Identifying AEAP ALAP Sequences For Optimization Using Dependency Structures

M.Sangeetha,Dr.S.Malathi

Abstract: Software Testing is a process of analysis whether a system or a product complies with needs of customer requirements. It is mainly performed by testing team using different tools and techniques and the main target is to identify different behavior in the software project and to make sure quality. Generally testing is not done completely, instead it focuses on different test stages in testing like Unit, Integration, System, User Acceptance etc., and before launching it to the real world testing confirms the performance of the product. Testing also prevents product failure or wastage of cost. Access the quality of the final product delivered to the customer is the main aim of testing. Different phases of Testing life cycle focuses on – Test plan, Test design, Test execution, Defect reporting and tracking it to closure etc., test designing is writing of test cases based on requirements are the main blocks of testing. Very crucial in this testing life cycle is writing effective test cases in minimum time period. Criticality and risks is a key task of tester to sequence the test cases based on the priority of test case generation. Proposed methodology is to improve the detection of fault at the earlier phase like planning. This methodology provides the sequential order in as per the dependency of modules. In this paper we mainly identifying the modules along with cyclic blocks to be tested in sequence during the planning phase and prioritize this with OATS techniques and dependency structure.

Index Terms: prioritization, faults, test cases, cyclic blocks, sequence.

1. INTRODUCTION

Software engineering is engineering discipline that focuses on software product stages Plan Analyze design coding testing and implementation. Software testing is a process of analyzing software to detect the difference between existing system and actual requirement. Difference is called as defect or bugs. Software Testing is a separate study under Software Engineering which has a detailed life cycle process that includes “Feasibility analysis”, “Test Plan”, “Test Design”, “Test Execute” and “Defect reporting and tracking”. It is an investigation conducted to provide customers with details about quality of the product. Highly qualifies staffs ensure that software product built on time within budget with respect to attributes such as reliability, correctness ability and usability to satisfies the customer requirement. Testing Process, in the software engineering, is the set of methods, practices, standards documents, activities, policies, and procedures that software engineers use to preserve a software system and its associated artifacts, such as project and test plans, design documents code and manuals Process has been developed as a series of phases, procedures, and steps that result in the production of a software product embedded within the several processes. Validation is the process of evaluating a software system or components during or at the end of, the development cycle in order to determine whether it satisfy the specified requirements. It is usually associated with traditional execution-based testing that is exercising the code with test cases Verification is the process of evaluating a software system or component to establish whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

- *M.Sangeetha is a research scholar in Sathyabama institute of Science and Technology,Chennai,India.Mail-id: sangeetharemi@yahoo.co.in.*
- *Dr.S.Malathi working as a professor in panimalar engineeringcollege,Chennai,India.Mail-id: malathi_raghu@hotmail.com*

Effective software testing will help to the delivery of reliable and quality-oriented software product. If the product is qualified then it has more satisfied users, lower maintenance cost, and more accurate and reliable result. All software projects include dedicated testing as a separate team. A test specialist is one whose education is based on the principles, practices and processes that comprise the software engineering discipline, and whose specific focus is on one area of that discipline – software testing. A test specialist who is trained as an engineer should have knowledge of test related principles, processes, measurements, standards, plans, tools and methods, and should learn how to apply them to the testing tasks to be performed. Time and budget plays major role for the success of task completion. For the successful project quality must be ensure the cost of the project and minimizing delivery time. Day by day there is rapid growth in technology increases the demand for high quality software. Test case prioritization is a process of organizing or selecting the test cases in sequence to increase the fault detection rate at the earlier stage, which helps to find critical defects as earlier as possible in the software testing life cycle. Drastically testing costs will get reduced if we identify defects as early as possible in the testing life cycle. Test Suite Optimizer developed for the purpose of testing pair – wise combinations of the test cases during the design phase. It ensures minimal test case designing with maximum coverage of defects for optimization of test cases. Test sequencing is a major challenge that will lead to diminished quality of work product. Improper test sequencing affects the proposed test schedule, planned budget. When budgets are not properly estimated, it becomes expensive. The aim of usability testing is to observe people using the product to discover errors and need to improve the areas. It measures efficiency, accuracy, recall, and emotional response. Testing that validates ease of use, speed and aesthetics of the product from the user’s point of view. It is a process to identify discrepancies between the user interface of the product and the human user requirements, in terms of the pleasantness. Security testing is a process to establish that an protects data and maintains functionality as intended. Security testing as a term has a number of different meanings and can be completed in a number of different ways. As such a Security categorization helps us to understand these different approaches and

meanings by providing a base level to work from. Penetration Test simulates an attack by a malicious attack. Building on the previous stages and involves exploitation of found vulnerabilities to gain further access. Because of this approach will result in an understanding of the ability of an attacker to gain access to confidential information, affect data integrity. The inspiration of this work is to automate the combinatorial test inputs and optimized combinations for test case generation. This will ensure effective test planning for all project constraints like schedules, budget by using CODEC and OA Strategies. For this work details the new features required on suite optimization. This includes building it more user friendly, making it up-to date to the database, making the application more intelligent which includes ability to decide on infeasible and combinations that are prioritized; also improving application's the reporting functionality. In this paper we observe the test case prioritization primarily based at the inbuilt structure of dependencies between tests and test cases which known as dependency structure prioritization. Those dependencies reproduce the machine itself. It's defined that ordering test executions based totally at the complexity of communication between tests can growth the fault identification rate in comparison with arbitrary test ordering. Test Case Prioritization is a method of ordering test cases. Average Fault detection rate must improve. Dependency Structure means Interactions or relationships between modules in an application are called dependency. There are two techniques 1. First one open dependency structure in which a functional dependency among test cases T1 and T2 represents that T1 must be achieved before T2: Another one closed dependency structure between test cases T1 and T2 represents that T1 must be executed immediately before T2.Some interaction cannot happen until unless some other interactions occur first in software sequences of iteration between modules.

2 .RELATED WORK

In existing system it's too difficult to recognize and find out the exact sequence of test cases during design phase. Still there is no effective mechanism to improve the efficiency of test case prioritization. In efficient people in software development team which delay the deployment of the product. In regression testing to simulate the expectations no separate team for that. Due to the lack of focus on configuration management process will lead delays because of changes. Test design challenges forecasting test cases related to test coverage in order to prove optimization. Other approaches among various test cases that use [12] derived dependency structures in a test suite are performed by manually. It took more time to test the job as well as tests the system time by a tester. Hai dry's approach is used [1] to find total count of dependents for each test case using DSP (Dependency Structure Prioritization) volume is not giving proper result. Hence, proposed methodology which identifies the best sequence of test cases and worst sequence of test cases in a module for testing using dependency structure matrix during the testing life cycle in planning phase. This approach will overcome all existing research techniques because it optimizes the test cases in planning phase itself. Previously Alessandro Marchetto et al.,[4] develop a technique applying metric based approach which automatically discover faults and reduce the cost of test cases. It is capable of giving prioritization not in the planning phase but in the design phase. Also Dan Hao, Lu Zhang et al.,[6] proposed an ideal optimal test-case prioritization technique that schedules the order of execution of test cases based on faults that are detected using optimal coverage based test-case prioritization as an integer linear programming (ILP) problem in design phase to produce the optimized result. Existing result proves that the combination of both techniques improves SPL testing effectiveness [13] but not identifying the proper test sequence.

3. PROPOSED WORK

When we plan testing, it is very important to define the order of test modules. These test modules are sequenced considering the functional dependencies of one module over other. Testing architecture basically consists of two core phases – Planning and Design.

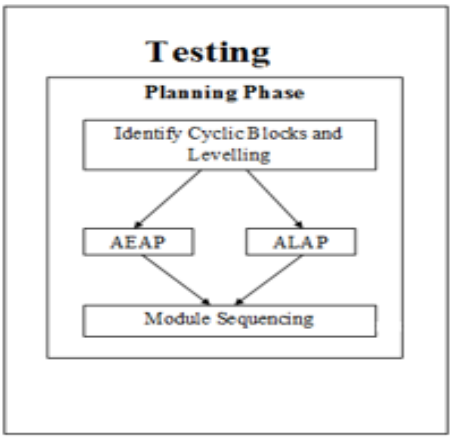

Fig1: *Architecture diagram for Sequencing in Planning phase.*

Ordering of test activities or test sequencing is a process of arranging all the modules in the application that has come for testing at the earliest possible in the overall test life cycle. This test sequencing or test ordering results in great reduction of time and cost. The main challenge is to effectively plan and sequence the order of modules for testing in the test plan stage itself rather than planning the same during the next stage – test design. When we plan the sequence of modules for execution during test plan phase, it minimizes the cost, effort and time thus minimizing the overall cost of quality of the project (COQ). This approach drastically minimizes the cost involved in rework, saves time and effort when we equally compare the other available test prioritizing mechanisms. Moreover, all test prioritization techniques focus only from the test design stage and none we have targeting from test plan stage of testing. CODEC identifies all the available cyclic blocks, leveling information of modules, which modules are tagged to what level, identifies the possible available critical paths, as early as possible (AEAP) sequencing and as late as possible (ALAP) sequencing information. The core part of CODEC is to identify the cyclic blocks among the modules – identifying all cyclic dependencies i.e.., A module is dependent on B module and vice versa. And it enables the tester to handle these dependent modules with extra care and effort. Leveling information is a helpful outcome to basically understand the complexity of the application. An application with 5 levels is simple, whereas an application with 500 levels is complex. This output enables the tester to get an idea on

the application complexity well in advance during the planning stage thus helps to plan testing effectively. Tagging information will be a time saving outcome which enables a tester to identify very easily that a module is tagged to which level. CODEC ensures identifying the statistical inputs, putting it in a dependency structure matrix considering the module dependencies. This produces early possible and late possible sequence outcomes which we define as early as possible sequence (AEAP) and as late as possible sequence (ALAP). As early as possible sequence is the optimized approach for sequencing the test modules as it saves time, effort, cost and ensures minimal project risks. As late as possible sequence is the worst-case approach that provides a boundary for testers to understand as how much we can stretch completing the projects with utmost schedule deviation ensuring maximum risks. As early as possible sequence testing and As late as possible sequencing testing techniques are the two extreme test sequence strategies that enables a tester to understand the best and worst sequence times understanding the risk factors. This enables the tester to plan well in advance considering proper sequencing of test modules during test plan itself. The tester will consider the dependencies of test modules, schedule, cost to arrive at the possible best test sequence for the proposed project. DSM - Dependency Structure Matrix – It is very important to be considered as a main statistical input for test module sequencing. This enables in which order the modules are to get executed and identifies the modules that must get executed concurrently. This identifies the order of module execution in the application for testing and enables the tester to understand which modules to keep under a single team of resources to work. It also identifies the test modules to get executed concurrently with no dependency conflicts. SCE –System Complexity Estimator is used to estimate testing efforts. This enables to distribute the test efforts over all test modules. It also identifies the dependent modules and provides distribution of efforts of all modules. This feature also enables to find the overall testing effort. For test effort estimation, we can use System Complexity Estimator. It mainly addresses the effort distribution of all test modules. If we have any past trend of similar test systems available, it is easy to determine the total testing effort involved. SCIM – System Change Impact Matrix is an important feature which is used for maintenance testing projects. It distributes the overall effort involved in testing when it encounters a change request (CR) from the client or business stakeholders. It also identifies the change impact for every new CR requested and it is very much helpful for maintenance projects to decide and prioritize the CR's. This is a very useful feature that enables a tester to understand the testing effort involved in implementing/testing the CR. It also enables to make decisions that are affects the original test effort plan. It also distributes the overall testing effort among all available modules considering the dependencies of modules. It also identifies the overall impact on the existing application by introducing every change request.

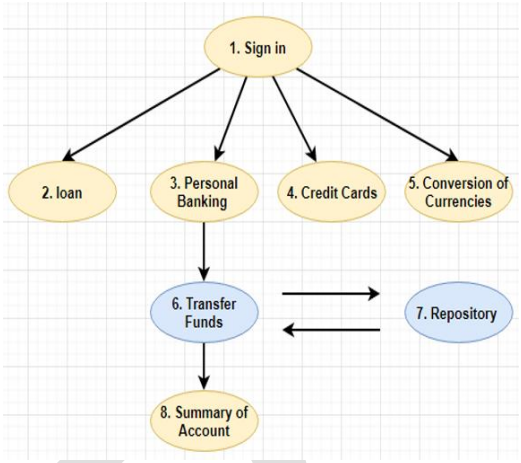

Figure 2: *Flow graph for Banking Transaction*

Fig 2 explains a dependency module structure which illustrates the core banking function. The core modules are Sign in, Loans, Personal Banking, Credit Cards, Conversion of Currencies, Transfer Funds, Repository, and Summary of Accounts. There is a dual dependency between Transfer Funds and Repository. It specifies there is a two-way relationship between these 2 modules, and they have coupled each other. It internally means that if we change transfer funds, it will affect the repository. Say for example: if we deposit cash to our account, it will increase our account balance in repository. Same way, if we change repository, it will reflect in Transfer funds module. This is called as "Cyclic Dependency" of modules. Other modules – Loans, Personal Banking, Credit Cards, Conversion of Currencies depends on Login. Login is the primary module which acts as an entry point to work with other modules/functionalities. This login module holds maximum dependency among all modules in this example.

1V. RESULTS AND DISCUSSION:

The input for the whole process is our Dependency Structure Matrix (DSM). It can be represented logically as per the below table. The self-dependent module information is mentioned in grey color cells. These dependent modules are identified automatically because every module is self-dependent on its own. We have to provide the external dependent module information in the table. It is represented as a logical "1" in the respective cells against the modules. Say for example, if we mention a "1" between Sign in and Loans, it means sign in and loan modules are dependent on each other.

Table 1: *Input – Dependency Matrix defining logical relationships*

		Signin	Loans	Personal Banking	Credit Cards	Conversion of Currencies	Transfer Funds	Repository	Summary of Accounts
Module Name	Module No.	1	2	3	4	5	6	7	8
Signin	1	1							
Loans	2	1	1						
Personal Banking	3	1		1					
Credit Cards	4	1			1				
Conversion of Currencies	5	1				1			
Transfer Funds	6			1			1	1	
Repository	7						1	1	
Summary of Accounts	8						1		1

Taking our example of core banking, it has various modules as like – Sign in, Loans, Personal Banking, Credit Cards, Conversion of Currencies, Transfer Funds, Repository, and

Summary of Accounts. Dependency matrix that we are using to define logical relationship between our core modules is a square matrix which has equal number of rows and columns. The number of rows and columns depends on the number of modules in the project. This structural matrix gives the logical relation of modules among each other. As mentioned earlier, self-dependencies are tagged on its own in grey cells and external dependencies we must provide with a representation according to the requirements and other logical relationships. Upon providing the inputs considering the logical dependencies of modules, it notifies all the cyclic blocks available in the projects. Cyclic blocks are modules with cyclic (2 way) dependencies. Along with cyclic outputs, it gives the detailed information on the number of levels and what modules are placed in what level etc. This is called leveling information. Leveling information is extremely important to decide on the complexity of the project. Projects with few levels are less complex and projects with more levels are more complex. With different project constraints like cost, schedule, efforts into consideration this leveling input places a huge advantage for testers to plan testing effectively during early phases. ALAP and AEAP outputs are enabling the testers to decide on the best and worst test sequences in testing.

V. CONCLUSION:

In this test methodology, we can ensure sequencing of test modules for testing as early in our testing life cycle. This approach is mainly based on module dependencies plotted in dependency structure matrix (DSM). This approach will reduce the detection of faults rate maximum comparing with available existing methodologies. Experience has proved that we can plan effective testing once we know the best – as early as possible test sequence and worst – as late as possible test sequence of testing. As early as possible sequencing is called as the best because it enables to identify defects as early as possible in the testing life cycle minimizing risks in the project. We can see happy customers across the globe as the overall time taken to release the project is comparatively less with existing optimization methodologies. We can possibly extend this process by considering continuous automation of delivery pipeline using Devops using different deployment and infrastructure automation tools like Chef, Puppet etc., to optimize the process even better and deliver the product at the earliest possible time.

REFERENCES

[1]. 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.

[2].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.

[3].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.

[4].Alessandro Marchetto et al.," A Multi-Objective Technique to Prioritize Test Cases ", IEEE Transactions". Vol. 42, No. 10, Oct 2016.

[5]. Sepehr Eghbali and Ladan Tahvildari " Test Case Prioritization Using Lexicographical Ordering/IEEE Transactions On Software Engineering, Vol. 42, No. 12, December 2016.

[6]. 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.

[7]. 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.

[8]. Joseph Krall, Tim Menzies, Member, IEEE, and Misty Davies, Member, IEEE," GALE: Geometric Active Learning for Search-Based Software Engineering", IEEE transactions on software engineering, vol. 41, no. 10, October 2015.

[9]. Stefan Simon and Steven Liu, Member, IEEE," An Automated Design Method for Fault Detection and Isolation of Multi domain Systems Based on Object-Oriented Models", IEEE/ASME transactions on mechatronics, vol. 20, NO. 3, June 2015.

[10]. Annibale Panichella, Rocco Oliveto, Massimiliano Di Penta and Andrea De Lucia, "Improving Multi-Objective Test Case Selection by Injecting Diversity in Genetic Algorithms", IEEE transactions on software engineering, Vol. 41, No. 4, April 2015.

[11]. Robert M. Hierons, "Generating Complete Controllable Test Suites for Distributed Testing", IEEE transactions on software engineering, Vol. 41, No. 3, March 2015.

[12]. Indumathi C , Selvamani K, " Test Cases Prioritization using Open Dependency Structure Algorithm ",1877-0509 © 2015 , doi: 10.1016/j.procs.2015.04.178The Authors. Published by ELSEVIER ICCC-2015.

[13]. Halim. S., Jawawi, D. N., & Sahak. M. (2018). Similarity distance measure and prioritization algorithm for test case prioritization in software product line testing. Journal of Information and Communication Technology, 18(1), 57-75.

[14]. Preference Based Multi-Objective Algorithms Applied tothe Variability Testing of Software Product LinesHelson Luiz Jakubovski Filhoa, Thiago Nascimento Ferreiraa,_, Silvia ReginaVergilioaaDInf - Federal University of Paraná, CP: 19097, CEP: 81.531-980, Curitiba, Brazil

[15]. Automated Software Testing Using MetahuresticTechnique Based on An Ant Colony Optimization-Praveen Ranjan Srivastava, Km Baby- Birla Institute of Technology and Science,(BITS), Pilani Rajasthan, Network System Software Technology GroupCisco Systems, Bangalore,India.

[16]. Utilization and Deployment of Software Testing Tools and Techniques- Naw Tin Tin Yu, University of Computer Studies (Hpa-an), Kayin State, Myanmar-2019