Debugging and Finding the Errors in Security Testing Techniques

[1]Dr.S.Kannan, [2]Mr.T.Pushparaj
[1]*Research Supervisor,* [2]*Research Scholar,*
Madurai Kamaraj University, Madurai

ABSTRACT

Software testing is significant to decrease mistakes, maintenance and overall software costs. One of the main problems in software testing part is how to get an appropriate set of test cases to test a software system. There is lot of exploration which has been through in past to improve overall testing procedure with determined of refining quality of software in a smallest quantity of time. After assessing all obtainable testing procedures it has been found that dissimilar improvement models are used for dissimilar types of requests and dissimilar testing techniques are achieved to test the same. In this paper main testing approaches and methods are shortly described. There are many methods to software testing, but active testing of complex product is basically a process of investigation, not simply a material of making and following route method. It is often incredible to find all the mistakes in the program. This important problem in testing thus throws open question, as to what would be the approach that we should accept for testing. Thus, the assortment of right approach at the right time will make the software testing competent and effective. In this paper I have described software testing methods which are categorized by purpose.

Keywords *Software testing techniques, umbrella activities, Software Testing strategies, Debugging, Block box testing, White box testing.*

I INTRODUTION

Software testing is the highest activity of assessing and implementing software through a view to find out mistakes. It is the procedure where the system necessities and system apparatuses are exercised and assessed manually or by using automation tools to find out whether the system is satisfying the quantified necessities and the alterations between estimated and actual results are determined. This paper at a high - level is separated into two sections. The first section covers improved testing procedure, which elaborates all stages of the testing life cycle and the second section covers testing types.

The first section accentuates the highest actions, which are Analysis, Development and Preparation, Implementation and Closure. Where closure comprises statement and root reason exploration doings and implementation phase goes hand in hand with bug classification and tracking. The software bug life cycle described in the paper in the coming section highlights the obligatory phases for bug classification and tracking. The test preparation phase comprises test case preparation, test case assortment, test case optimization and test data preparation which is going to be enlarged later in this paper. There are lots of obtainable difficult types like black box testing, white box testing, state created testing, security testing, look and feel testing, receiving testing, system testing, alpha and beta testing, and arrangement based testing, verification and validation testing. Based on the exploration and study complete this paper considered all of them under three high - level testing types, which is Functional, Performance and Security. The last segment deals with the assumption, which shows significance of our elevated software testing procedure and FPS as a basis for testing methods.

However, maximum people difficult in noticing and eliminating those faults would it as an art reasonably than a technique. All bugs stem from a one simple statement: something assumed to be correct, was in fact mistaken. Due to this modest value, truly inexplicable bugs can defy reason, manufacture debugging software challenging. The typical behavior of many inexperienced programmers is to freeze when unexpected problems arise. Without a denote process to follow, solving problems seems impossible to them. The maximum apparent response to such a condition is to kind some random variations to the code, hoping that it will start occupied again. The problem is simple: the programmers have no awareness of how to method debugging. This address is an effort to appraisal some methods and implements to assist non-experienced programmers in debugging. It contains both tips to solve problems and proposals to avoid bugs from establishing themselves. Finding a bug is a procedure of combing what is working until something wrong is found.

Therefore, an algorithm good in every situation should not be expected: there is no silver bullet for debugging. Experience and ingenuity are part of the quest for bugs, but also disciplined usage of tools. The importance of a method of ending errors and axing them during the life-cycle of a software product cannot be stressed enough. Testing and debugging are fundamental parts of programmer's everyday activity but some people still consider it an annoying option. When not carried out properly, consequences can be dreadful. A group associate of the guided-missile vessel USS Yorktown incorrectly arrived a zero as information assessment, which occasioned in a detachment by zero. The mistake cascaded and ultimately shut depressed the ship's impulsion system. The ship was dead in water for numerous hours since a program didn't patterned for valid input.

The miscarriage answerable for damage of the orbiter was accredited to a failure of NASA's system engineering procedure. The method did not identify the system of capacity to be used on the project. As a consequence, one of the increase teams used Imposing measurement while the additional used the metric system. When strictures from one component were approved to alternative, during orbit navigation alteration, no alteration was achieved, subsequent in the loss of the craft. These two renowned bugs, as others in history of software, must create the reader appreciate the position of ending mistakes in software: it is not just an inescapable portion in the increase cycle but vital portion of every software system's life span.

II. SOFTWARE TESTING TECHNIQUES

In this Part the responsiveness is mostly on the different software testing Approaches. Software Testing Methods can be parted into two types:-

2.1. Manual testing

It is a slow procedure and difficult where testing is done statically .It is complete in primary stage of life cycle. It is also named static testing. It is complete by analyst, designer and testing team. Different Manual testing Methods are as follows:

A) Walk through
B) Casual Assessment
C) Procedural Review
D) Assessment

2.2. Automated Testing

In this tester runs the script on the testing device and testing is complete. Automated testing is also called dynamic testing.

Automated testing is further categorized into four types

A) Correctness testing
B) Performance testing
C) Reliability testing
D) Security testing

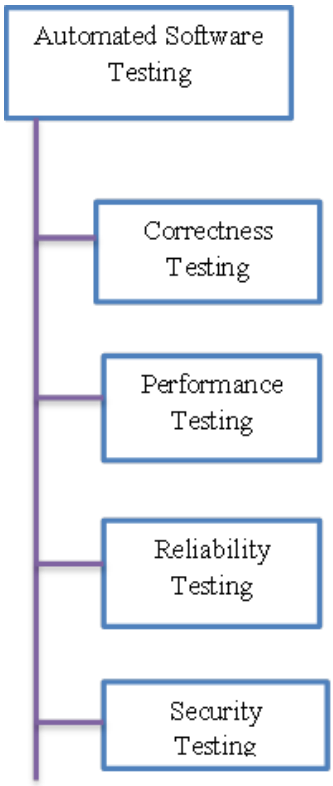

FIG 1 Further classification of Automated software Testing

2.3 Performance Testing

Performance Testing include all the stages as the mainstream testing life cycle as an independent correction which contain approach such as plan, enterprise, performance, analysis and reporting. This testing is directed to appraise the acquiescence of a system or section with detailed performance necessity. Assessment of a concert of any software system comprises resource usage, amount and incentive reaction time. By concert difficult we can amount the features of performance of any submissions.

One of the maximum significant purposes of performance testing is to sustain a low latency of a website, high amount and low exploitation. Characterize two types of presentation testing approximately of the main areas of performance testing are:

1) Determining answer period of end to end connections.
2) Capacity of the interruption of network between Client and server.
3) Observing of method possessions which are under Numerous loads.

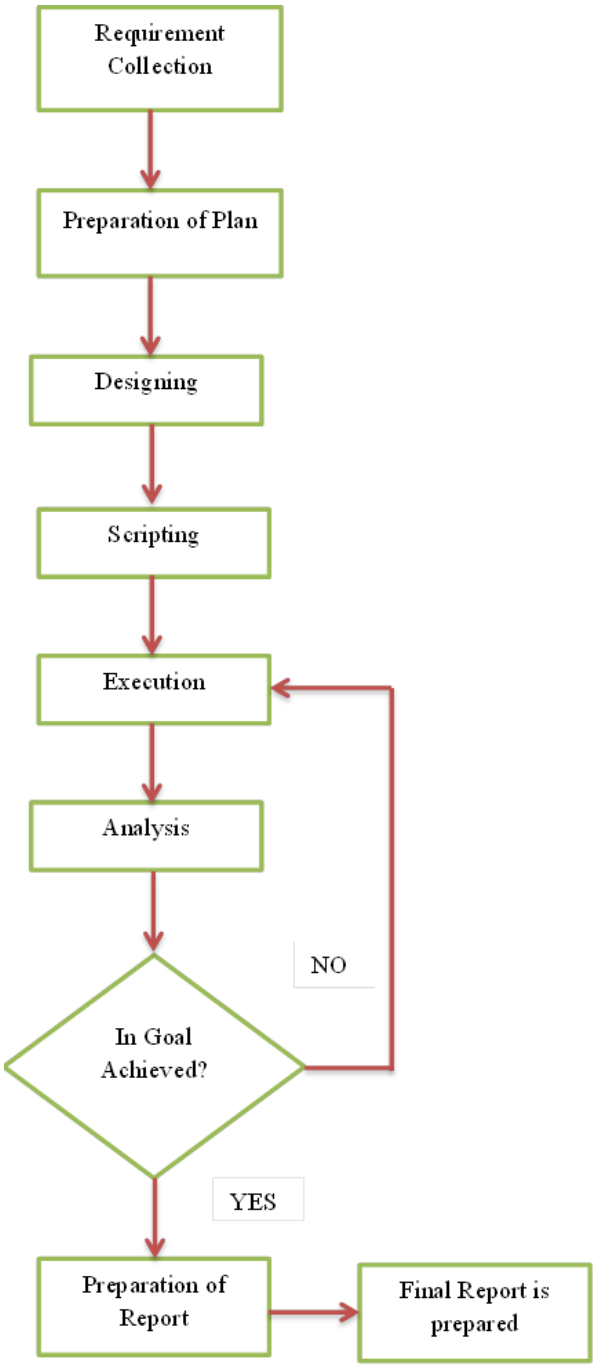

FIG 2 Performance Testing Process

Some of the mutual faults which ensue through Performance testing is:

1) Ignoring of errors in input.
2) Analysis is too complex.
3) Erroneous analysis.
4) Level of details is inappropriate.
5) Ignore significant factors.
6) Incorrect Performance matrix.
7) Important parameter is overlooked.
8) Approach is not systematic.

There are seven dissimilar stages in performance testing procedure:

Phase 1 – Necessity Study
Phase 2 – Test plan
Phase 3 – Test Design
Phase 4 – Scripting
Phase 5 – Test Implementation
Phase 6 – Test Analysis

III. DEBUGGING

After numerous days of suggesting, planning and coding, the programmer lastly has a delightful portion of code. He accumulates it and runs it. Despite being the monarchy of creativity and improbability, a debugging procedure can be separated into four main steps:

1. Restricting a bug
2. Categorizing a bug
3. Thoughtful a bug
4. Renovating a bug

3.1 Localizing a bug

A characteristic assertiveness of unproven programmers concerning bugs is to contemplate their localization an easy mission: they sign their code does not do what they predictable and they are led afield by their assurance in significant what their code must do. This assurance is entirely deceptive because noticing a bug can be precise difficult. As it was explained in the introduction, all bugs stem from the statement that approximately assumed to be right, was in detail wrong. Here is a very humble example of a conceivable difficult.

3.2 Classifying a bug

Although the entrance, bugs have often a mutual background. This permits endeavoring a relatively abrasive, but infrequently useful, organization. The list is decided in order of aggregate exertion (which providentially resources in direction of decreasing frequency).

1) Syntactical Errors

It should be simply gathered by your compiler. I say "should" since compilers, besides being very complicated, can be buggy themselves.

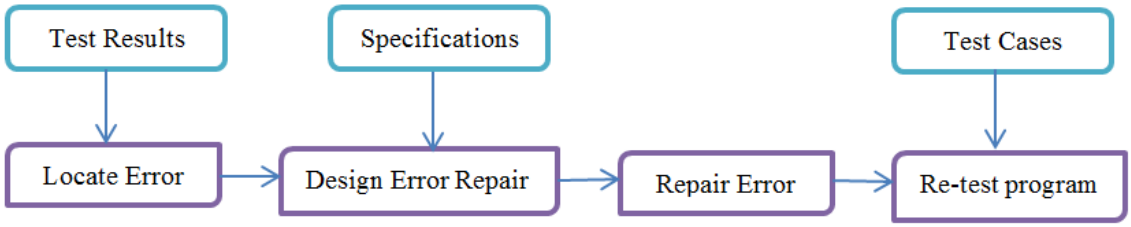

FIG 3 Debugging Processes

In some case, it is energetic to recollect that rather frequently the difficult might not be at the exact location designated by the compiler error message.

2) Build Errors

That originates from connecting object _les which remained not recreated after a modification in some basses. These difficulties can simply be avoided by expending tools to drive software structure, like GNU Make.

3) Basic Semantic Errors

Then include using uninitialized variables, dead code (code that will not ever be executed) and problems with variable types. A compiler can highpoint them to your attention, though it frequently has to be explicitly requested finished flags (cp. 2.1).

4) Semantic Errors

It comprises using wrong variables or operators. No tool can catch these difficulties, since they are syntactically precise declarations, though reasonably wrong. A test case or a debugger (see par. 2.8) is necessary to spot them.

3.3 Understanding a bug

A bug must be fully assumed previously struggling to _x it. Trying to _x a bug earlier thoughtful it totally could end in aggravating even extra impairment to the code, since the difficult could alteration form and patent itself anywhere else, maybe casually. Again, a characteristic example is retention exploitation: if there is any suspicious recollection was corrupted through the implementation of some procedure, all the data complicated in the algorithm must be patterned already trying to alteration them. More about memory exploitation is obtainable.

The subsequent check-list is valuable to promise a accurate method to the exploration:

- Do not complicate perceiving indications with finding the real basis of the problem;
- Check if similar faults (especially wrong assumptions) remained complete elsewhere in the code;
- Confirm that just a programming error, and not a more important problem (e.g. an incorrect algorithm), was found.

3.4 Repairing a bug

The ending step in the debugging procedure is bug protective. Repairing a bug is additional than changing code. Any answers must be recognized in the code and verified correctly. Additional significant, education from mistakes is an operative approach: it is good practice substantial a small file with thorough clarifications about the way the bug was exposed and modified. A check-list can be a useful aid.

Several points are worth recording:

- How the bug was observed, to help in inscription a test case;
- How it was followed down, to give you a improved perception on the method to select in similar conditions;
- What type of bug was faced;
- If this bug was encountered often, in order to set up a approach to avoid it from frequent;
- If the primary suppositions were unfounded; this is frequently the highest purpose why following a bug is so time consuming.

IV UMBRELLA ACTIVITIES

A procedure is defined as a assortment of effort doings, actions, and tasks that are achieved when some work creation is to be formed. Respectively of these doings, movements, and tasks reside inside a framework or classical that describes their association with the procedure and with one another. The software procedure is characterized schematically to the each framework movement is inhabited by a set of software engineering movements. Each software engineering achievement is well-defined by a task set that recognizes the work tasks that are to be concluded, the work products that will be formed, the quality declaration points that will be essential, and the milestones that will be used to designate development.

FIG 4 Umbrella Activities

A mutual process outline for software engineering defines five framework activities communication, progress, modeling, construction, and standing. In intention, a set of umbrella actions development subsequent and control, risk association, quality declaration, preparation association, technical assessments, and others are practical finished the technique. It necessity note that one important part of the software process has not yet been convened. This part named procedure flow designates how the framework doings and the actions and tasks that happen within each framework movement are prepared with deference to arrangement and time. The existence of a software procedure is no assurance that software will be distributed on time, that it will happen the customer‘s wants, or that it will exhibit the practical entrances that will central to long-term excellence characteristics. Process patterns must be attached with solid software engineering exercise.

IV SECURITY TESTING TECHNIQUES

4.1 Model-based security testing

Model-based security testing is an MBT technique that validates software system requirements associated to security properties. It relations security properties like preference, integrity, obtainability, verification, agreement and non-repudiation with a traditional of the SUT and identifies whether the calculated or proposed security assemblies hold in the model. Both MBT and MBST have in company, that the input thing is a model and not the SUT. Consequently the concept gap between the model and the SUT has to be lectured.

In specific, a recognized concern at the classical level does not repeatedly authorize a concern at the SUT. Consequently an additional step is desirable to map an intellectual test case to an executable test case that can be performed on the SUT. Possible imperfections essential to be designated by imperfection hypotheses. In direction to turn these suppositions into functioning adequacy standards, they essential to be captured by some form of categorical defect model. One procedure of defect is a fault, unstated as the root reason of an incorrect scheme state (error) or improper system output (failure). As we show below, susceptibilities can be understood as faults. In addition to categorical replicas of (the functionality of) the system under test, model based security testing typically varieties use of one or more of the three subsequent models for test assortment: possessions, susceptibilities, and aggressors. Models of an attacker encode an attacker's performance: the data they need, the dissimilar steps they take, the way they craft adventures. Attacker replicas can be seen as models of the situation of a organization under test, and knowledge about a targeted susceptibility frequently is left implicit.

4.2 Vulnerabilities as faults

Frequently, as a response to known applicable pressures, effects are endangered by categorical security apparatuses. Mechanisms comprise input purification, Address Space Layout Randomization (ASLR), encryption of password files, but also intrusion detection systems and access control components.

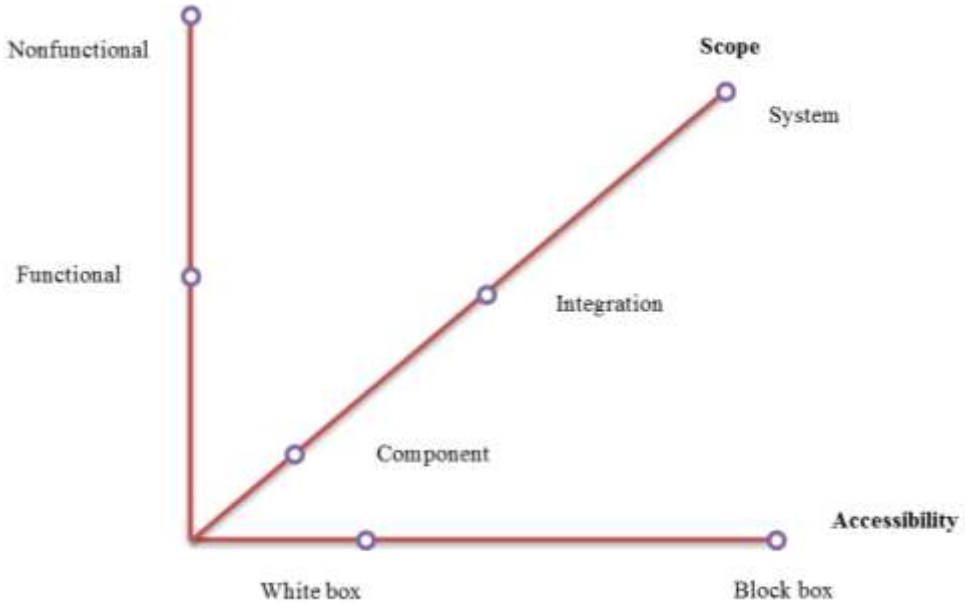

FIG 5 Testing Dimensions

There is a portion of code (the mechanism) that is invented to defend the benefit; or there is no such portion of code. Susceptibility is a singular kind of responsibility with security consequences. It is distinct as the deficiency of a suitably effective mechanism. This can mean both

(1) That there is no mechanism at all (e.g., no input purification receipts residence which can lead to buffer over flows or SQL Injections)

(2) That the instrument does not work appropriately, i.e., is incompletely or inaccurately executed, for occurrence, if an admission device strategy is damaged.

4.3 Classification of Security Testing

Numerous publications have been available that proposition taxonomies and classifications of current MBT and MBST methods. The authors entitlement that MBST requirements to be created on dissimilar types of models and differentiate three types of input models for security test generation, i.e., architectural and functional models, threat, fault and risk models, as well as weakness and vulnerability models. Architectural and practical replicas of the SUT are disturbed with security necessities and their execution. They focus on the predictable system performance. Threat, error and risk models focus on what can go wrong, and distillate on reasons and significances of system botches, weaknesses or susceptibilities. Weakness and susceptibility models describe weaknesses or vulnerabilities by themselves.

V PROPOSED SYSTEM

Towards authenticate our proposed framework, we organized a test bed contained of one server machine and one added machine which hosts numerous customer apparatuses in the method of essential machines. For this determination, we produced four virtual machines on the computer using VMware Workstation 10.0. All these effective apparatus were associated to the server and we connected soft bots on individually of the client machine as well as on the server side. These proxies are in detail a portion of code to interconnect and organize with other managers organized on other client machine as well as server machine.

It's to be achieved the testing actions; we used a web-based request for "Employee Management". This web-based application was organized for a company "Cafedunord". Employee organization is a shift organization platform to make a shift roaster for dissimilar employees of an association. The manager can generate dissimilar shifts and can assign them to dissimilar employees. Employees obtain emails about their entire weekly or monthly working schedule. Manager can also produce shifts connected work report for employees. We arranged 50 functional test cases which relate to dissimilar functionalities delivered by the software such as login, totaling employee, and allocating tasks to the employee, formulating shift schedule, making duty rostrum.

Sr. No.	Type	Testing Environment	Effective (fault detection)	Size of test pool	Testing technique
1	Random Testing	Black	Least effective	Large	Specification based
2	Functional Testing	Black	Effective	Large	Specification based
3	Control Flow Testing	White	Effective	Medium	Code based
4	Data Flow Testing	White	Effective	Small	Code based
5	Mutation testing	White	Most Effective	Small	Fault based
6	Regression testing	White/ Black	Most Effective	Based on Program size	validation

Table 1 Comparison of testing techniques

VI PERFORMANCE ANALYSIS

Some of the specific test cases are attached as Annex-1 to this statement. The organized test cases were then changed into test scripts consuming the Selenium testing tool. The test cases were run on the Selenium using the "record" selection, for which Selenium organized the test scripts consequently. We saved these test scripts for future use. The test scripts, which were in executable form, were then approved on to the test manager for circulation to the client apparatuses for their implementation.

Then designated a web-based application called "Cafedunord employee organization system" to test its functionalities for authentication of our framework. It is essentially a shift organization submission for the employees. We can generate dissimilar shifts e.g. day-shift or night-shift and then we can assign them to dissimilar employees. Employee obtains their complete weekly or monthly schedule by email. We have produced 50 test cases in the establishment from which we particular specific test cases to perform testing which are designated in Table 1. Our framework is a allocating functional testing with multi-agent in which we must a server and a minor bunch of client machines. In our test bed, the client machines are not physical machines but are effective machineries formed using VM-ware Workstation. For the difficult determination, we use Selenium computerization testing tool. Selenium is a collection of tools to industrialize web browser across many platforms. This testing tool is free and open source software.

So, we can initial create a circulated organization and then generate numerous tests cases, which will be implemented in this disseminated test environment. Selenium is powerful tool which can work with dispensing environment and we can also greatest a test script for a specific test case. When we have to authenticate a test case, we will run its correspondent test script. Selenium automation difficult tool and that are be organized on server and client machines. Software manager is in fact a part of code snippet that displays and controls all the work connected to statement and collaboration between the system nodes. To initiate with the testing, all of our test cases are located in the Test Suite Repository. Test Controller makes the test cases from the repository. In Test Controller milieu, we use Selenium to make test scripts for those cases and managers will allocate those test scripts between the client machines depending upon the load on individually device.

Each client apparatus will authenticate the assessment script using challenging tool and will deliver the result. Agents can segment those consequences with respectively other comprehensive message passing which can generate an immediate implementation of test cases as well as dependable and robust testing atmosphere. Every test consequence is directed to the Test Analyzer. Test Analyzer appraises the fountain with the test case position whether the test has remained approved or unsuccessful. Test Controller can run the failed test cases again at a later period.

Functionality	Test cases	Passed	Failed	Remarks
Employee Authentication	8	6	2	The logoff functionality was not occupied correctly.
Registering New Employee	7	7	3	Alteration password selection on the first login attempt was not energetic. Also employee's roles alteration was not being approved out.
Shift Organization	15	14	1	All the test cases approved in this module.
Preparation	19	17	2	Erroneous performance detected while allocating off days. Similar off day could be allocated to all the employees.
Produce Reports	7	5	3	Two of the immediate reports in the menu list did not produce everything.

Table 2 Test Case Execution

Test Cases	Quantity	Percentage
Passed	50	76%
Failed	8	16%
Deferred (Error in test script)	4	8%
Total	62	100%

Table 3 Passed and Failed Test Case Execution

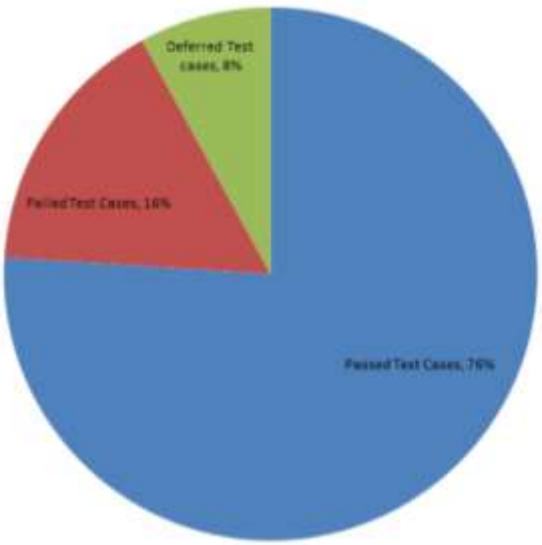

Fig 6 Test Execution Summary

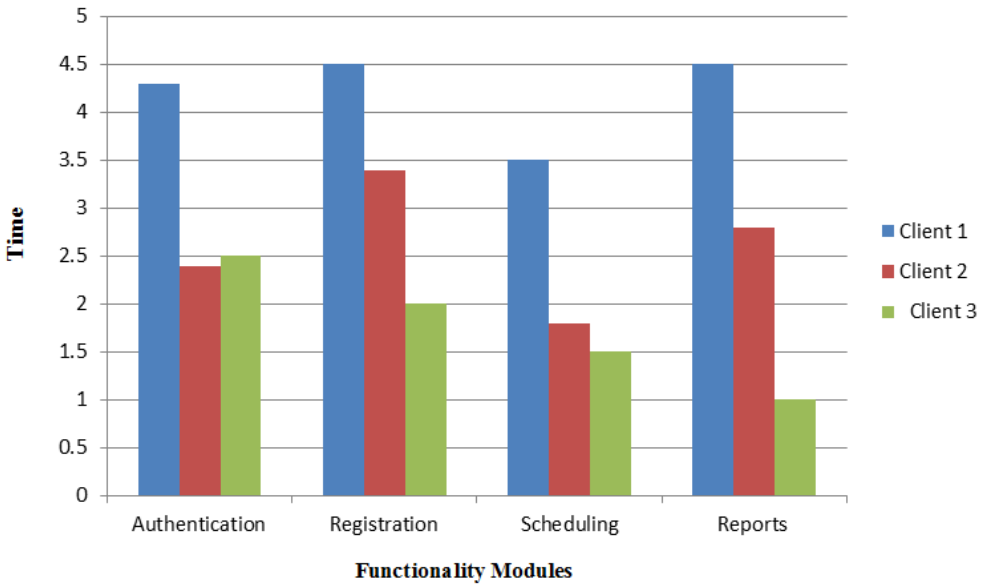

Fig 7 Execution Time Analysis

VII CONCLUSION

Testing requires remained extensively used as a method to help engineers develop high-quality systems and the methods for testing have progressed from ad hoc activities resources of minor group of programmers to a prepared correction in software engineering. The core purpose for execution testing actions in the dispersed situation was to decrease the cost, time and determinations generally required to perform functional testing. We return to the problem that how to get an appropriate set of test cases to test a software system and find out mistakes. A portion of time is consumed on Functional testing and there is hardly any software which grew crashed due to lack of functional testing in current past. So this paper proposed a new accurate mix of testing which must comprise some concert and security testing checks in totaling to functionality testing for improved quality of software. As there is continuously a possibility so Additional to this paper a exploration and study can be complete on the software testing to intend a generic testing framework and methods to support practical, performance and security testing for object oriented improvement framework and added platforms consuming some algorithm with/ without use of tools in smallest quantity of period.

REFERENCE

[1] Somerville, I, "Software Engineering", 8th edn. , Addison Wesley, 2006.
[2]K.K Aggarwal, Yogesh Singh, "Software Engineering", 2003.
[3] Testing Computer Software, by C. Kaner, J. Falk, and H. Nguyen volume 2 Issue 3 March 2015.
[4].N.Divya, M.Sudarson, An Efficient Secure Node Selection with Trust Based Aware For QOS in Mobile, volume 2 Issue 1 Jan to Feb 2015.
[5]. N.Suresh, An Overview of Object Oriented Software Testability, volume 1 Issue 1–Feb 2014.
[6]. N.Sendhil Kumar1 D.Dilli Babu2 C.Thulasiram, Secure Data Aggregation for the Wireless Sensor Networks, volume 2 issue 5 May 2015.
[7]. S.Kokila1 , T. Princess RaichelSoftware as a Service, a Detailed Study on Challenges and Security Threats, volume 2 issue 12 December 2015.
[8]. Dr.G.J Joyce Mary Asso. Prof,, R.Kokila, Secure Data Aggregation in Mobile Sensing, volume 2 Issue 3 March 2015.
[9] Khan, Mohd Ehmer, and Farmeena Khan. "A Comparative Study of White Box, Black Box and Grey Box Testing Techniques." *International Journal of Advanced Computer Sciences and Applications* 3, no. 6 (2012): 12-15 .
[10]Grey Box Testing from Wikipedia available at http://en.wikipedia.org/wiki/Gray_box_testing.
[11]Swain,Kumar,Santosh.Mohapatra,Durga,Prasad.Mall,Rajib.2010. Test Case Generation Based on Use case and Sequence Diagram. *International Journal of Software Engineering*(IJSE). Swain et al.3,2(July 2010).
[12]Akhilesh,Babu,Kolluri.K,Tameezuddin.Kalpana,Guddikadula. 2012.Effective Bug Tracking Systems. Theories and Implementation", *IOSR Journal of Computer Engineering* ISSN:2278-0661 Volume 4,Issue 6(Sept-Oct 2012), pp 31-36.
[13]Rina,DCSK,KU,Haryana,INDIA,Tyagi,Sanjay.DCSA,KU,Haryana.2013.AComparative Study Of Performance Testing Tools. IJARC SSE. 3,2(May 2013).
[14] Karen ,Scarfone.2012. Intro to Information Security Testing & Assessment.*ScarfonecyberSecurityCsr.nist.gov*.(7June 2012).
[15] B,Beizer.1990.Software Testing Techniques. Technology Maturation and Research Strategies Carneige Mellon University Pittsburg, USA.
[16] B.Beizer .1995.Software Testing Techniques.2006.Van NostrandReinhold,New York.1990.ISBN.0-442-20672-0.(31.Oct.2006).

[17] A,Bertolino.2001.Chapter 5: Software Testing . IEEE SWEBOK trial version 1.00.IEEE(May 2001).

[18] Khan,Mohd.Khan,Farmeena.2012.A Comparative Study of White Box, Black Box and Grey Box Testing Techniques.2012. *International Journal of Advanced Computer Science and Applications*(IJACSA). Vol. 3.No.6.(2012).

[19] Tarika,Bindia. Computer Programmer CSE,GNDEC,Ludhiana,Punjab-India.IJRITCC.2,1 .68-72.(2321-8169).

[20] B,Swarnendu.R,Mall.CSeDeptt,IIT Kgp.2011.Regression Test Selection Techniques, A Survey-Informatics 35 .2011.

[21] Swain,S.k.Mohapatra,D.P.Mall,R.2010.Terst Case Generation Based on Use Case and Sequence Diagram. *International journal of Software Engineering* (IJSE).3, 2.(July 2010),(289-321).

[22]Thakre,Sheetal.Chavan,savita.Chavan,P.M.]2012.Software Testing Strategies and Techniques. International Journal of Emerging Technology and Advanced Engineering .Website: www.ijetae.com .2, 4.(April 2012), (2250-2459).

[23] An Approach to Cost Effective Regression Testing in Black-Box Testing Environment - IJCSI International Journal of Computer Science Issues. 8, 3, 1(May 2011),(1694-0814).

[24] Chauhan,Kumar,Vinod.2014.Smoke Testing- International Journal of Scientific and Research Publications4,2 (February 2014),(2250-3153).

[25] Gupta,Varuna.Sen,Saxena,Vivek.2013.Software Testing: Smoke and Sanity- International Journal of Engineering Research & Technology (IJERT).2,10(October 2013) (2278-0181).

[26]Liskin,olga.Hermann,christoph.Knauss,Eric.Kurpic,Thomas.Rumpe,Bernhard.Schneida,Kurt.2012.Supporting Acceptance Testing in Distributed Software Projects with Integrated Feedback Systems: Experiences and Requirements. IEEE Seventh International Conference on Global Software Engineering.(2012).

[27] Yoo, Shin. Harman, mark2012.Regression Testing Minimization, Selection and Prioritizations.

A Survey. King's College London. Centre for Research on Evolution, Search &Testing. Strand, London, WC2R 2LS, UK.22,2 (March 2012) (67-120).http://onlinelibrary.wiley.com/resolve/doi?DOI=10.1002/stv.430.

[28] Sumalatha, Mary.Raju,G.2013. Object Oriented Test Case Generation Technique using Genetic Algorithms. International Journal of Computer Applications (0975-8887). 61, 20 (January 2013).

[29] Ostrand,T,J.Balcer,M, J.1988.The category-partition method for specifying and generating functional tests. Communications of the ACM 31 ,6(June 1998) (676-686).doi>10.1145/62959.62964.

[30] Nirpal, B,Premal.Kale,K,V.2011.Using Genetic Algorithm for Automated Efficient Software Test Case Generation for Path Testing. Int. J. Advanced Networking and Applications.(911-915).02,06 (January 2011).