Prioritize the Sequence of Test Cases and Increase the DRE Rate by Implementing Orthogonal Array Testing Strategy

Sangeetha M. (sangeethamphd2021@gmail.com)
Sathyabama Institute of Science and Technology
Malathi S
Panimalar Engineering College

Research Article

Keywords: prioritization, dependency, quality of service, Emerging technology, effective design

Posted Date: September 27th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-921365/v1

Abstract

Software testing is an emerging technology which is use to increase the rate of error detection as early as possible in the software testing life cycle process. Test case prioritization technique plays a crucial role in organizing the test cases in sequencing order both ascending and descending such that test cases having high priority or high severity are planned to get executed doing a proper risk-based analysis. This prioritization technique effectively addresses two important organization constraints namely “Time” and “Budget”, also improve the quality of service. The proposed work is all about how effectively we can sequence the application modules for testing during Test plan phase using fuzzy logic and how to write optimized test cases efficiently design during Orthogonal Array Test Strategy (OATS) in Test design phase of testing life Cycle.

1. Introduction

Software Testing isn't done all in all, it focuses on various test stages – Test Plan, Test Analyze, Test Design, Test Execution, Defect Reporting/Tracking and so on., Writing test cases based on specifications are the building blocks of testing that mainly occurs in test design stage. Test case experimental sequencing or organizing the test cases is a procedure of requesting the experiments as per business requests to accomplish a specific objective**.** On the off chance a similar imperfection is recognized in plan or coding phase of the product improvement life cycle, it will expand the Cost of Quality (COQ). COQ includes three different costs: A. Prevention costs B. Appraisal costs C. Rework costs. Prevention costs are the costs that are spent in organizations to prevent the defects. Example: Training costs. Appraisal costs are the cost incurred for appraising defects. Example: A tester identifying critical defects will get paid for his/her efforts. Rework costs are the costs spent in backtracking and reworking on any already completed work. Example: Requirement defects identified during testing phase must get reworked again from the scratch. Any project that targets on ensuring quality mainly focuses on minimizing the cost of quality (COQ).

Testers usually have a different mind-set from that of the developers who coded the application. Since people can hardly find their own mistakes, an independent team is recommended who can able to test the product quality in a cost-effective way. The higher the independence, the more it can contribute to finding defects, focus on the testing efforts, and provide an independent view on the product.

Software testing helps in the real world to reduce cost and time, to maintain reputation, to meet contractual or legal requirements. Phases of software testing can guarantee that the application meets all the execution related measurements, for example, reaction time, accessibility, and limit necessities. Any algorithm that targets on test case prioritization needs to take care of these two parameters for successful completion of the project ensuring “quality” minimizing quality costs.

There is a technique widely used in the industry called Boundary value analysis (BVA) to prioritize the test cases for testing. Here extreme boundary values are considered while designing the tests. A boundary is

the place where the system's behavior is expected to change. A boundary can also be defined as a limit where a valid value turns into an invalid value. In BVA, boundaries between equivalence partitions are tested. BVA is also known as 'Range Checking'. The boundary values for the most part incorporate greatest, least, only inside/outside, common esteems, and blunder esteems. The Test Lead is the person who designs it when the testing courses of events are characterized. Equivalence Partitioning and BVA are related to each other and can be used together at all levels of testing. Using the BVA technique will allow you to be more successful in your testing over equivalence partitioning, only in that defects are more likely to occur at boundary points between partitions. This technique believes that more errors are prone to occur as clusters in bounding edges. 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.

Decision table is another popular technique used for test case optimization which targets at system level considering causes (inputs) and effects (outputs). This is also otherwise called as "Cause-effect Graphing". This technique is suitable for problems that has huge number of inputs (causes), on combining provides huge number of different outputs (effects). This technique is used to solve real world business problems that have varying combinations of test inputs which provides different set of test outcomes. The fundamental preferred standpoint of this system is that it guarantees all basic test mixes are secured staying away from redundancies and miss-outs. Decision tables are also a very good method to deal with a combination of inputs producing different results. The Decision Table method is very significant because, with this method we can test multiple combinations. The number of possible combinations is given by 2 ^ n ('n' being the number of inputs). We can choose a rich subset of all possible combinations using this method. The test lead is the one who plans it when the testing timelines are defined. Decision Table testing is a technique that assists in breaking down the complex logic by recording business rules based on a set of test conditions. The Decision Table technique helps to test business rules against various system conditions (user inputs) and corresponding actions (system outputs). This technique analyses two parameters: Conditions: Represent process or application inputs. Actions: Represent what should be done or what should happen for each set of conditions. The steps testers use to build a decision table includes, identify conditions and their values, listing all possible actions, identifying all condition combinations, identifying all actions for each condition combination to create business rules and link the rule to the corresponding actions. Tabular format is the best way to represent complex business rules and analyze them for test design.

The next technique mainly used in organizations for effective test designing is State transition testing. 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. The limitation with the State Transition diagram is that it does not allow users to completely describe a complex or a big process with many branches due to lack of space. It is a fact that the usability of over complicated State Transition diagrams degrades significantly. State Transition tables must be utilized to list all blends of the states, conditions, and activities, that assistance to thoroughly dissect the framework advances.

Utilizing use case-based testing is a discovery test that outlines system in which TCERs are intended to execute client situations that are characterized by the base and elective streams of a utilization case. Utilize Cases are client situated portrayals of the application's usefulness. A Use Case portrays an arrangement of activities that an application performs for a specific Actor. Utilize Cases are intense technique for portraying necessities. They frame reason for Software Design, Test cases, User documentation and GUI outline. By having an arrow connecting the Actor and the Use Case, the Tester knows the actions or features the Actor can perform on the system under test.

Orthogonal Array Test Strategy (OATS) is a discovery test advancement system which utilizes a factual method for recognizing the improved test suite. This technique is well suited when the number of test inputs is relatively huge and where exhaustive testing goes ineffective, testing all possible combinations. Say for example, if the requirement is to test a train ticket booked through IRCTC mobile app, the factors (input combinations) needs to be tested are “the number of passengers travelling”, “ticket number”, “PNR number”, “seats allocated”, “train number”, “train name” etc., Considering each factor identified can have different values. Say for example: the factor “seats allocated” can have multiple values if the numbers of passengers travelling are more. This type of problems will have huge challenge identifying the number of factors and for each factor identifying the number of levels. Such problems that can have factors and levels are what we call as a “combinatorial” problem. OATS technique is mainly suited for combinatorial problems. Identifying all possible factors and levels and writing effective test cases for every possible combination is extremely challenging and impractical to achieve.

Orthogonal Array Test Strategy is a test optimization technique that concentrates on identifying and prioritizing the test combinations which ensures maximum test coverage identifying major defects with minimum number of test combinations. OATS is targeted on identified different modes of defects that occur in real life projects. A defect that is occurred because of a single statistical input is called as a “Single mode” defect. A defect that is occurred combining two statistical inputs is called as “Double mode” defects. Like the same way, there can be triple mode, multi-mode and so on. Universal researches have proved that maximum of real-life projects fail because of double mode combinations of defects.

2. Existing Work

Numerous works relating to experiment prioritization has been proposed and executed by numerous analysts before. A portion of the couple of essential work has been referred in this segment. Existing approaches are used [1], [2] derived dependency structures among various test cases in a test suite is performed manually that are consumes more time. Hai dry's procedure [3] is utilized for discovering

absolute number of wards for each experiment utilizing Dependency Structure Prioritization volume isn't work legitimately. Existing approaches [6] that uses surmised dependence structures among different examinations [12] of test cases are performed manually that exhausts greater chance to test the action by an analyzer and furthermore tests the system time.

Dan Hao, Lu Zhang et al., [16] actualized the perfect ideal experiment prioritization method, which plans the execution request of experiments in view of their identified shortcomings utilizing ideal scope-based experiment prioritization as a number straight programming (ILP) issue yet not in more prior stage keeping in mind the end goal to deliver the outcome in streamlined way. Already Alessandro Marchetto et al., [21] proposed method that goes for both early finding deficiencies and diminishing the execution cost of experiments by applying a metric-based way to deal with consequently distinguish basic and blame inclined bits of programming relics, in this way getting to be ready to give them more significance amid experiment prioritization however in the outline period of testing not in arranging stage. Hence, my proposed work an effective test sequencing methodology which identifies the best sequence and worst sequence of module identifications for testing using dependency structure matrix during the planning phase of testing life cycle. This overcomes all the existing research techniques[23,25] as my propose our work optimizing test quality during test plan itself, identifying the best and worst sequence of modules for testing followed by optimizing the test suites identified during test design phase customizing OATS techniques optimizing single, double and triple mode test combinations.

3. Proposed Work

As like software development life cycle, we do have software testing life cycle which core software testers in software project will work on. It includes the following stages: Requirements, Test Plan, Test Design, Test Execution and Defect reporting & Tracking. In Requirements stage, testers will analyze the requirements ensure whether it is flexible enough to achieve in terms of budget, technology, practicality of requirements. A thorough feasibility study is required to take the testing process to the next stage. Generally during this stage, testers lack clarity on the requirements, dependency on all requirements, mainly on how to follow testing in a sequence – which modules of application to test first, test next and test last for early detection of defects. This has become a huge challenge for testers working in Test Plan phase to decide on the flow of testing, plan budget and schedule timelines in a testing project.

The proposed idea of test sequencing using dependency structure matrix provides a methodology for testing using fuzzy logic to decide on the best sequence of testing called as “As early as possible sequencing” and the worst sequence of testing called as “As late as possible sequencing” which is referred in short as AEAP and ALAP respectively. This test sequencing of modules identification using fuzzy logic identifying the intermediate modules which are partially true or partially false is very much appropriate during Test plan phase rather than Test design phase as other proven techniques exist shows in fig.1. Implementing effective test sequencing using Fuzzy logic mechanism provides a valuable flexibility for reasoning. The main advantage of using fuzzy logic is that it resembles human reasoning in many ways picking the right modules for test sequencing deriving 2 main fuzzy sets AEAP and ALAP. In

our fuzzy sets – AEAP and ALAP, every module identified as sequence are elements in the fuzzy set. So, consider our fuzzy set as “As early as Possible” sequence set (AEAP), the members of the set will be the modules – Login, Bank Account, Credit cards etc., considered in sequence. Modules considered for testing in the first release R1, will form the fuzzy set of AEAP and ALAP and each module considered for release 1 will be assigned a membership value. Fuzzy logic systems AEAP and ALAP are generally governed by membership functions. Using fuzzy sets has an edge over classical sets, as it allows partial membership and it contains elements of varying degrees of membership. This enables to sequence the modules for testing more effectively and efficiently so that considering the sub-modules, external modules holding partial membership to a greater extent in AEAP and ALAP sequence. In test sequencing the modules for testing gives a better view to the testers to plan effectively and efficiently the flow of subsequent testing phases which in turn helps the project minimizing defects, rework and able to deliver the project well ahead of scheduled time.

After deciding the best sequence or worst sequence considering the risks, availability and other factors, moving to test design phase and writing quality test cases for the identified test sequence is a time killing process among the entire testing life cycle. Our goal is to identify a suitable test optimization technique for creating effective test cases which identifies maximum defects with minimal efforts. The entire architecture as shown in fig.1

4. Test Module Sequencing In Test Plan Phase

My proposed work consists of a real time banking project which has the following core modules like Login, Loans, Bank Account, Credit cards, Currency conversion, Fund transfer, Database and Account summary. The flow graph of the same is shown in fig.2.

Totally 8 core modules to get tested both individually which includes of Unit Testing and as a whole testing the interactions and the dependencies of one module over the other modules. Generally, in a typical testing project once the testers receive the modules for testing, there is no defined way so far as on how to start testing or which module to start testing first and move on till all functionalities are covered. It is not always necessary that the first module should get tested first, in our case Login must get tested first. What if, the first module is still under development or under design where testing team doesn’t have a better clarity to start testing it? There are situations where we will get only the design document that defines the modules, flow of execution, but the actual modules are still in development phase with the developers. Test sequencing is an approach that tells the testing team in which order we need to test the modules to save time, budget, man-power etc., Almost in all projects till today, test sequencing is decided individually by the project folks considering the availability of the modules, resources, ease of use etc., There is no standard way of deciding the best approach of Test sequencing and worst approach of Test sequencing of modules.

My research proposes a standard way of deciding the best possible sequence for Testing and the worst possible sequence for testing the modules using Fuzzy sets. For deciding this, the dependency matrix is

important which has the information about relationship of one module over the other.

Dependency relationship in Banking Application:

Login – It is independent of all modules. It is the root/top module

Loans – It is dependent on Login module

Bank Account – It is dependent on Login module

Credit Card – It is dependent on Login module

Currency Conversion – It is dependent on Login module

Fund Transfer – It is dependent on Bank Account module and Database module

Database – It is dependent on Fund Transfer module

Account Summary – It is dependent on Fund Transfer module

All modules are self-dependent on its own by default and it may or may not dependent on other modules. The dependencies are defined clearly in the form of dependency matrix. In our banking project, Fund Transfer is dependent on Database and Database is dependent on Fund Transfer. This is an example of "**Cyclic Dependency**".

The proposed concept takes the dependency information of modules in a excel table called as "Dependency matrix" as mentioned below in the figure below. Dependency matrix acts as an input here. Dependency matrix for the mentioned banking project is as below table.1.

Table.1.Matrix representation of dependency 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

The grey celled "1"s is self-dependency and the other "1"s are the dependencies of modules based on the requirements shared by the customer based on the flow graph. Once input these details, our proposed concept on test sequencing using fuzzy logic identifies the intermediate modules which are partially true and partially false forming the fuzzy sets. All the information in the identified fuzzy sets are discrete and

they represent the degree of truth in the fuzzy logic system. Here we deal with partial true and partial false levels; we identify membership levels for each module assigned to level. These membership values depict the degree of truth in the identified fuzzy systems. The 2 major fuzzy systems – "As early as possible sequencing (AEAP)" and "As late as possible sequencing (ALAP)" forms the base of our test sequencing way of testing the modules considering the risks, assumptions and other factors that impact the testing process in subsequent stages of testing life cycle.

The "As early as possible sequencing" or "best sequencing" of bank modules represented in fuzzy systems is as below in Fig:3

The sequence structure of modules placed for possible sequencing in levels identified (1-4) as shown in fig.4.

Fig 5 shows the "As late as possible sequencing" or "worst sequencing" of bank modules represented in fuzzy systems is as below in Fig:

In fig 6 the sequence structure of modules placed for possible sequencing in levels identified (1-4) as shown.

5. Test Case Optimization In Test Design Phase

Test case optimization is a technique using which we can minimize the number of test cases planned for testing ensuring maximum coverage with minimal testing effort. It will not ensure 100% test coverage. It satisfies one of the 7 testing principle – "Exhaustive Testing is not possible". 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., There are different Test case optimization techniques available that many projects are using to reduce the number of test cases. Few popular techniques include – Equivalence Partitioning, Boundary Value Analysis, Decision Tables, Orthogonal Array strategy (OATS) etc.,

My proposed work focuses on fine tuning Orthogonal Array test strategy methods with additional proposed features like adding precedence factors & identifying infeasible combinations which will help a tester for better test case optimization.

Orthogonal Array Test Strategy has **g**iven a combinatorial problem which consists of "n" factors and "m" levels, OATS technique will ensure all double mode combinations of inputs are tested 100%. Any tools/techniques which follow OATS principle should follow "Double-mode combinations are tested 100% in the optimized suite of test cases.

My proposed work analyzed Combinatorial Problem in Real life for a 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 "Product category" indicates the type of purchased item. Meaning assumptions include 2 levels – Loose (item given in loose based on buyer's need) and Packed (packaged

item/sealed item). The next factor is “Discount”. Is price cut or discount applicable for the item purchased? It accepts 2 levels as answers – Yes and No. The third factor is “Tax”. 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 “Mode of Payment”. 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, Google Pay etc.

6. Results And Outputs

Factors & Levels Abstraction:

The following table.2 represents the factors name and level for the retail marketing project.

Table.2.Factors and Levels

S.No	Factor Name	No. of Levels	Level Name			
1	Product Category	2	Loose		Packed	
2	Discount	2	Yes		No	
3	Tax	3	VAT	Duty	Sales	
4	Mode of Payment	4	Cash	Credit	Debit	Google Pay

Precedence Factors

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.

Optimized Outputs with Custom Precedence Factors

The sample output below shows the optimized output generated for “Custom” article category with greater precedence. We can able to see more test case combinations have “Custom” category in it.

Case1: As per the customer requirement, for Factor: Article Category, Level: Custom should appear a greater number of times in the optimized output of test cases.

The outputs are shown in table.3.

Table.3.Product Category; level: loose

SELECTED PRECEDENCE FACTOR: Product Category; LEVEL: Loose				
S.No	Product Category	Discount	Tax	Mode of Payment
1	Loose	Yes	VAT	Cash
2	Packed	No	Duty	Cash
3	Loose	Yes	Sales	Cash
4	Loose	No	Sales	Credit
5	Packed	Yes	VAT	Credit
6	Loose	Yes	Duty	Credit
7	Loose	Yes	Duty	Debit
8	Loose	No	VAT	Debit
9	Packed	Yes	Sales	Debit
10	Loose	Yes	VAT	Google Pay
11	Packed	No	Duty	Google Pay
12	Loose	Yes	Sales	Google Pay

Case 2: As per the customer requirement, for “Factor: Article Category”, “Level: Box up” should appear a greater number of times in the optimized output of test cases.

The output will look like table.4:

Table.4.Product category; Level: Packed.

SELECTED PRECEDENCE FACTOR: Product Category; LEVEL: Packed				
S.No	Product Category	Discount	Tax	Mode of Payment
1	Loose	Yes	VAT	Cash
2	Packed	No	Duty	Cash
3	Packed	Yes	Sales	Cash
4	Loose	No	Sales	Credit
5	Packed	Yes	VAT	Credit
6	Packed	Yes	Duty	Credit
7	Loose	Yes	Duty	Debit
8	Packed	No	VAT	Debit
9	Packed	Yes	Sales	Debit
10	Loose	Yes	VAT	Google Pay
11	Packed	No	Duty	Google Pay
12	Packed	Yes	Sales	Google Pay

Adding precedence feature in optimized test suite will enable ease of customization by saving enormous time and effort for the test designers in test design phase.

Infeasible Combinations:

Infeasible combinations are the combinations that will NEVER occur in real-time. Consider the below simple project – Checking an employee passport, visa status and the employee is in Offsite or Onsite. Table.5 represents the combinatorial table representing the possible combinations.

Table.5. Combinatorial possible combination.

S.No	Factor Name	No. of Levels	Level Name	
1	Passport	2	Yes	No
2	Visa	2	Yes	No
3	Location	2	Offsite	Onsite

When we try to generate a pair-wise combinatorial output using any OATS, we will get a combinatorial optimized test suite shown in table.6.

Table.6.Combinotorial optimized test suite.

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

The tool 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. In the above analysis, the combination with Passport – No and Visa – Yes is irrelevant. It's illogical to create a test case like Passport – No, Visa – Yes, Location – Onsite. There is no question of Visa, if an employee doesn't hold a passport. Such a combination is called as "Infeasible Combination"

We can well in advance avoid generating such an infeasible combination by providing the combination as a infeasible input before generating the test optimization suite.

Infeasible Combination Input: "Passport – No, Visa – Yes, Generate no test case"

Considering the infeasible input, the optimized output is as below table.7.

Table.7.Infeasible input and optimized output.

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

7. Comparison Analysis And Discussions

The following are implications that we want to show from our usual way of testing applications against implementing my proposed work.

Fig.7 represents the total no of test cases written for the given requirements. Only it needs half have from existing system. Also it shows the increased rate of DRE against existing system.

We see a decent raise in the Defect Removal Efficiency(DRE) rate which we usually use to analyze the test effectiveness .

DRE= (No. of Defects found during QA testing / (No. of Defects found during QA testing +No. of Defects

Found by End-user)) * 100

8. Conclusion And Future Work

My proposed work depicts the Test Module Sequencing in Test Plan Phase for effective test sequencing models (AEAP and ALAP) using Fuzzy logic systems – I propose to fine tune the usual test plan phase of testing where we usually arrive at preparing the test strategy, test approach etc., which acts as the building blocks of further testing in the testing life cycle. Understanding the best possible test sequence and worst possible test sequence will help any tester to effectively prepare the deliverables in Test plan phase also in Test Case Optimization in Test Design Phase: Fine tuning the usual test optimization techniques by introducing precedence factors and indicating infeasible combinations well in advance will help any tester who designs test cases to create much efficient and optimized test design suite in Test design phase.

In future my work can be extended with any other real time projects analysis and increase the DRE rate further.

Declarations

Ethical Compliance

Not applicable

Conflicts of Interest

There is no conflict of interest

Funding

There is no funding

Acknowledgments

All authors have seen the manuscript and approved to submit it to the journal.

References

[1]. 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.

[2]. 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.

[3]. 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.

[4]. 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.

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

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

[7].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.

[8].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.

[9].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.

[10].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. [11].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.

[12].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.

[13]. 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.

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

[15].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.

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

[17].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.

[18]. 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. [19].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.

[20].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.

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

[22].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.

[23].M.Sangeetha,S.Malathi,Test Suite Sequencing Using Dependency Structure Matrix”,”Advanced Research and Engineering”March-18.

[24].AleksandrPilgun et al.,”Fine-grained code coverage Measurement in Automated Black-box Android Testing” ACM transaction on software Engineering AND Methodology-nO:23, https://doi.org/10.1145/3395042-July 2020.

[25]. Andrea Arcuri et al.,”Handling SQL database in automated system test generation”,ACM Transaction on Software Engineering and Methodology-July 2020,No:22,https://doi.org/10.1145/3391533

Figures

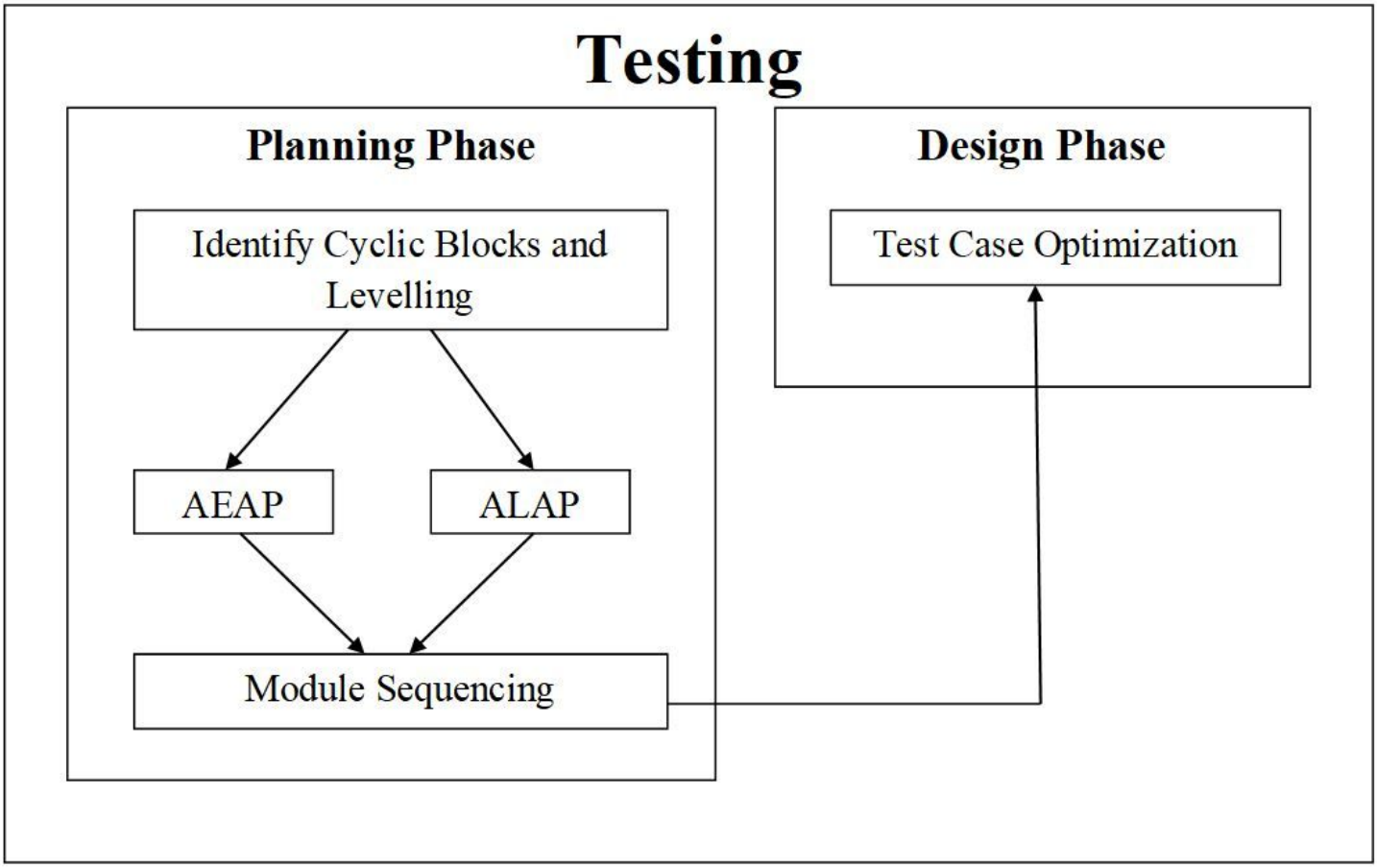

Figure 1

Architecture diagram of Test Sequencing and Test case optimization

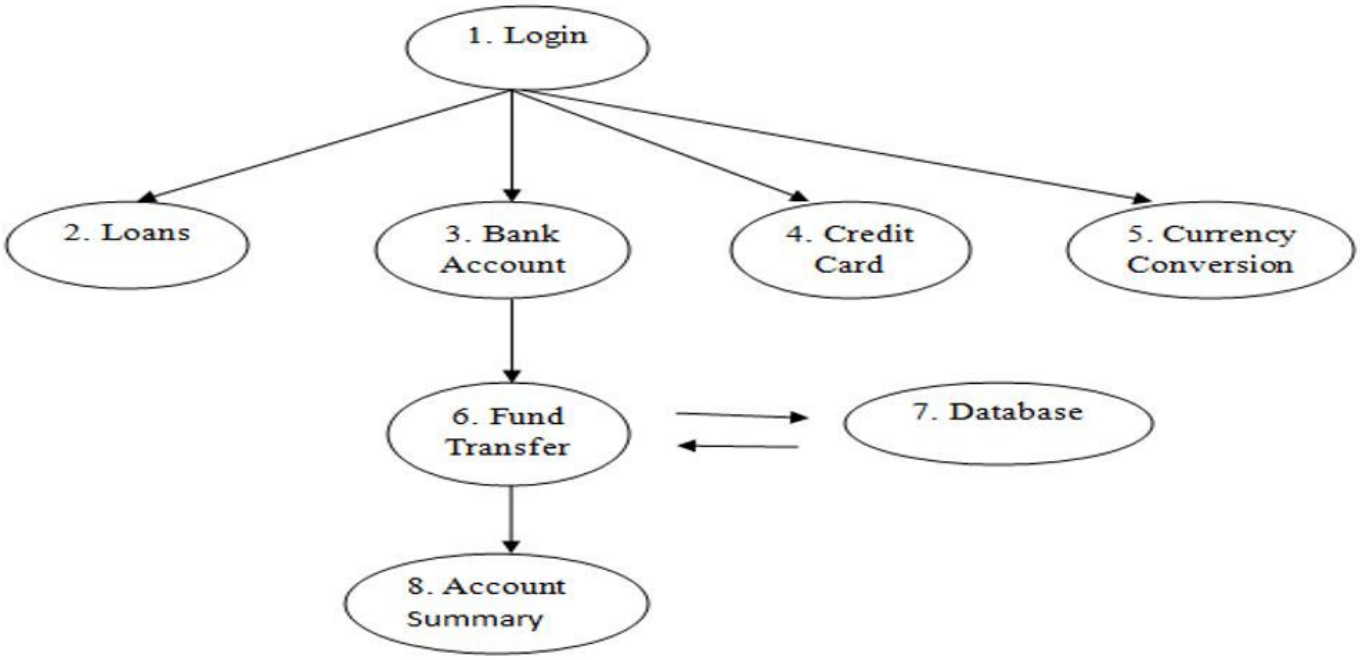

Figure 2

Module representation of banking project.

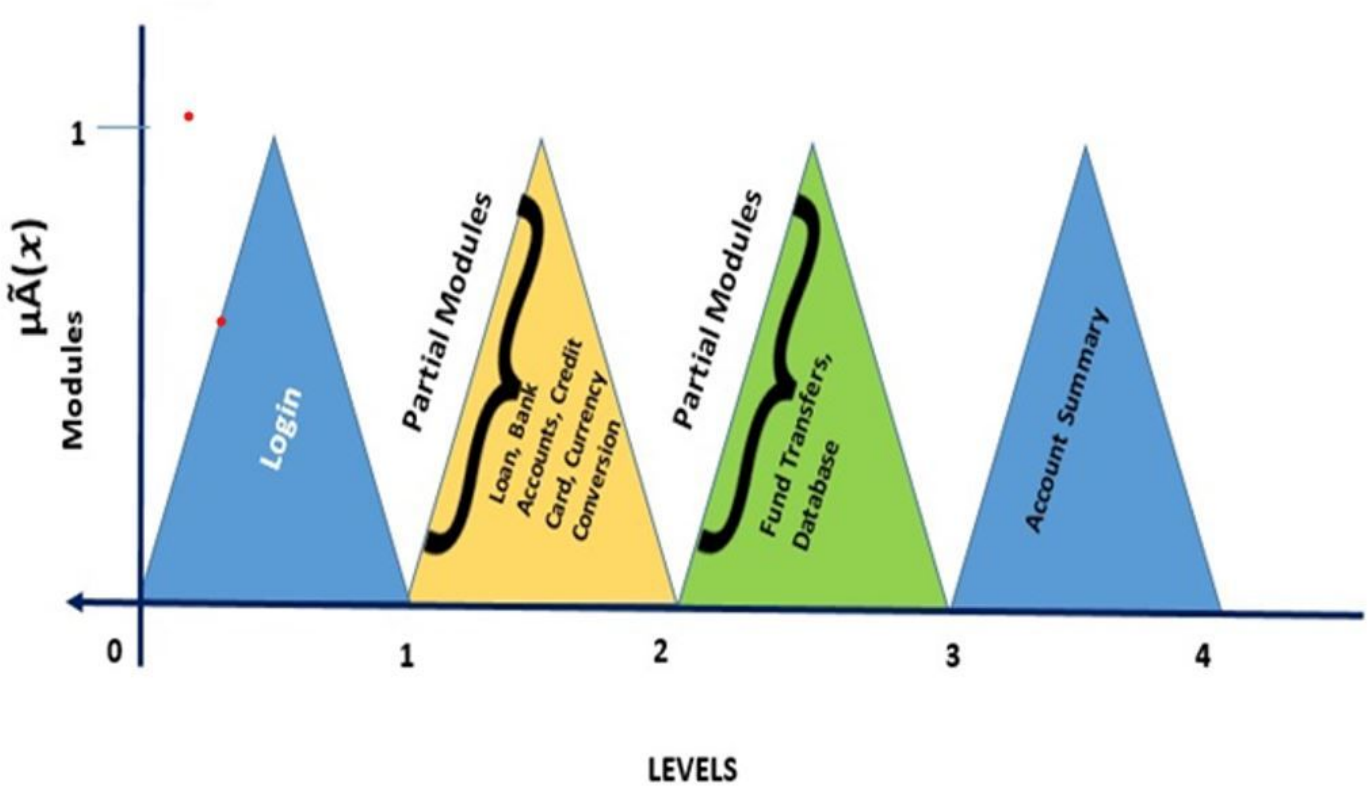

Figure 3

AEAP Sequence of modules represented In fuzzy system

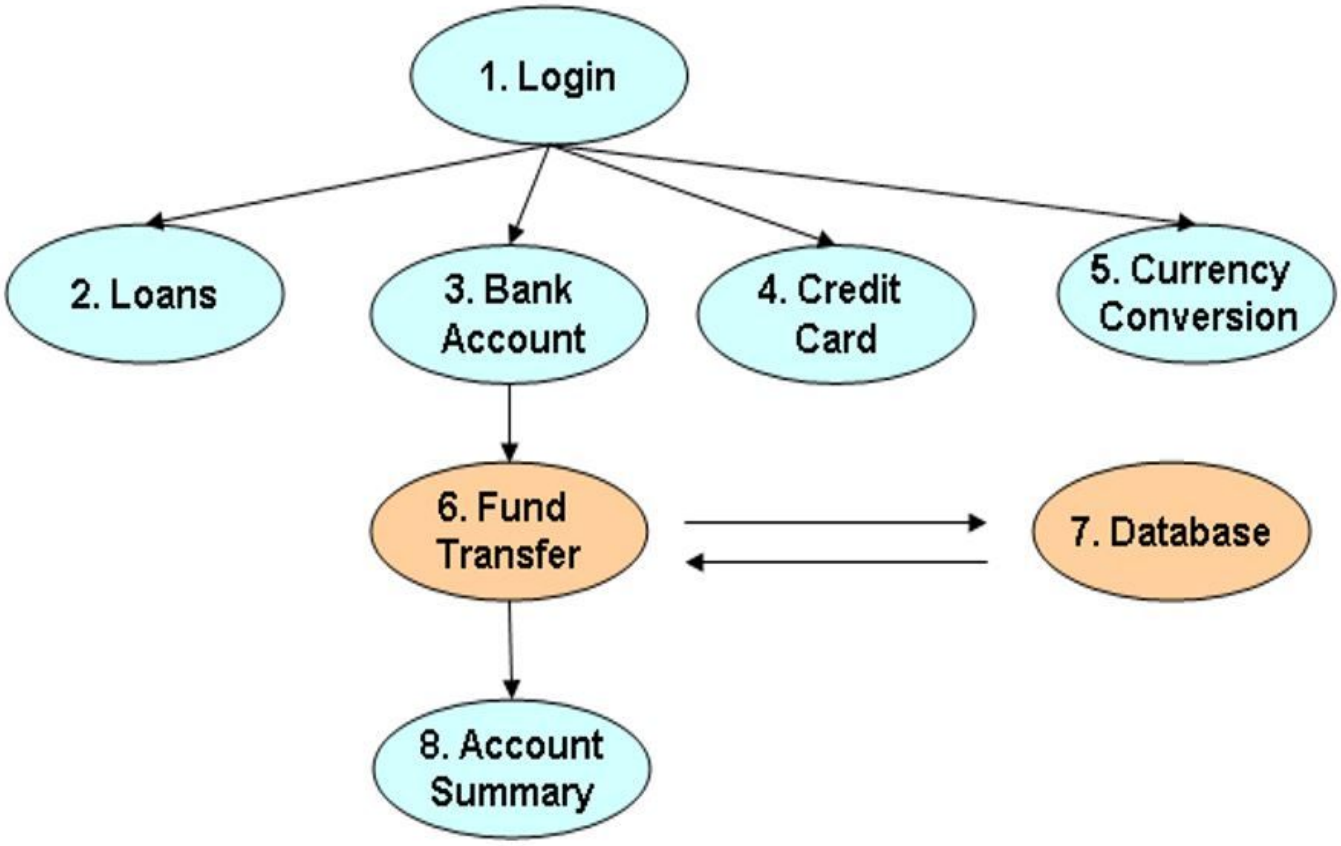

Figure 4

Structure of modules in AEAP

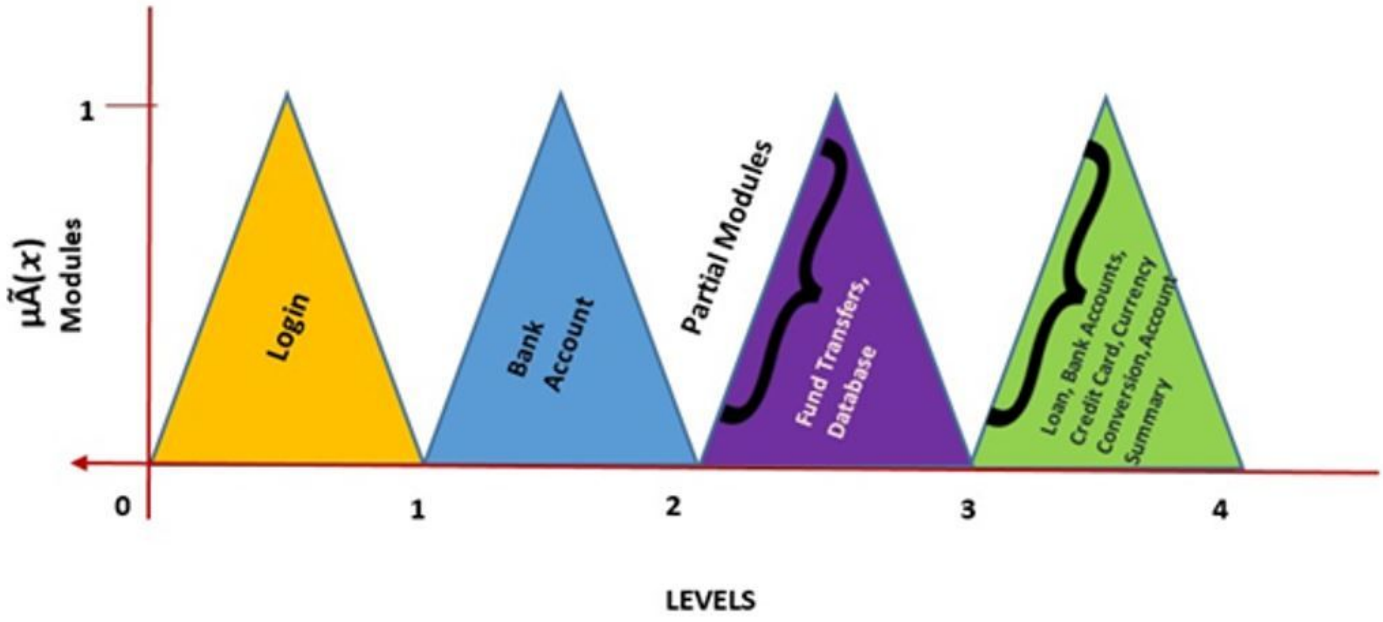

Figure 5

ALAP Sequence of modules represented In Fuzzy systems

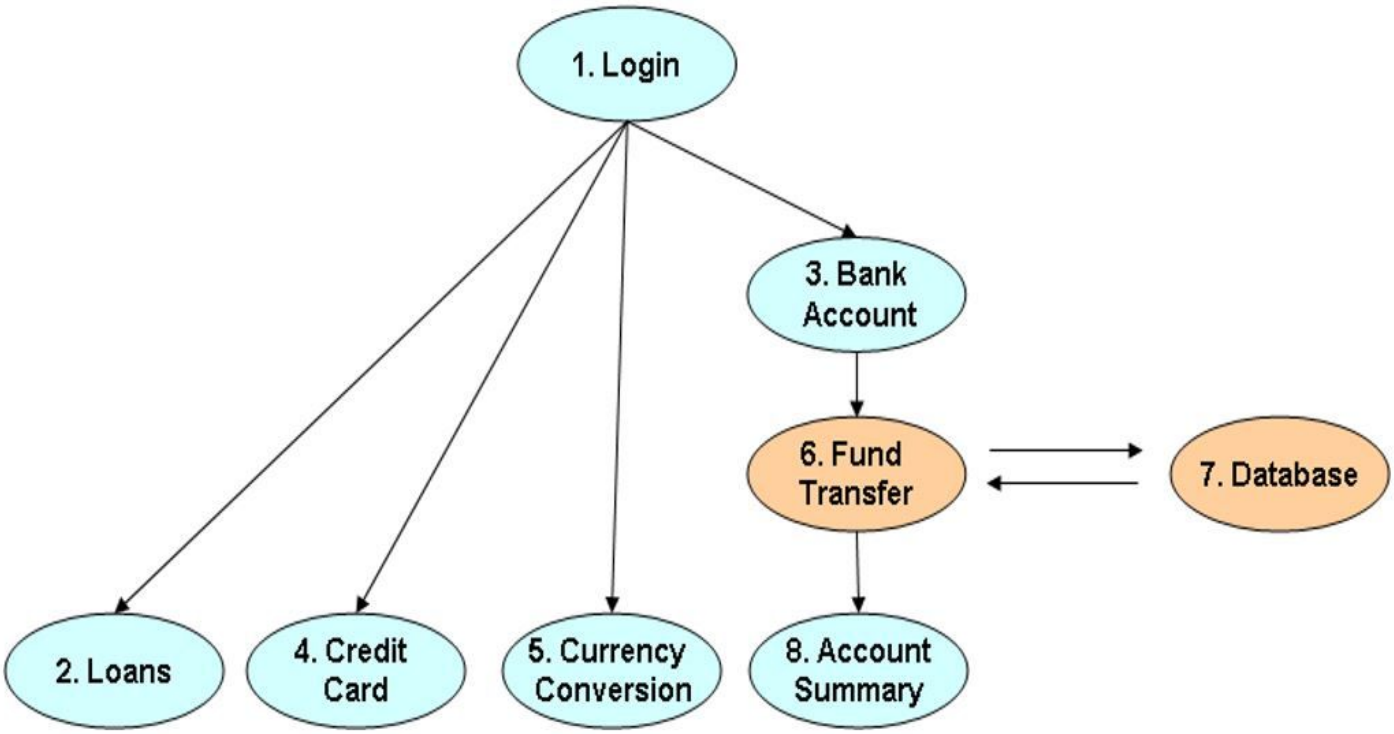

Figure 6

Sequence structure of modules in ALAP

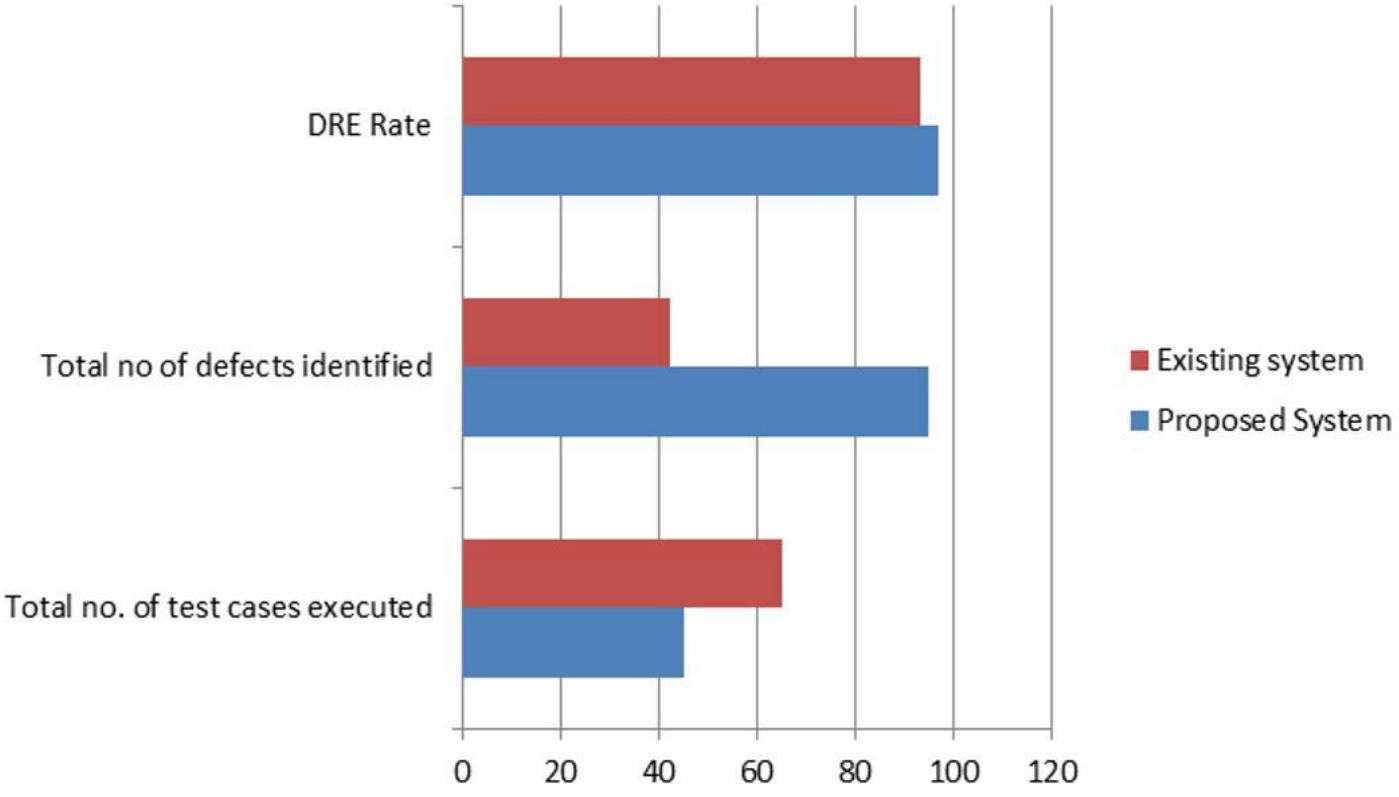

Figure 7

Comparison chart for proposed system.