VAT	Credit
6	Custom	Yes	Duty	Credit
7	Custom	Yes	Duty	Debit
8	Custom	No	VAT	Debit
9	Box up	Yes	Sales	Debit
10	Custom	Yes	VAT	Sodexo
11	Box up	No	Duty	Sodexo
12	Custom	Yes	Sales	Sodexo

The sample output below shows the optimized output generated for "Box up" article category with greater precedence. We can be able to see more test case combinations have "Box up" category in it.

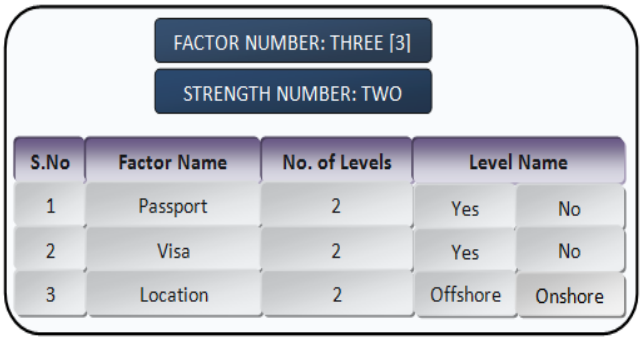
Table 5: Optimized Output- Box up

SELECTED PRECEDENCE FACTOR: Article Category, LEVEL: Box up				
S.No	**Article Category**	**Price Cut**	**Excise Type**	**Payment Mode**
1	Custom	Yes	VAT	Cash
2	Box up	No	Duty	Cash
3	Box up	Yes	Sales	Cash
4	Custom	No	Sales	Credit
5	Box up	Yes	VAT	Credit
6	Box up	Yes	Duty	Credit
7	Custom	Yes	Duty	Debit
8	Box up	No	VAT	Debit
9	Box up	Yes	Sales	Debit
10	Custom	Yes	VAT	Sodexo
11	Box up	No	Duty	Sodexo
12	Box up	Yes	Sales	Sodexo

6.3. Optimized Outputs with Infeasible combinations:
Consider the combinatorial input set as below table.6:

Table 6: combinatorial input

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023



This generates a pair-wise combinatorial output as shown below table 7.

Table 7: pair-wise combinatorial output

S.No	Passport	Visa	Location
1	Yes	Yes	Offshore
2	No	No	Offshore
3	Yes	No	Onsite
4	No	Yes	Onsite

The algorithm reasonably considers the factors and levels provided as statistical inputs for the combinatorial output generation, rather than considering the real meaning of the inputs provided. The test cases predict the failures after the deep learning classification. So, the APFD value has calculated on deep learning network after the splitting of failures into two sub-datasets based on their running time. Old failures are used as a feature used in deep neural network and new failures has used for testing. The split is done in a unique fashion, so, the failures in two sets of similar size where both sets contain about 50% of all failures. In the above analysis, the combination with Passport – No and Visa – Yes is irrelevant. Such a combination generation can be avoided much in advance by providing the combination as an infeasible input. Considering the infeasible input, the optimized output for this example is as below table 8:

Table 8: Infeasible input, the optimized output

S.No	Passport	Visa	Location
1	Yes	Yes	Offshore
2	No	No	Offshore
3	Yes	No	Onsite
4	Yes	Yes	Onsite
5	No	No	Onsite

7. Conclusion

The proposed work is created and successfully realized. We understood from various project hassles, introduced the concepts of "precedence" where the customer can choose any factor/level as priority before generating the optimized combinations and "infeasibility" to omit certain combinations which doesn't have meaning to the real world going one step ahead with other existing OA strategy methodologies and random forest algorithm. Precedence factor selection decides the level chosen as priorities, will appear a greater number of times in the output. Selecting infeasible combinations of factors and levels will skip the combination in output. The entire methodology can also be further extended implementing various strengths like "Strength 1" which focuses on covering only single mode combinations, "Strength 2", "Strength 3" covering double mode, triple mode at one shot taking the same combinatorial case studies.

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023

This will make the functionality more flexible to try out different possibilities in test design which enables making quick decisions in test design creation process.

References

[1]. Greiler, M., van Deursen, A., Zaidman, A. (2012a). Measuring test case similarity to support test suite understanding. In Proceedings of the 50th International Conference on Objects, Models, Components, Patterns (TOOLS'12), Springer-Verlag

[2]. Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. Softw Test. Verif Reliab, 22(2), 67–120

[3]Anand, S., Burke, E. K., Chen, T. Y., Clark, J. A., Cohen, M. B., Grieskamp, W., Harman, M., Harrold, M. J., McMinn, P. (2013). An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software, 86(8), 1978–2001

[4]. Amannejad, Y., Garousi, V., Irving, R., Sahaf, Z. (2014). A search-based approach for cost-effective software test automation decision support and an industrial case study. In 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE (pp. 302– 311).

[5]. Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., Yoo, S. (2015). The oracle problem in software testing: a survey. IEEE Trans Software Eng, 41(5), 507–525.

[6]. Epitropakis, M.G., Yoo, S., Harman, M., Burke, E.K. (2015). Empirical evaluation of Pareto efficient multi-objective regression test case prioritisation. In Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12-17, 2015, ACM, (pp. 234–245).

[7].Flemström, D., Sundmark, D., Afzal, W. (2015). Vertical test reuse for embedded systems: A systematic mapping study. In Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA'15)

[8]. Marijan, D. (2015). Multi-perspective regression test prioritization for time-constrained environments. In 2015 IEEE International Conference on Software Quality, Reliability and Security, QRS 2015, Vancouver, BC, Canada, August 3-5, 2015, IEEE, (pp. 157–162).

[9]. Wang, R., Jiang, S., Chen, D. (2015). Similarity-based regression test case prioritization. In The 27th International Conference on Software Engineering and Knowledge Engineering, SEKE 2015, KSI Research Inc. and Knowledge Systems Institute Graduate School, (pp. 358–363).

[10].Noor, T.B., & Hemmati, H. (2015). A similarity-based approach for test case prioritization using historical failure data. In 26th IEEE International Symposium on Software Reliability Engineering, ISSRE 2015, IEEE Computer Society, (pp. 58–68).

[11].Strandberg, P.E., Sundmark, D., Afzal, W., Ostrand, T.J., Weyuker, E.J. (2016). Experience report: automated system level regression test prioritization using multiple factors. In 27th IEEE International Symposium on Software Reliability Engineering (ISSRE'16).

[12].Strandberg, P.E., Sundmark, D., Afzal, W., Ostrand, T.J., Weyuker, E.J. (2016). Experience report: automated system level regression test prioritization using multiple factors. In 27th IEEE International Symposium on Software Reliability Engineering (ISSRE'16).

[13].Wang, S., Ali, S., Yue, T., Bakkeli, Ø., Liaaen, M. (2016). Enhancing test case prioritization in an industrial setting with resource awareness and multi-objective search. In Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume, ACM, (pp. 182–191).

[14].Annibale Panichella_, Fitsum Meshesha Kifetewy, Paolo Tonellay _SnT ,"Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets"- DOI 10.1109/TSE.2017.2663435, IEEE Transactions on Software Engineering.

[15].Marco Autili, Antonia Bertolino, Guglielmo De Angelis, Davide Di Ruscio, and Alessio Di Sandro ,"A Tool-Supported Methodology for Validation and Refinement of Early-Stage Domain Models-IEEE transactions on software engineering, VOL. 42, NO. 1, January 2016.

[16].Matthew B. Dwyer and David S. Rosenblum," Editorial: Journal-First Publication for the Software Engineering Community", IEEE transactions on software engineering, vol. 42, NO. 1, January 2016.

[17].Alessandro Marchetto et al.," A Multi-Objective Technique to Prioritize Test Cases ", IEEE TRANSACTIONS". VOL. 42, NO. 10, OCT 2016.

[28]. Sepehr Eghbali and Ladan Tahvildari " Test Case Prioritization Using Lexicographical Ordering/IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 42, NO. 12, DECEMBER 2016.

[19]. Dan Hao, Lu Zhang, Lei Zang, Yanbo Wang, Xingxia Wu, and Tao Xie, " To Be Optimal or Not in Test-Case Prioritization", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 42, NO. 5, MAY 2016.

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023

[20].Antonio Filieri, Giordano Tamburrelli, and Carlo Ghezzi, Fellow, IEEE," Supporting Self-Adaptation via Quantitative Verification and Sensitivity Analysis at Run Time", IEEE transactions on software engineering, vol. 42, no. 1, January 2016.
[21].M.Sangeetha,S.Malathi,Test Suite Sequencing Using Dependency Structure Matrix","Advanced Research and Engineering"March-18.
[22].Daniel Flemström., Pasqualina Potena., Daniel undmark.,Wasif Afzal.,Markus Bohlin-2018- Similarity-based prioritization of test case automation

Doi: https://doi.org/10.54216/FPA.130111
Received: April 01, 2023 Revised: June 26, 2023 Accepted: September 04, 2023