Test Suite Sequencing Using Dependency Structure Matrix

[1]M.Sangeetha, [2]S.Malathi
[1]Research Scholar, Computer Science and Engineering,
Sathyabama Institute of Science and Technology, Chennai. India.
[2]Professor, Computer Science and Engineering,
Panimalar Engineering College,Chennai. India.
sangeetharemi@yahoo.co.in., malathi_raghu@hotmail.com

***Abstract*—Software Testing is a formal process of checking whether a system or a product complies with the consumers need and requirement. It is mainly done by a dedicated testing team using different tools and techniques and the main intention is to identify deviations in the software product and to ensure quality. Testing is generally not done fully, instead it focuses on testing different test stages like Unit, Integration, System, User Acceptance etc., Testing also confirms with a product performance before launching it to the real world. It mainly prevents product failure or breakdown in large scale. The main goal of testing is to access the quality of the end product delivered to the customer. Testing life cycle focuses on different phases – Test plan, Test design, Test execution, Defect reporting and tracking it to closure etc., designing test cases based on requirements are the main building blocks of testing. This work is primarily focused during the test design stage of development. To write effective test cases in shorter time which identifies maximum defects is very crucial in testing life cycle. Effective test case sequencing or prioritizing the test cases based on criticality and risks is a key task of a tester. This methodology is to increase the fault detection at the earlier stage in testing. This prioritization technique organizes the test cases in sequential order in either ascending or descending order. This paper mainly focuses on effecting test sequencing identifying the right modules for testing during the planning phase and prioritize the same using OATS technique and application dependency structure algorithms**

***Index Terms*— prioritization, defects, test cases, cyclic block.**

I. INTRODUCTION

Software Engineering is an engineering discipline that focuses on different stages of software production [1]. Software is not developed as a whole unit; it has different stages namely “Plan”, “Analyze”, “Design”, “Coding”, “Testing” and “Implementation”. 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”[2]. The main intent of software testing is to identify all business-critical defects as much as possible using various testing techniques and ensuring better quality product reaches the customer [3]. Effective software testing will contribute to the delivery of reliable and quality oriented software product. A quality product has more satisfied users, lower maintenance cost, and more accurate and reliable result.

Almost all software engineering projects involves dedicated testing as a separate phase. Because of rapid technology growth and competitiveness in real user business, “time” and “budget” play a major role and these two are the key success factors in task completion [4]. To lead the project in more successful manner, utmost quality must be ensured at the same time minimize the project cost and reduce the delivery time. Consecutively, to increase the effectiveness and efficiency of testing within limited planned resources, effective test case prioritization considering the business demands can be performed. TCP is a process of organizing or selecting the test cases in sequence to increase the fault detection rate at the earliest, which helps to find most critical defects as earlier as possible in the software testing life cycle. Testing costs will drastically go down 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 design. It ensures test optimization ensuring minimal test case designing with maximum coverage. Infeasible unrealistic combinations are also considered in test output. 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 and testing overhead.

Project milestones are improper and not inclined to the proposed test plan. Inadequate test planning due to lack of time of management and lack of focus of testing activity during estimation and testers aware of test estimation model will delay the product to production. Delays occur because of lack of resources to perform the activities in reasonable agreed time frame. Schedule is estimated after the effort is estimated. Developers underestimate the effort and resources required for testing. As a consequence of which, deadlines are not met or software is delivered only partially tested to the ultimate end user.

When budgets are not correctly estimated, it becomes relatively expensive; it might result in some test activities to be canceled causing more insecurity about the quality of the project. [5]Usability testing is a black-box testing technique. The aim is to observe people using the product to discover errors and areas of improvement.

Usability testing generally involves measuring how well test subjects respond in four areas: efficiency, accuracy, recall, and emotional response. The results of the first test can be treated as a baseline or control measurement; all subsequent tests can then be compared to the baseline to indicate improvement. Performance testing details how much time, and how many steps, are required for people to complete their tasks during normal and peak times.

Reliability has to do with the quality of measurement. In its everyday sense, reliability is the "consistency" or "repeatability" of our measures. Before we can define reliability precisely we have to lay the groundwork. First, have to learn about the foundation of reliability, the true score theory of measurement. Along with that, need to understand the different types of measurement error because errors in measures play a key role in degrading reliability. With this foundation, consider the basic theory of reliability, including a precise definition of reliability. There will find out that cannot calculate reliability -- can only estimate it. Because of this, there a variety of different types of reliability that each has multiple ways to estimate reliability for that type. In the end, it's important to integrate the idea of reliability with the other major criteria for the quality of measurement -- validity -- and develop an understanding of the relationships between reliability and validity in measurement

Security testing is a process to determine that an information system protects data and maintains functionality as intended. The six basic security concepts that need to be covered by security testing are: confidentiality, integrity, authentication, availability, authorization and non-repudiation. 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 Taxonomy helps us to understand these different approaches and meanings by providing a base level to work from

Discovery - The purpose of this stage is to identify systems within scope and the services in use. It is not intended to discover vulnerabilities, but version detection may highlight deprecated versions of software / firmware and thus indicate potential vulnerabilities. Vulnerability Scan - Following the discovery stage this looks for known security issues by using automated tools to match conditions with known vulnerabilities. The reported risk level is set automatically by the tool with no manual verification or interpretation by the test vendor. This can be supplemented with credential based scanning that looks to remove some common false positives by using supplied credentials to authenticate with a service (such as local windows accounts).

Vulnerability Assessment - This uses discovery and vulnerability scanning to identify security vulnerabilities and places the findings into the context of the environment under test. An example would be removing common false positives from the report and deciding risk levels that should be applied to each report finding to improve business understanding and context.

Security Assessment - Builds upon Vulnerability Assessment by adding manual verification to confirm exposure, but does not include the exploitation of vulnerabilities to gain further access. Verification could be in the form of authorized access to a system to confirm system settings and involve examining logs, system responses, error messages, codes, etc. A Security Assessment is looking to gain a broad coverage of the systems under test but not the depth of exposure that a specific vulnerability could lead to

Penetration Test – Penetration test simulates an attack by a malicious party. Building on the previous stages and involves exploitation of found vulnerabilities to gain further access. Using this approach will result in an understanding of the ability of an attacker to gain access to confidential information, affect data integrity or availability of a service and the respective impact [6]. Each test is approached using a consistent and complete methodology in a way that allows the tester to use their problem-solving abilities, the output from a range of tools and their own knowledge of networking and systems to find vulnerabilities that would/ could not be identified by automated tools. This approach looks at the depth of attack as compared to the Security Assessment approach that looks at the broader coverage.

Security Audit - Driven by an Audit / Risk function to look at a specific control or compliance issue. Characterized by a narrow scope, this type of engagement could make use of any of the earlier approaches discussed (vulnerability assessment, security assessment, penetration test). Security Review - Verification that industry or internal security standards have been applied to system components or product. This is typically completed through gap analysis and utilizes build / code reviews or by reviewing design documents and architecture diagrams [7]. This activity does not utilize any of the earlier approaches (Vulnerability Assessment, Security Assessment, Penetration Test, and Security Audit)

The motivation of this work is to automate the combinatorial test inputs and give the tester the optimized combinations for test case generation. Ensures effective test planning keeping all project constraints like schedules, budget etc. using CODEC and OA Strategies. The work details the new features required on suite optimization. This includes making 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 prioritized combinations, improving the reporting functionality of the application.

II. Related Work

In existing system, it is impractical for a testing team to decide on the exact sequence of application modules to get tested [1]. There is no effective tracking mechanism for tracking the test progress when the number of iteration of testing is high. Poor intra-group coordination with the development / other support team which delays the deployment of the product to production. There is no availability of separate test environment for Regression Testing to simulate the actual expectations [5]. Lack of focus on Configuration Management Process which causes delays due to changes. Test design challenges include test case forecasting, over test cases, relevant test coverage and optimum proving.

Many works pertaining to test case prioritization has been proposed and implemented by many researchers in the past. Some of the few important works has been cited in this section. Existing approaches that uses [12] derived dependency structures among various test cases in a test suite is performed manually which consumes more time to test the job by a tester as well as tests the system time. Hai dry's technique is used [1] for finding total number of dependents for each test case using DSP (Dependency Structure Prioritization) volume is not working properly. Hence, propose a test sequencing methodology which identifies the best sequence and worst sequence of module identification for testing using dependency structure matrix during the planning phase of testing life cycle. This overcomes all the existing research techniques as we propose our work optimizing test plan itself identifying the best and worst sequence of modules for testing.

Previously Alessandro Marchetto et al.,[4] proposed technique aims at both early discovering faults and reducing the execution cost of test cases by applying a metric-based approach to automatically identify critical and fault-prone portions of software artifacts, thus becoming able to give them more importance during test case prioritization but in the design phase of testing not in planning phase. Also Dan Hao, Lu Zhang et al.,[6] implemented the ideal optimal test-case prioritization technique, which schedules the execution order of test cases based on their detected faults using optimal coverage based test-case prioritization as an integer linear programming (ILP) problem but not in more earlier phase in order to produce the result in optimized manner.

III. Proposed Work

Sequencing the test modules considering the functional dependencies is crucial to plan testing. Figure 1 shows the system architecture of testing which has two phases planning and design

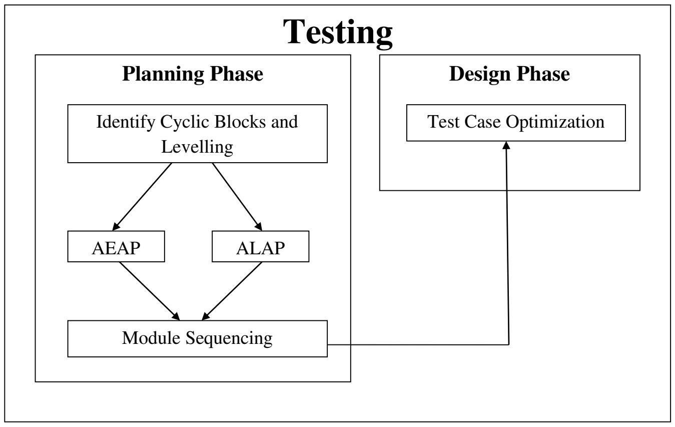


Fig. 1. Architecture of Test Sequencing

Test sequencing is a way or technique used to sequence the modules of the application for testing as early as possible in the testing life cycle. Test sequencing yields great outcomes minimizing time and budget when effectively planned during the planning stage of testing life cycle comparatively over test design stage as other researches proved. Planning effective test sequencing during test plan stage reduces the overall cost of quality (COQ) minimizing the rework cost radically when compared to other test prioritization techniques that targets the test design stage. CODEC ensures Cyclic Block identification, Leveling and Tagging outputs, Critical path identification, AEAP and ALAP Sequencing. Cyclic Block identification feature in Codec ensures that the tool identifies all modules that has cyclic dependencies among them and lists for the tester to tackle the same separately during test planning. The testing team to understand in how many levels the application modules can fit in. Also, it indicates the complexity of overall project.

This methodology identifies the relational statistical input provided is converted into As Early As Possible (AEAP) sequencing and As Late As Possible (ALAP) sequencing. AEAP sequencing tells the estimator the appropriate test module sequencing with minimal risk. ALAP sequencing tells the estimator the appropriate late sequencing with maximum risk without schedule deviation.

AEAP – As early as possible testing and ALAP – As late as possible sequencing are two extreme suggestions of test sequencing. In AEAP risk is minimum and in ALAP risk is maximum. Orthogonal Array test design technique calculates the optimized combinations of combinatorial input, enables a test designer to design test cases with minimal effort and maximum coverage. Infeasible combinations feature will not generate test combinations which logically may not happen. Before generating the output, we can instruct the tool to skip the possible combinations. Adding priority to specific combinations, will generate more combinatorial output.

DSM – Dependency Structure Matrix analysis is mainly used for test sequencing. It ensures the order of execution of modules and ensures which modules to be tested in parallel. This determines the sequence of execution of all the modules in the application and determines the modules to be kept under a single team. It also tells the modules to be tested in parallel without any dependency clash. SCE – System Complexity Estimator feature, is mainly used for test effort estimation. The main advantage here is the efforts are distributed across all modules while testing. This analyze the dependencies and gives the effort distribution across all modules. It also determines the total effort required for testing. System complexity estimator work is used for test effort estimation. SCE addresses the distribution of efforts across all modules while testing. It analysis the dependencies and gives the effort distribution across all modules. It determines the total effort required for testing, if past trend of similar system is available

SCIM - System Change Impact Matrix Analysis is used during Maintenance Testing. It addresses to distribute testing effort across modules after receiving change requests from the customer. It also determines the impact on the system by each change request. It gives the effort distribution across all the modules in the maintenance phase and it also prioritize the order of the change requests. is used during maintenance testing to identify the additional test effort required because of change requests raised by the customer that may affect the original estimation plan. It distributes testing effort across modules after receiving CRs. It determines the impact on the system by each CR.

Test suite optimizer is a part of robust testing methodology, which has been formulated to overcome the challenges of reduce cycle time of test phases, to find maximum defects with minimum test cases and offer maximum coverage with minimum test cases. Systematic and statistical method of pair-wise combinations of factors across their levels. It creates an optimized test suite with lesser test cases, detects all single mode and double mode defects and increases confidence in the system by executing a concise set of tests and uncovering most of the bugs as early as possible. This ensures improved productivity with cycle time reduction, improved test coverage, minimizes the size of test suite by eliminating redundant test cases. These test cases can be customized based on the available time and known problems. Test effort reduction in terms of test case writing and execution, Infeasible combinations can be removed and Priority can be given for factors and levels.

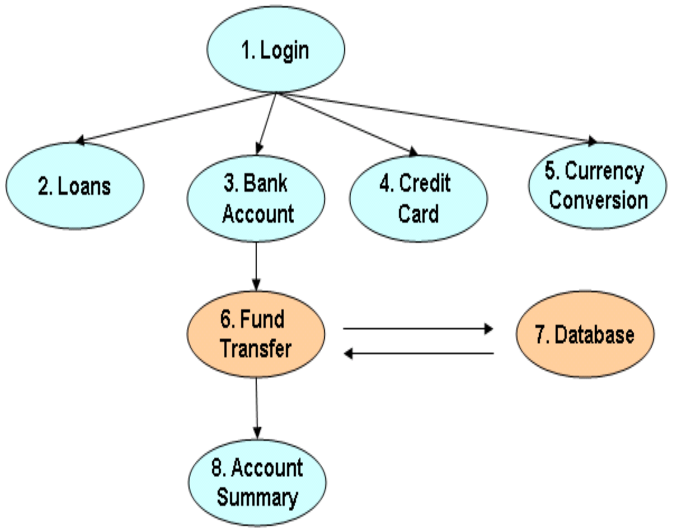


Fig. 2. Sample flow graph for Banking Transaction

Fig.2 gives the flow graph, which illustrates a bank example. There modules are Login, Loan, Bank Account, Credit Card, Currency Conversion, Fund Transfer, Database and Account Summary. There is a two-way relationship between Fund transfer and Database. It means both are tightly coupled, both the modules are inter-related to each other. If we change fund transfer it affects the data in the database module. If we change database module, it affects the fund transfers in fund transfer module. This is otherwise called as “cyclic dependency”. Other modules include Loans, Bank Account, Credit Card and Currency Conversion are dependent on Login module. Login acts as a root or entry point to access all other modules which carries a greater dependency among all modules.

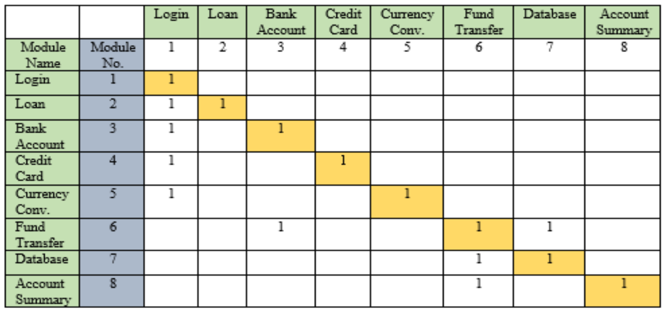
Algorithm for the Update Routing Table

```
Input: Modules
Output: Return Path
Step 1: Add Route in Routing table.
Step 2: Get first entry of Routing table.
Step 3:If field list_node_link contains target nod
          Update List_node_link field
          Update Additive_cost field
          Update Restrict cost field
          Else Get more entry
Step 4: End.
```

IV. Results And Discussion

The dependency structure matrix of the above example can be represented in Table 1 shown below. The gray cells indicate the self-dependency that every module has on itself. Self-dependencies are identified automatically in proposed methodology. Other external dependencies will provide in the matrix as a statistical input. If indicated "1" on the interconnected cell between login and loan modules that designates a coherent dependency between login and loan modules.

TABLE I
DEPENDENCY MATRIX STRUCTURE



		Login	Loan	Bank Account	Credit Card	Currency Conv.	Fund Transfer	Database	Account Summary
Module Name	Module No.	1	2	3	4	5	6	7	8
Login	1	1							
Loan	2	1	1						
Bank Account	3	1		1					
Credit Card	4	1			1				
Currency Conv.	5	1				1			
Fund Transfer	6			1			1	1	
Database	7						1	1	
Account Summary	8						1		1

Considering the example of a simple banking functionality, it has 7 modules – Login, Loan, Bank Account, and Credit Card, Currency conversion, Fund Transfer, Database and Account summary. Dependency structure matrix is a square matrix which has n rows and n columns based on the number of modules taken into consideration. This matrix defines the relationship of one module over other modules with a representation. The grayed diagonals with 1 represent self-dependency of a module on its own. The other dependencies will provide representing 1 as per the requirements.

On providing the statistical inputs considering the relations and dependencies between the application modules using Dependency matrix analysis the cyclic blocks – the modules that holds a cyclic dependency among each other, the levels where the modules are placed, tagging table that provides a clear picture on to which level the corresponding module is tagged. These results are very important and crucial to study the complexities of the overall project which helps planning the test activities considering the budget and schedule constraints. As-early-sequencing and As-late-sequencing outputs helps to derive the right sequencing technique to plan our testing activities smoothly at the earliest in our life cycle of testing.

Conclusion

This technique is used to ensure test sequencing as early as possible in the test plan stage itself based on dependency structure algorithm which will improve the fault detection rate drastically considering key project constraints of budget, time in mind comparing with the existing techniques. Testing phase to effective plan testing based on two extreme variants of test sequencing AEAP and ALAP based on dependency structures. This also helps to prioritize the test cases during test design phase within a very short period time, as well as to minimize the speed of test process and increase the rate of fault deduction at earlier stage and hereby reduce the time taken to deliver the end product to the client.

This methodology can be further extended implementing DevOps way automating the entire delivery pipeline of application development considering various deployment and infrastructure automation tools which will even speedup the process of delivering the right product.

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] D. Bianculli, A. Filieri, C. Ghezzi, and D. Mandrioli, "Incremental syntactic-semantic reliability analysis of evolving structured workflows," in Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change. New York,NY, USA: Springer, 2014, pp. 41–55.

[14] S. Yin, H. Luo, and S. Ding, "Real-time implementation of fault-tolerantcontrol systems with performance optimization," IEEE Trans. Ind. Electron.,vol. 61, no. 5, pp. 2402–2411, May 2014.

[15] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints," IEEE Trans. Evol. Comput., vol. 18, no. 4, pp. 577–601, Aug. 2014.

[16] Kamel Rekab, Member, IEEE, Herbert Thompson, and Wei Wu," A Multistage Sequential Test Allocation for Software Reliability Estimation", IEEE Transactions on Reliability, Vol. 62, No. 2, June 2013.

[17] Cemal Yilmaz, "Test Case-Aware Combinatorial Interaction Testing", IEEE transactions on software engineering, Vol. 39, No. 5, May 2013.

[18] Gordon Fraser, Member, IEEE, and Andrea Arcuri," Whole Test Suite Generation",IEEE Transactions on Software Engineering, Vol. 39, No. 2, February 2013.

[19] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B.Turhan, and T. Zimmermann, "Local versus global lessons for defect prediction and effort estimation," IEEE Trans. Softw. Eng.,vol. 39, no. 6, pp. 822–834, Jun. 2013.

[20] Gordon Fraser, Member, IEEE, and Andreas Zeller, Member, IEEE Computer Society," Mutation-Driven Generation of Unit Tests and Oracles", Oracles/IEEE Transactions on Software Engineering, Vol. 38, No. 2, March/April 2012.

[21] Siavash Mirarab, Soroush Akhlaghi, and Ladan Tahvildari, Senior Member, IEEE," Size-Constrained Regression Test Case Selection Using Multicriteria Optimization",IEEE Transactions on Software Engineering, Vol. 38, No. 4, July/August 2012.

[22] I. Hwang, S. Kim, Y. Kim, and C. E. Seah, "A survey of fault detection,isolation, and reconfiguration methods," IEEE Trans. Control Syst.Technol., vol. 18, no. 3, pp. 636–653, May 2010.