Journal of Positive School Psychology http://journalppw.com
2022, Vol. 6, No. 3, 6361–6369

Using Deep Learning and Dependency Matrix Structure Perform Optimization and Increase the Accuracy Rate

M. Sangeetha

Research Scholar, Sathyabama Institute of Science and Technology,

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119, Tamilnadu, India/ Associate Professor, Department of Computer Science and Engineering, Panimalar Engineering College, Poonamallee, Chennai – 600 123, Tamilnadu, India.

S. Malathi

Professor, Department of Computer Science and Engineering,
Panimalar Engineering College, Poonamallee, Chennai – 600 123, Tamilnadu, India.

Abstract

Software Testing is a perceived software development life cycle methodology, which is used to test the software product quality by writing test logic that matches with customer requirements. Interactions/relationships between systems or sub systems or modules in an application are called dependency. We use software in different contexts like Aircrafts, Medical Equipment's, Stock exchanges, Space systems, banks, Machine production etc., Software also manage enterprises and their bonding to clients and suppliers. It also supports taking strategic decisions in business organizations. Reliability and performance of software is very crucial to consider for effective management of our systems. Earlier in the past, few techniques have been derived considering dependency structures in applications which enables to select test case prioritization both manually and using internal or open-source or commercial vendor based automated tools. This paper analyzes the application dependency structure algorithms to effectively plan the module sequences prioritize the optimized test cases using a novel Deep Diverse Prototype Forest Model by improving performance and efficiency.

Keywords: Prioritization, Dependency, Optimization; Medical Equipment; Test cases; Defects.

1. Introduction

Programming Engineering is a designing order that focuses on various phases of programming creation [2]. Programming isn't produced all in all unit, it has distinctive stages to be specific "Plan", "Investigate", "Outline", "Coding", "Testing" and "Execution. Effective Successful programming testing will add to the conveyance of solid and quality situated programming item. A quality item has more fulfilled clients, brings down support cost, and reduces rework and more exact and solid outcomes. The need for high reliability products in one hand, defective system/components on the other hand makes the software organizations to depend more on Software testing as a discipline to identify the defects in the software before it is deployed in the client environment.

All product building ventures includes devoted testing as a different stage. Due to quick innovation development and intensity in genuine client business, "time" and "spending plan" assume a noteworthy part and these two are the key achievement factors in venture culmination [4]. To lead the undertaking in more effective way, most extreme quality must be guaranteed in the meantime, limit the venture cost and decrease the conveyance time. Continuously, to expand the viability and proficiency of testing inside constrained arranged assets, powerful experiment prioritization considering the business requests can be performed. TCP is a procedure of sorting out or choosing the experiments in succession to expand the blame recognition rate at the soonest, which finds most basic deformities as prior as conceivable in the product testing life cycle. Testing expenses will go down if we distinguish absconds as right on time as conceivable in the testing life cycle.

Early detection of defects in software product reduces "rework costs" as it saves time and

effort in the overall testing life cycle. Say for example: We have the following software development phases namely: Requirements, Design, Coding, Testing and Deployment. Bugs/Defects can introduce in any of the said phases. Identifying the right defect at the right stage is a critical task for the testing team. It means defects getting originated from requirement phase must get identified and fixed in requirement phase it. This is technically known as "Stage Containment" – identifying the defect at the right test stage.
This validates one of the 7 principles of testing – "Defect Clustering" or "80-20 rule". This technique believes the principle that 80% of defects are revealed in 20% of the overall code. This technique is suited for projects where we have multiple test partitioned identified with different expected outcomes.

State transition
It is used to determine the invalid system transitions. This testing technique is used in the application design. State Transition testing technique represents the system behavior as finite number of states. This technique is useful when analysis and system testing are dependent on previous and current behavior, previous and current states. The test lead prepares it when the testing timelines are defined. It chooses the starting point, where we begin examining an application or system. Understand all the states that an object or user can be in. Identify the transitions (events, conditions, actions) that are applicable to each state. Build a table or diagram to describe the states transitions.

2. Related Work

Test design optimization has been proposed in various studies. It analyzed by various people by different stages. Wang, S.¸et al, 2016 developed prioritization process by using fuzzy logic in which they used analytical hierarchy method and othe model combined to perform optimization. Equivalence partitioning focuses on partitioning the input space into a small number of partitions agreeing to the requirements (Raj Kumar, et al, 2018). BVA focuses to test these bounding edges to identify maximum errors writing the minimum number of test cases (Strandberg, et al, 2016). This method is preferred for projects where there are plenty of equivalence class partitions to test where each partition has varied expected outputs.
In multi objective algorithms author discussed about seeding strategies are used to solve test case optimization. Greedy approach ((Jatana, Nishtha, et al, 2021) developed for yielding optimized test suite.

3. Methodology

Sequencing the test modules considering the functional dependencies is crucial to plan testing. It is very important for organizations to sequence their test activities for effective testing. This ensures optimized testing in terms of test coverage, achieving maximum test coverage with minimal test effort. Planning this reduces cost of quality (COQ), ensuring we are doing the right things which is required for us to achieve with minimal effort. The goal of testing is to ensure the overall solution, its components and its assembly of components meets the established functional and technical requirements. A clear set of requirements from the customer ensures the following benefits: It helps ensure that QA covers all customer expectations; it helps to develop quality solutions. It enables to clarify questions related to the requirements in the initial planning stages. It helps to reduce defects, when requirements are provided in the initial stages. It also helps understand the scope of requirements. In planning stage, the testing strategy defines the major aspects of the test effort and outlines the coverage to key sponsors. The testing strategy fills in as a contribution to the test methodologies and plans for the phases of testing. The test strategy is captured within the delivery strategy composite created during Planning. We use the testing strategy throughout the analysis, design, and build stages as a starting point for creating initial test approach documents for specific test stages. For each test stage, the test strategy provides: Scope, high-level objectives, tasks, and responsibilities, test design techniques to be used, test types to be carried out, high-level entry and exit criteria for verification, validation, and test tasks. The Test Strategy deliverable records the general technique for testing the application, specialized engineering, and preparing and execution bolster. These deliverables are made to characterize the test stages to be directed in the undertaking. The test approach tends to every

single significant part of the test stage. The first step in any test stage is to develop a test approach. Testing Metrics provides planned rates for all testing metrics specified in the Testing Overall Approach document. Configuration Management includes details of the configuration management and version control process and tools that are used for code migration and release. Test Environment specifies logical and physical diagrams of the proposed test environment. Test Tools includes details of all test tools to be employed for the test stage. Assumptions, Issues, and Risks use tables to show all assumptions, issues, and risks affecting the test stage. Include the name and role of each person involved with the test stage. Hazard is an issue that could cause some misfortune or debilitate the achievement of our task, however which hasn't happened yet or may never occur by any means. These potential issues may adversary affect the cost, timetable, or specialized accomplishment of the task, the nature of our product items, or undertaking camaraderie. It can run from cataclysmic (loss of a whole framework; death toll or lasting incapacity) to insignificant (no framework harm or damage). The probability program blame will bring about an effect on business. Usually, testing is attempts to focus test activities on the functional area with greatest amount of risk. Some benefits of this include improving test efficiency and effectiveness and improving the quality of requirements. It likewise expands test viability while possibly bringing about less tests and diminishing expense. Hazard Based Testing Approach organizes testing where testing exercises center around the most essential tests that will relieve the most elevated measure of hazard. It is prescribed to include the fitting partners in gatherings given that they will close the test situations. Accumulate all prerequisites simultaneously (utilitarian) and obviously distinguish the out of degree things. Functional and Non-functional requirements are unique. Useful prerequisites center around what the application must do while the Non-utilitarian necessities expresses the operational execution criteria, for example, framework throughput and reaction time. In design phase, we have lots of black box test optimization techniques. Error Guessing is an Experience-based testing technique where the previous experience of a tester is used to find the application defects. The Tester will guess the errors in the application and has no specific rules. In this technique, the Testers will try guessing the situations where the application will fail. For example, errors like clicking on the Submit button before entering the mandatory fields, entering number in Name field etcetera. In this technique, the previous experience of a tester is used to find the application defects.

Ensuring Performance is very important task in Software Engineering process [10]. Performance Engineering is a systematic process by which the performance of an application is ensured by planning and design, Modeling and forecasting, Testing and tuning. It helps clients to improve customer satisfaction and to manage development costs by building performance into applications as they are being developed or by alleviating performance problems in existing applications [10]. Organizations striving for high performance depend on optimal systems and applications performance. Stage Containment is a major underlying principle of the V-Model. It ensures identifying the right defects at the right stage ensuring cost, schedules, timelines are all intact. The objective of stage control is to recognize issues in the framework amid advancement before they are passed to the following stage. This helps incorporate quality with the framework. The process of determining the origin (originating stage) of a defect is termed root cause analysis. This stage containment process highlights problem stages and can be used to continually improve the testing process. In this way, Stage Containment can also help fix the process that originally caused the errors or the process that should have prevented them from occurring. Discovering issues or blunders in the stage they begin/happen in is imperative since issues turn out to be costlier and harder to settle later in the undertaking life cycle. Advantages of stage control incorporate: Less imperfections, limited expenses and exertion for settling issues. Therefore, by enforcing stage containment, a project can minimize this cost and reduce the number of residual problems in the finished solution.

Fig 1 represents a project using stage containment has a defect-finding pattern like that shown in below:

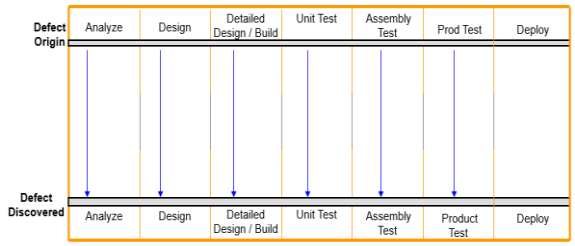


Figure 1: With Stage Containment.

Without stage containment, a project could see a defect-finding pattern like the one shown below:

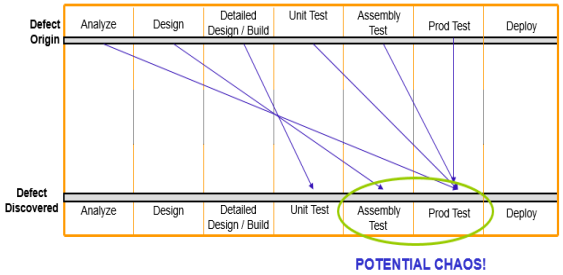


Figure 2: Without Stage Containment

The above fig 2 shows how problems can disrupt the accuracy of project plans, making schedules more difficult to keep, and increasing costs

There is no better embarrassment than customer identifying defects at production environment before project stakeholders do. This will create a huge business loss, customer trust, reputation etc. The example below fig.3 shows the worst case of stage containment where most of the defects are identified at production environment. The worst-case scenario where analysis, design, testing defects is all identified at production environment

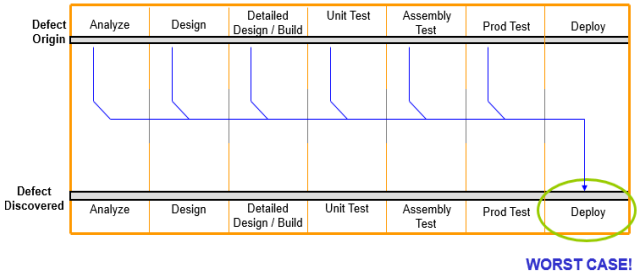
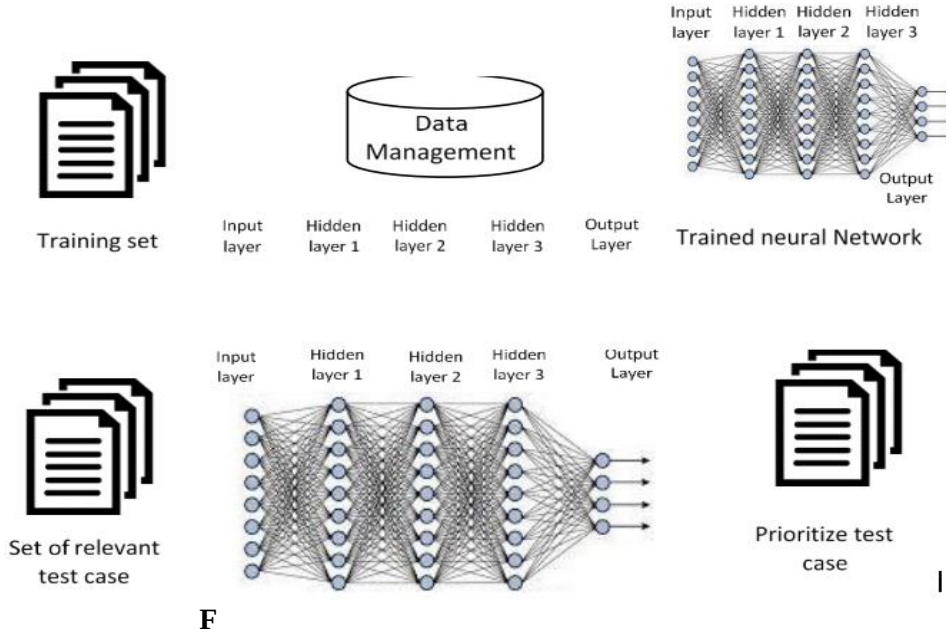


Figure 3. Without Stage Containment-worst case

Architecture Diagram

.

The following fig 4 shows the system architecture of testing life cycle phases – Planning & Designing which we are considering for effective test sequencing.



F

F ig.4. Architecture for Training and testing phase

The test design OATS optimization process is as follows: Analyze the existing system and understand whether combinatorial situations exist, since OA strategies cannot be implemented for any level of test optimization. Try identifying factors and levels which acts as a sole input for any strength of OA test optimization. Apply combinatorial test

optimizer to generate the optimized test combinations. Any test optimization tool will not exactly formulate the precise test optimization matching set that any project/client requires. So, we analyze the outcome given by the combinatorial optimizer and decide on the amendment factors. A typical real-time project always has its own top priorities on analyzing the factors and levels. It can be the looked-for input from the customer or end-user or from the project team analyzing many superiority factors like risks, budget, time etc. There is a provision to provide the prioritized factors or levels or both as a input and asking the tool to generate the optimized combinations. Any test optimization tool in fact is always a tool, not more intellectual than an experienced human. There can be one more needed test combinations that either the tool might have omitted in the combinatorial output or it doesn't fall under the pair-wise or triple-wise combination so that the tool might keep away from combination generation. Such combinations can be added from the proven expertise and inquire the tool to take account of the same in the optimized combination. The objective of test design phase includes: Producing an application design pack for the solution, Derive the application design from and ensure the consistency of the design deliverables with the application requirements, the use cases, business rules, and other requirements-related deliverables, ensure that the application design contains enough information for the builds.

4. Implementation

Dependency Structure matrix Algorithm

Algorithm 1: Update Routing Table

Input: Modules
Output: Return Path

Public Function FindPath (ByVal dicModules)
Set value of dicModules to dicNewModule.
For Each intModule in dicModules
If dependancy of intModule and intStartModule is less
than of intModule and intCurrentModule Then
Assign intModule to intCurrentModule
End If
Next
Add intCurrentModule to arrPath
Set flgPath as False
While flgPath as False
For Each intModule In dicModules
If dependency of intCurrentModule and
intModule is less than of 150 Then
Add inModule to arrNextModule

End If
Next
For Each intModule In arrNextClosestModule
If summation of dependency value of
intCurrentModule & intModule and intModule
& intEndModule is less than summation of
dependency value of intCurrentModule &
intNextClosestModule and distance between
intNextClosestModule and intEndPoint Then
Assign intModule to intNextClosestModule
End If
Next
Add intNextClosestModule to path
Remove intNextClosestModule
Assign intNextClosestModule to intCurrentModule
If dependency of intCurrentModule and
intEndMdule is less than of 200 Then
Set flgPath as True
End If

End While
Return path
End Function

Algorithm 2: Get Dependency Value between Modules

Input: Set First Module value as intModule1 and set Second module value as intModule2
Output: Dependency value between two modules as double.
Find value difference between intModule2 and intModule1.
Calculate the power value.
Return the square root of power value.

Flow Chart:

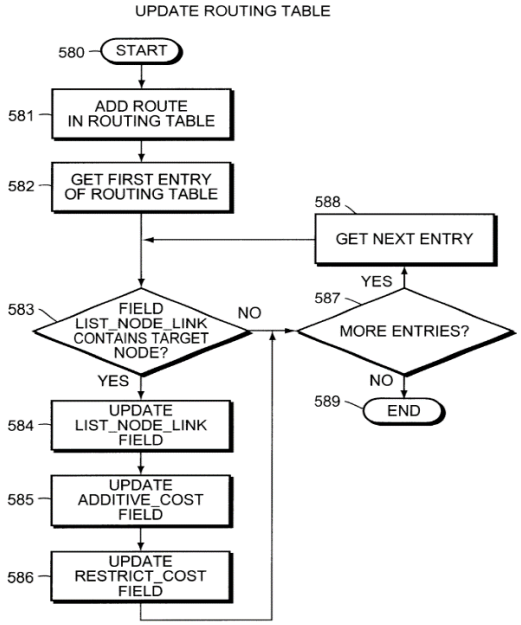


Fig 5: Flow chart for update Routing Table

5. Results & Discussions

OUTPUTS FOR AS EARLY AS POSSIBLE TEST SEQUENCING (AEAP): BEST SEQUENCE

Cyclic Blocks:

This output indicates separately the list of cyclic dependent modules and assigns a block number for every block. The table 2 output would be helpful in a realistic project where we have "n" number of two-way dependent modules which has to get treated exceptionally equated to other hooked on modules. Cyclic Blocks are outputs of modules that have interdependency. DSM lists out the modules which are cyclically dependent and assigns a block number (reference number) and lists in this cyclic block output. The list of all cyclic blocks and the modules belonging to them is displayed in the below output:

CYCLIC BLOCKS		
Blocks	Modules list -->>>	
301	7	6

Table 1: List of Cyclic Block for AEAP

Leveling Table:

This table 3 output table is extremely helpful identifying the number of levels in the project and the modules aligned to the same level. Number of levels indirectly indicates the complexity of the project that enables the testing team to plan effectively on how to proceed with testing considering time and budget constraints. This output defines the seclusion of modules/cyclic blocks into hierarchies, it defines the best sequence to be followed and it also tells which modules can be executed in parallel saving time and effort. Leveling table outputs the modules according to their levels of early sequencing. This enables us to identify very easily "n" module is in which level. It segregates the modules/blocks into hierarchies. It suggests the best and worst case sequence to be followed. It also suggests the modules that can be executed in parallel.

LEVELING				
Level	Modules / Blocks			
1	1			
2	2	3	4	5
3	301			
4	8			

Table 2: Leveling Table for AEAP

As a best sequence among modules, we can see modules tagged in level 1 can be tested first; level 2 modules can be tested next and so on. Also, we can see that modules tagged under level 2, i.e., modules 2, 3, 4 and 5 can be executed parallel .

OUTPUTS FOR AS LATE AS POSSIBLE TEST SEQUENCING (ALAP): WORST SEQUENCE

Leveling Table:

This table 3 output table is extremely helpful identifying the number of levels in the project and the modules aligned to the same level as worst sequence. Number of levels indirectly indicates the complexity of the project that enables the testing team to plan effectively on how to proceed with testing considering time and budget constraints. This output defines the seclusion of modules/cyclic blocks into hierarchies, it defines the best sequence to be

followed and it also tells which modules can be executed in parallel saving time and effort.

Table 3: Leveling Table for ALAP

LEVELING				
Level	Modules / Blocks			
1	1			
2	3			
3	301			
4	2	4	5	8

Table 3: Leveling Table for number of levels

As a worst sequence among modules, we can see modules tagged in level 1 can be tested first; level 2 modules can be tested next and so on. Also, we can see that modules tagged under level 4, i.e.., modules 2, 4, 5 and 8 can be executed parallel after carrying out testing all other modules.

Partitioning Table:

This table 4 output table arranges the input according to the leveling table and it provides the relationship between module numbers and levels to which the modules are tagged under worst possible test sequence of modules. How much we can relax between testing of all modules is defined in "As late as possible sequencing". This ensures that we are within quality, budget and time. This also helps for proper risk management

PARTITIONING (Expanded)

Levels	Block Number	Module Name	Module Number	1	3	7	6	2	4	5	8
1		Login	1	1 (1)							
2		Bank Account	3	1	3 (2)						
3	301	Database	7			7 (3)	1				
3	301	Fund Transfer	6		1	1	6 (3)				
4		Loan	2	1				2 (4)			
4		Credit Card	4	1					4 (4)		
4		Currency Conv.	5	1						5 (4)	
4		Account Summary	8				1				8 (4)

Table 4: Partitioning Table for ALAP

Deep learning-random forest algorithm (dl-rl) for black box test case selection

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.

Steps involved in DL-RL:

There are mainly two operations - training phase, and testing phase. The data i used to perform test case prioritization is stored in a data management system and increase the accuracy rate by comparing the previous algorithms. 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 .

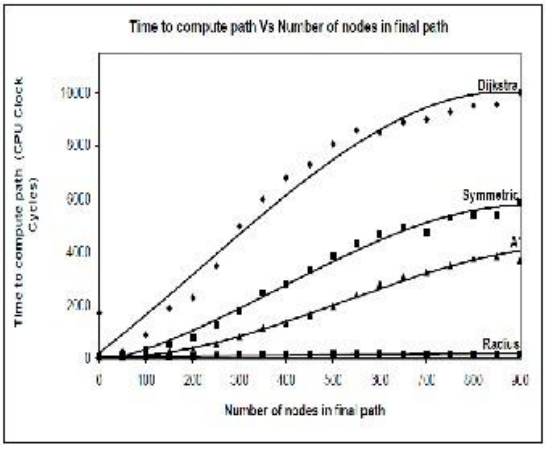


Fig 6: Time to Compute Path

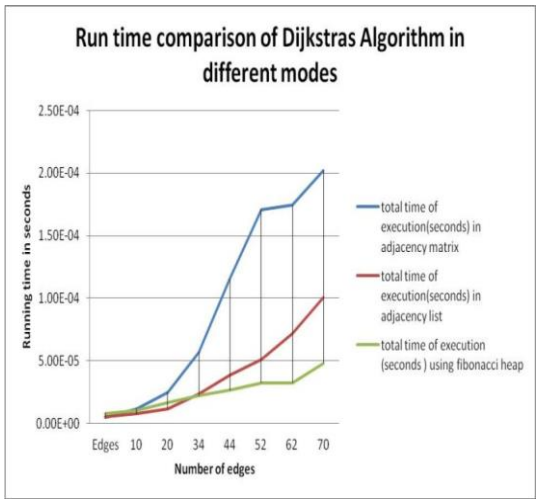


Fig 7: Run time Comparison of Dijkstra's Algorithm

The objective of this technique is to ensure test sequencing as early as possible in the test plan stage itself on the basis of dependency structure algorithm which will increase the error detection rate drastically considering key project constraints of budget, time in mind comparing with the existing techniques.

Comparison analysis

By using DDPF (Deep diverse prototype forest) model Table 5 shows the accuracy of the proposed method and existing methods. Multiple factor-based prioritization is based on the factor that causes the fault occurs in the test cases. But the accuracy of multiple factors was not efficient to select the best test cases, so the Paratoo algorithm has introduced with weigh update function. The accuracy achieved by multiple factors, Paratoo, hybrid, optimized method was 86.54, 83.54, 89.45, 85.34, and 93.4.

Method	Accuracy (%)
Multiple factor based prioritization	86.54
Paratoo Algorithm	83.54
Hybrid Algorithm	89.45
Optimized and unique method	85.34
DDPF	93.4

Table 5. Accuracy rate of various methods.

6. Conclusion

The proposed work analyze the various phase of project and using application find the shortest path between the node of modules. By doing easily identify the best and worst case sequence of running test cases and perform the test case optimization. Also using deep learning and fuzzy logic the proposed technique increases the accuracy rate. Implementing DDPF model proposed work prove the decent raise in accuracy as well as reduce the time complicity. In future this can be extended by considering other factors.

References

[1].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).

[2].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).

[3].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).

[4].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.

[5].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.

[6].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.
[7]. Sepehr Eghbali and Ladan Tahvildari " Test Case Prioritization Using Lexicographical Ordering/IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 42, NO. 12, DECEMBER 2016.
[8]. 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', ICSE 2016, pp. 182–191..
[9].M.Sangeetha,S.Malathi,Test Suite Sequencing Using Dependency Structure Matrix","Advanced Research and Engineering"March-18.
[10]].M.Sangeetha,S.Malathi Sequencing of Modules and Prioritization of Test Cases using Dependency Structure Matrix: Survey, IEEE 6th International Conference on smart structures and systems ICSSS 2019.
[11].Daniel Flemström., Pasqualina Potena., Daniel undmark.,Wasif Afzal.,Markus Bohlin-2018- Similarity-based prioritization of test case automation.
[12].].M.Sangeetha,S.Malathi Identifying AEAP ALAP Sequences For Optimization Using Dependency Structures, INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 9, ISSUE 02, FEBRUARY 2020.