ISSN 2278-3091
Volume 8, No.6, November – December 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse129862019.pdf
https://doi.org/10.30534/ijatcse/2019/129862019

Exploring the Essentials and Principles of Software Development

Sandeep Dalal[1], Kamna Solanki[2], Sudhir[3], Diksha[4]
[1,2]Assistant Professor, Maharshi Dayanand University, Rohtak, India
Sandeepdalal.80@gmail.com, Kamna.mdurohtak@gmail.com
[2,3]Research Scholar, Maharshi Dayanand University, Rohtak, India
Sudhir.gehlawat@gmail.com, Dikshanandal3012@gmail.com

ABSTRACT

There is utmost pressure on software development industry to build quality software's as we have become highly dependent on software's directly or indirectly. Software development has many phases like testing which requires the maximum time as well as resources like human efforts and investment. The time required for completion of software testing is more than the time required to complete all remaining phases of software development. The quality of software heavily depends on quality of software testing process. Therefore, Software testing can be considered as necessary evil for verifying the software quality. This paper discusses some of the most critical aspects and principles of software development and testing.

Key words: Software Engineering, Software Testing Principles, Software Testing, Software Development.

1. INTRODUCTION

"It is the study and an application of engineering to the design, development, and maintenance of software". Fairly defined software engineering as "*Software engineering is the technological and managerial discipline concerned with the systematic production and maintenance of software products that are developed and modified on time and within cost estimates*" [1]. IEEE defined software engineering as "*Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software*" [2]. Baur defined Software Engineering as "*The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines*" [3]. The essential objective of software engineering process is to develop and timely deliver a quality product that can satisfy customers. It aims to accomplish this objective by using an "engineering approach" [21]. Engineering includes a set of key rules that ought to be followed to build a quality product.

2. SOFTWARE DEVELOPMENT LIFE CYCLE

"*Software Process is a coherent set of activities for specifying, designing, implementing and testing software systems*" [4]. SDLC outlines a detailed and thorough plan to build, test, maintain and modify any software product.

Requirement Analysis: The most substantial and important phase of "Software Development Life Cycle (SDLC)" is requirement analysis [5]. This phase focuses on capturing the requirements of end-user in unbiased manner. User requirements can be of two types: "functional requirements" and "non- functional requirements". "*A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. This should be contrasted with functional requirements that define specific behavior or functions*" [6]. Sometimes, end user is unable to clearly elicit their requirements or an analyst face problem in understanding user requirements. Therefore, requirement elicitation techniques can be utilized by the requirement analysts to reveal and capture the user requirements [7]. The huge criticality of capturing user requirements lies in the fact that any mistake in capturing user requirements can cause tremendous loss to the software organization leading to un-necessary wastage of cost, time and efforts as shown in figure 1.

Therefore, a requirement analyst must pay attention to clearly and unambiguously capture the requirements. The information gathered in this phase provides base to the subsequent steps in software development. Risk assessment and planning related to various types of software feasibility like operational, economic, social, technical feasibility is also carried out in this phase.

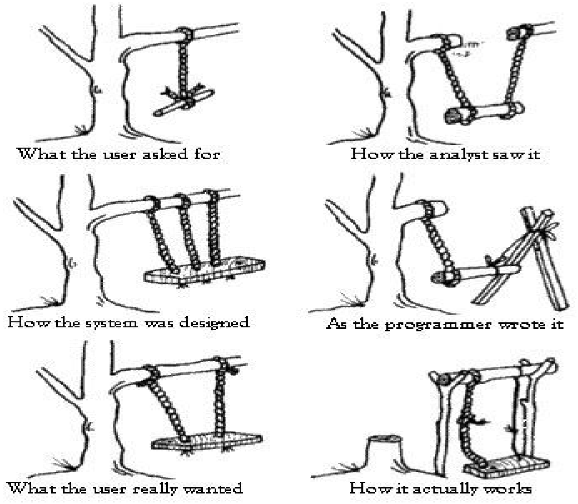
What the user asked for
How the analyst saw it
How the system was designed
As the programmer wrote it
What the user really wanted
How it actually works

Figure 1: Significance of Requirement Analysis Phase

Requirements captured are documented in Software Requirement Specification (SRS) document. SRS consists of all "functional and non-functional" user requirements within the scope of the software. SRS serves the purpose of final agreement between software organization, stakeholders and end user. SRS is finalized when all the stakeholders agree to the terms and conditions of SRS [8].

Software Design: An architectural design for the software is planned depending on the user's requirements mentioned in SRS. "*Software design* is a process to transform user requirements into some suitable form, which helps the programmer in *software* coding and implementation" [9]. A high-level software design specifies the layout of modules, their interactions and interfaces, overall structure of the software. A low-level software design specifies the finer details of modules of the software product like data flow representation, data structures, intra-module utilities and communication strategies. The final design is documented as a Design Document Specification (DDS).
All the finer details related to inner and interface design of every module must be described precisely in the DDS architecture.

Coding: The software programmer or developer writes the actual code using some high-level programming language in this phase. A software code is always written as per the software design to build a software that satisfy the user requirements documented in SRS. The optimal programming language is chosen by the developers based upon the type of user requirements as every language has its own technological strengths and limitations [5] [6].

Software Testing: Software testing can also be stated as verification and validation. Software testing is carried out after coding to find the faults/defects in the software code. Testing remains most time consuming, costly and most significant phase among all the phases. Software testing can be carried out in static or dynamic manner. Static software testing is performed iteratively at the end of every phase in terms of reviews, inspections, walk-throughs. The center objective of static testing is to grab any kind of error which may have penetrated in the software at the earliest. Static testing or verification deals with the question: "Are the building the product, right?" It confirms that the software programmers pick right strategy and processes to develop the software product [7] [8].

Dynamic testing deals with actual execution of test cases on software code to discover software bugs (or failures). Execution of t est cases during dynamic testing triggers the software faults/defects to detect software failures (or bugs). A software failure is nothing, but a fault that gets activated when test cases are executed. A fault or human error is the root cause of a software failure. Dynamic testing or validation deals with the question: "Are we building the right product?" It confirms that the developed software product is right and fulfills the requirements mentioned in SRS as shown in figure 2.

Dynamic testing scans and scrutinize the code by executing test cases, while static testing scans every phase of software development without actually executing software code. The focus of static testing is on revealing software and the focus of dynamic testing is on finding failures (or bugs). Henceforth, testers carry out testing keeping a negative psychology to find maximum faults and bugs. "Software verification and validation use both static and dynamic techniques for ensuring that the resulting software meets all the customer's requirement specifications in SRS" [7].

Debugging starts subsequent to software testing. Debugging deals with fixing the bugs uncovered during testing [11]. Bugs or failures revealed during testing are reported to the developers/programmers and he developers fixes the bugs by making changes in the software code.

Deployment and Maintenance: Soon after the completion of software testing and debugging, a software becomes ready for installation and implementation and it is released formally. After the release of software, maintenance is required iteratively for enhancing the functionalities, technological updates, managing compatibility issues or bug fixing [10]. Three types of maintenance can be defined as:

- o Corrective: To rectify and fix the older or new bugs.
- o Adaptive: To make technological advancements and compatibility adaptations
- o Perfective: Adding the latest features and functionalities to the software.

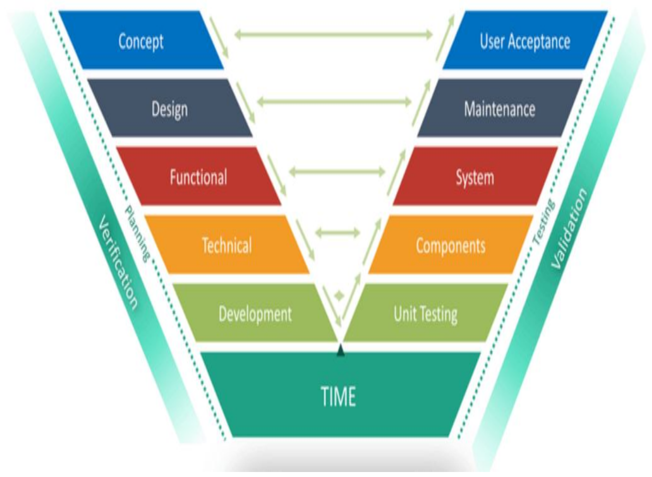

Figure 2: Verification and Validation Process [17]

3. SOFTWARE TESTING

This section describes the software testing process, principles, types and levels in detail. Testing is structured towards evaluating a software system with the intention of finding maximum no. of failures or bugs. In other words, testing deals with executing a software program to detect any quality gaps, errors or missing requirements as compared to the actual desire or requirements. Testing performs the verification and validation of a software product to ensure its accuracy and correctness [4].

IEEE described testing as "*The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component*" [2] and "*A process of analysing a software item to detect the differences between existing and required conditions (defects/errors/bugs) and to evaluate the features of the software item*" [2].

3.1 Software Testing: Concepts and Terminologies

Test Cases and Test Suite: "A *test case* is a document, which has a set of *test* data, preconditions, expected results and post conditions, developed for a particular *test* scenario in order to verify compliance against a specific requirement" [2]. A test case typically answers the question "What I am going to test?" A test case basically checks for any discrepancy between the observed and expected output by entering a set of input values (test data) under some pre-specified conditions. If the expected and observed output are same, then the test case is considered as "pass", otherwise a test case is considered as "fail". Every test case written by the testing team serves a unique purpose of finding some defects or faults. Writing relevant, useful and effective test cases is as important as software testing process itself. The test cases must have following characteristics [5]:

- Test cases must be written to ensure that it provides a sufficient coverage to all the functionalities.
- Test cases must ensure coverage of all possible negative inputs from valid and invalid domains and ranges.
- Test cases must be refined, updated and modified iteratively and continuously.
- Test cases must be stored for future use.

A test suite is a collection of similar test cases. It is sometimes known as "Validation Suite". A test suite provides an organized way to manage individual test cases. ("If an individual file on disk is considered as a test case, then a folder having many similar files can be considered as a test suite".)

3.2 Software Testing Life Cycle

The software testing life cycle can be defined in the following steps:

1) *Review Requirements*: A software testing life cycle begins with reviewing the user requirements to ensure that any sort of amendments or discrepancy in user requirements can be captured as depicted in figure 3.

2) *Define Test Strategy*: Second step deals with defining a test plan or test strategy. A test plan is a document or artifact describing the plan or strategy to effectively test a software? A test plan answers the question: "How it should be tested?"

A test plan designs the layout of testing strategy, resources required for testing, testing objectives, estimation of test schedule, test adequacy criteria and deliverables during testing. A test plan is basically a blueprint for carrying out software testing as described in figure 4. Every finer and minute details of test plan is well controlled and managed by test manager [6] [7].

3) *Develop Test Specifications:* This step carries out the test plan execution in detail. Test cases are written and test environment for execution of test cases is designed. Test data for test cases is prepared carefully to fulfill the objectives of software testing.
4) *Execute Tests:* This step deals with actual execution of test cases. A test case when executed results in either "pass" or "fail". The failed test cases show the presence of defects and are reported to the developers for fixing.

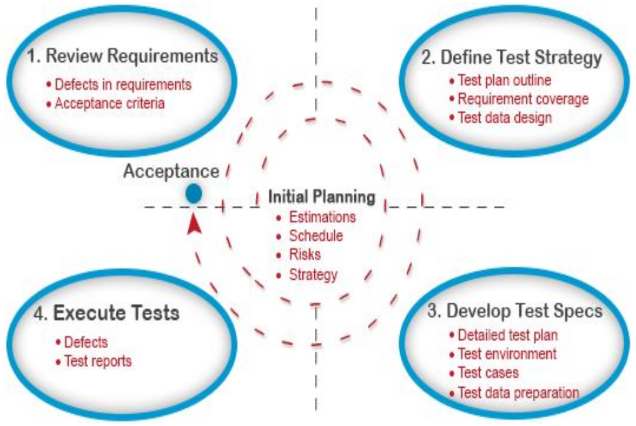

Figure 3: Software Testing Cycle

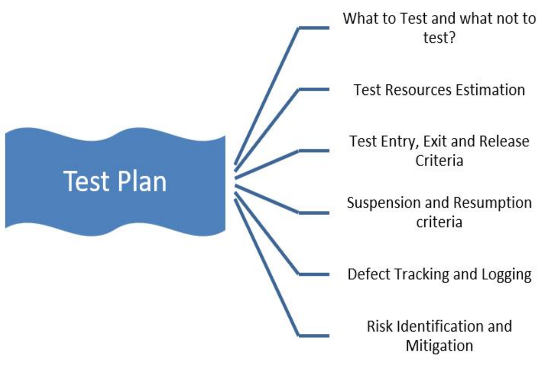
Test Plan
What to Test and what not to test?
Test Resources Estimation
Test Entry, Exit and Release Criteria
Suspension and Resumption criteria
Defect Tracking and Logging
Risk Identification and Mitigation

Figure 4: Software Test Plan

3.3 Types of Software Testing Techniques

"Static Testing Techniques":
Static testing ("Non-execution-based testing") techniques are primarily used to verify the software program without actually executing the software program. "Static techniques are concerned with the analysis and checking of system representations such as the requirements documents, design diagrams and the program source code, either manually or automatically, without actually executing the code" [10]. Static testing techniques includes program analysis, code inspection, model checking and symbolic analysis etc. program documents like SRS, DDS, test reports, test plan document etc. are analyzed manually by a committee of reviewers [9] [11].

"Dynamic Testing Techniques":
Dynamic testing or execution-based techniques actually executes the software written by the developers using some test cases or test suites. This is the best approach to produce quality software. Test cases are designed with utmost care by the testing team to uncover all possible range of valid and invalid inputs [12]. "*The software modules are tested by executing test cases using real or simulated inputs, under both normal and abnormal environment using controlled and expected conditions to evaluate and analyze how a software system reacts to various input test data. It is highly mandatory to test the software in controlled and expected conditions as a complex, non-deterministic system might react with different behaviors to a same input, depending on the system state*" [9]. The significance of dynamic testing can be easily understood as the result of a dynamic testing i.e. passed or failed test cases, one can measure the quality levels achieved so far by the software system [11][22].

Dynamic testing techniques are again classified into three broad categories based on a whether one requires the information about the source code for test case design or not as shown in figure 2.6. In case the knowledge of the source code is not required for test case design, it is known as black box testing. One needs to have knowledge of source code for designing test cases; it is known as white box testing [13]. Grey box testing is the combination of white and black box testing. Grey box testing deals with testing the functionalities of a software by knowing its internal details. White box testing has two different starting points for dynamic software testing: the requirements specification and internal structure of the software. Black box testing evaluates only the external view (behaviour) of the software as it concentrates on what the software does and is not concerned about how software does it. Black box testing totally focuses on testing requirements and functionality of the software under test" [8].

One must have deep understanding of requirement specifications of SRS (Software Requirement Specification

Document) so as to successfully implement the black box testing strategy. In addition, one must know the expected behavior of the system response to a particular input as shown in figure 5.

White box testing strategy focuses on evaluation of internal logic and construction of the software code and relies on the deep information regarding software design and software code for test case design. "White box testing is also known as *Clear Box testing*, *Open Box testing*, *Structural testing*, Transparent Box testing, Code-*Based testing*, and *Glass Box testing"* [11]. "The test cases designed using the white box testing strategy include coverage of the software code in terms of branches, paths, statements and internal structure of the code, etc. White box testing approach examines the logic of the program or system without bothering about the requirements of software which software is expected to fulfil" [4]. The expected results are evaluated on a set of coverage criteria. "Therefore, one needs to have knowledge of coding and logic of the software to implement white box testing strategy" [7].

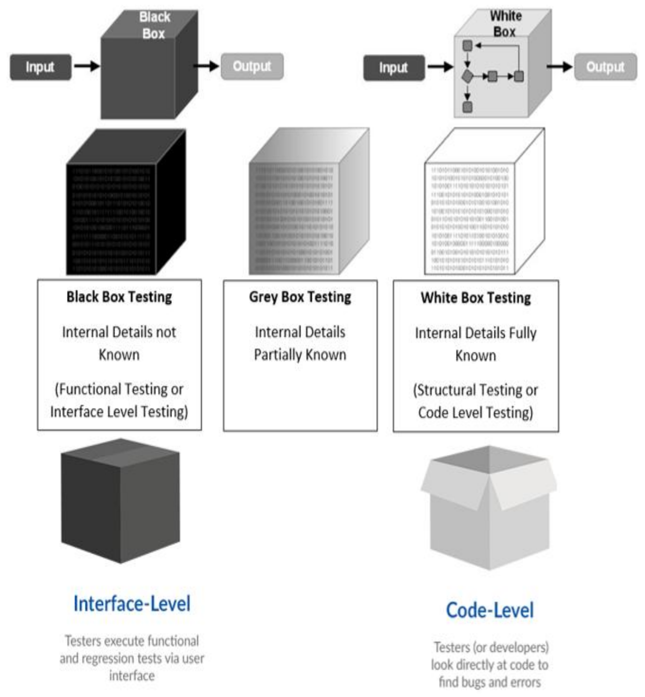
Black Box
Input
Output
White Box
Input
Output
Black Box Testing
Internal Details not Known
(Functional Testing or Interface Level Testing)
Grey Box Testing
Internal Details Partially Known
White Box Testing
Internal Details Fully Known
(Structural Testing or Code Level Testing)
Interface-Level
Testers execute functional and regression tests via user interface
Code-Level
Testers (or developers) look directly at code to find bugs and errors

Figure 5: Static and Dynamic Software Testing

3.4 Levels of Software Testing

Software testing can be conducted at three levels namely "unit testing, integration testing and system testing". Unit testing deals with testing of individual modules or components independently to detect coding faults. Integration testing combines two or more modules and test them jointly to reveal design faults in interfaces and communication as depicted in figure 6. System testing tests the integrated system as whole to validate the software against the requirements specified in SRS. After completion of system testing, acceptance testing is carried out in the presence of end user to validate the client needs and acceptance to functionalities, interfaces etc. Acceptance testing is generally performed after system testing.

The complete software system is tested by the end users for usability and acceptance before the system is actually handed over to the customers or end users using testing tools[23][24]. It is also called as "beta testing, application testing, and end-user testing". Acceptance testing boosts the morale of the developers regarding the correctness and usability and the software can be released to end-user after acceptance testing is passed by the customers or end-user [14].

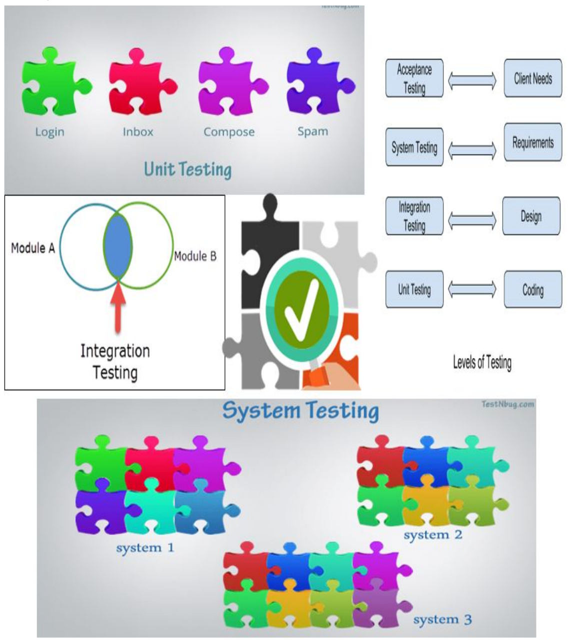

Figure 6: Levels of Software Testing

3.5 Seven Principles of Software Testing

- *Principle 1: "Testing only shows presence of bugs, not their absence"*

Efficient and effective testing of a software can detect the presence of bugs, but it can never guarantee about the absence of the bugs. No matter how thorough and rigorous a system is tested, it can never be promised that a software will be completely bug-free.

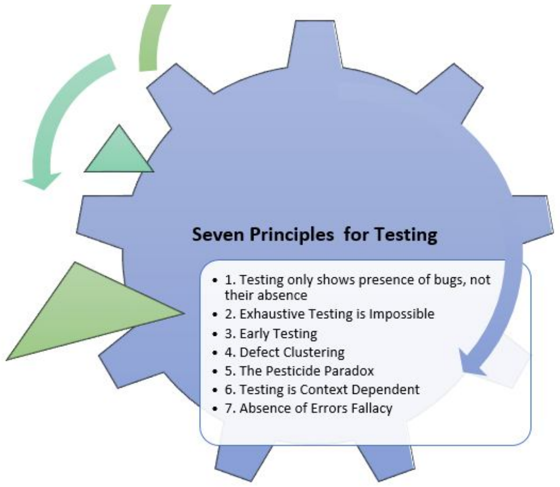
Seven Principles for Testing
1. Testing only shows presence of bugs, not their absence
2. Exhaustive Testing is Impossible
3. Early Testing
4. Defect Clustering
5. The Pesticide Paradox
6. Testing is Context Dependent
7. Absence of Errors Fallacy

Figure 7: Principles of Software Testing

- *Principle 2: "Exhaustive Testing is Impossible"*

Exhaustive testing is a type of testing approach which states that a software program must be tested for all possible combinations of test inputs to ensure that a software program becomes completely bug free. But the limitation of this approach is that exhaustive testing can be performed only for very small programs. It is practically impossible to perform exhaustive testing on a big software system due to cost and time constraints. Ideally, a software program must be tested for every possible combination covering all valid and invalid range of input variables by the software testing team [15]. Hence, complete testing (or exhaustive testing) is just not possible due to time and resource constraints.

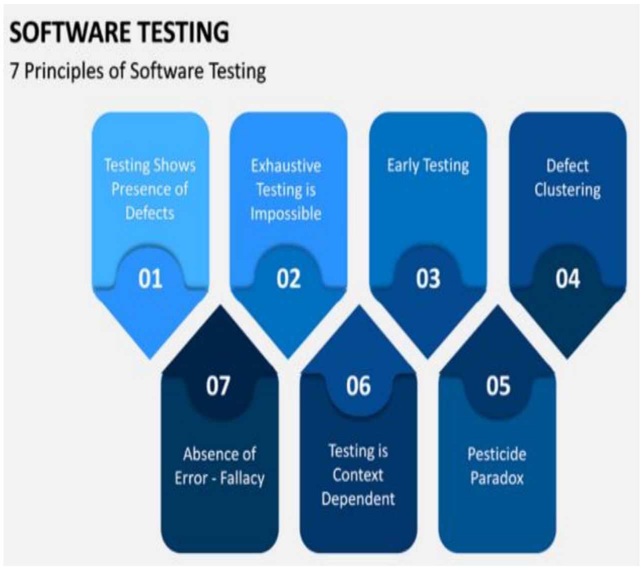
SOFTWARE TESTING
7 Principles of Software Testing
Testing Shows Presence of Defects
01
Exhaustive Testing is Impossible
02
Early Testing
03
Defect Clustering
04
07
Absence of Error - Fallacy
06
Testing is Context Dependent
05
Pesticide Paradox

Figure 8: Principles of Software Testing [17]

- *Principle 3: "Early Testing"*

The third principle states that the cost and efforts to fix a bug increases multi-fold with delay in capturing a bug. Therefore, an error committed by human being, or a fault must be captured as soon as it invades into the system. First, it is 100 times costlier to rectify a defect (fault) after deployment than during the requirement analysis or design phase. Second, 40 to 50 percent of the testing effort in the project is due to rework Therefore, Boehm stated that "timeline of capturing software defects highly influences the total costs of software projects" [16].

Jones stated that "*Defects penetrated during the requirements phase as the hardest to locate and fix. While the defects introduced during coding are the most numerous, they are the easiest to locate and fix. The defects introduced during the design phase are the gravest. Documentation defects can be severe, if ignored*" [15]. So, poorly designed test cases are often more of a trouble than help. "*It is commonly believed that the earlier a defect is detected, the cheaper it is to fix it*".

"The cost of fixing the defect depending on the stage it was found". Earlier a defect is detected, least costly it is to fix the bug. Later a defect is detected, costlier it is to fix the bug. For example, "if a problem in the software requirements is found only after post-release, then it would cost 10–100 times more to fix than if it had already been found during the requirements review phase".

- *Principle 4: "Defect Clustering"*

The fourth principle for software testing states that the defects or faults are not evenly scattered across various modules. Some modules in the software are suspected to have higher density of bugs or defects in comparison to other modules. This rule confirms the results of Pareto rule for software testing which states that almost 80% of the software bugs (or failures) originate from 20% software modules. This is known as 80:20 rule for software testing.

- *Principle 5: "Testing is Context Dependent"*

This principle for software testing states that different kind of testing strategies are required for different types of software applications. There is no golden technique that can work efficiently for all types of software. Therefore, the testers must carefully choose a testing plan and strategy depending upon the types of software under test.

- *Principle 6: "Pesticide Paradox"*

Every method you use to prevent or find bugs "leaves a residue of subtler bugs against which those methods are ineffectual." This principle states that the test cases must be updated, refined and modified for improving software test process. The test cases that are repeated without changes fails to capture new bugs. Hence, continuous improvements in test case must be carried out regularly.

- *Principle 7: "Absence of Errors Fallacy"*

Software testing is not only about finding and fixing bugs. A software testing process must ensure that it meets the user requirements as closely as possible. In case, a software has not been designed and coded as per user requirements, finding and fixing bug won't help [13][14][18][19][20]. Software testing must assure that the software program/product meets the commercial and technical requirements and fulfils its intended purpose.

4. CONCLUSION

The development of software is not only confined to just writing software code. Basically, software development is a long process having number of phases and each phase is very crucial in itself. Software testing is inherently iterative, expensive, time consuming, utmost complex and most significant activity of software development. These characteristics make testing an open research as in-effective and in-efficient testing may lead to production of poor quality and un-reliable software. An un-reliable software requires high maintenance cost and un-necessary repetitions of efforts. This paper explores the software development efforts and principles to better understand the conceptual knowledge of software development and testing.

REFERENCES

1. R. Fairley**,** ***Software Engineering Concepts***. McGraw-Hill Publication, 1st Edition, 1985
2. IEEE Standard Glossary of Software Engineering Terminology , IEEE Press, 1991
3. F. L. Baur "***Software Engineering***". Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 1969.
4. I. Sommerville, ***Software Engineering***. Addison Wesley Publications, 1995
5. B. Beizer. ***Software Testing Techniques***. Thomson Computer Press, 2nd Edition, 1990
6. S. Desikan. ***Software Testing Principles & Practices.*** Pearson Education Publication, 2007
7. A. Mathur. ***Foundation of Software Testing***. Pearson Education Publication, 1st Edition, 2008
8. K. K. Aggarwal, and Y. Singh. ***Software Engineering.*** New Age International Publishers, 2nd Edition.
9. C. Kaner, J. Bach, and B. Pettichord. "***Lessons Learned in Software Testing***". John Wiley & Sons, 2008
10. R. S. Pressman. ***Software Engineering: A Practitioner's Approach.*** McGrawHill Publication, 6th Edition, 2005
11. M.E. Khan. **Different Approaches to White Box Testing Technique for Finding Errors**. *International Journal of Software Engineering and Its Applications*, vol. 5, no. 3, pp. 1-14, 2011
12. A. Malishevsky, G, Rothermel, and S. Elbaum. **Modelling the Cost Benefits Trade-offs for Regression Testing Techniques**. *IEEE International Conference on Software Maintenance,* pp. 204-213, 2002
13. H. Mcminn . **The State Problem for Evolutionary Testing**. *Genetic and Evolutionary Comuting*, pp. 2488-2498, 2003. https://doi.org/10.1007/3-540-45110-2_152
14. R. Mall. **Model Based Testing of Object Oriented Software's.** *CSI Communications*, vol. 31, pp. 16-18, 2008.
15. C. Jones. **Software Engineering Best Practices**. McGraw-Hill Publication, 2009.
16. B. W. Boehm. **Software Engineering-As It Is** *IEEE International Conference on Software Engineering*, pp. 11-21, 1979
17. https://www.sketchbubble.com/en/presentation-material-req-planning.html
18. K. Solanki, Y. Singh and S. Dalal, **A Comparative Evaluation of "m-ACO" Technique for Test Suite Prioritization**", *Indian Journal of science and technology*, Vol. 9, No. 30, pp.1-10, Aug. 2016. https://doi.org/10.17485/ijst/2016/v9i30/86423
19. K. Solanki, Y. Singh, and S. Dalal, "**Test case prioritization: an approach based on modified ant colony optimization (m-ACO)",** *International Conference on Computer, Communication and Control (IC4), IEEE*, pp. 1-6, Sept. 2015. https://doi.org/10.1109/IC4.2015.7375627
20. O. Dahiya and K. Solanki, **A systematic literature study of regression test case prioritization approaches**. *International Journal of Engineering & Technology*, Vol. 7, No. 4, pp.2184-2191, 2018. https://doi.org/10.14419/ijet.v7i4.15805
21. K. Solanki, S. Dalal and V. Bharti. **Software Engineering Education and Research in India: A survey**, *International Journal of Engineering Studies*, vol. 1, no. 3, pp 181-192, 2009.
22. O. Dahiya and K. Solanki and S. Dalal, **Comparative Analysis of regression test case prioritization approaches**. *International Journal of Advanced Trends in Computer Science Engineering*, Vol. 8, No. 4, pp 1532-1536, 2019.
23. R.B Jadhav, S. D. Joshi, U. G. Thorat, and A. S. Joshi, **Software defect learning and analysis utilizing regression method for quality software development**. *International Journal of Advanced Trends in Computer Science Engineering*, Vol. 8, No. 4, pp 1275-1282, 2019. https://doi.org/10.30534/ijatcse/2019/38842019
24. L Rajamanickam, N. A. Maatsat and S.N. BintiDaud, **Software Testing: The generation tools** *International Journal of Advanced Trends in Computer Science Engineering*, Vol. 8, No. 3, pp 231-234, 2019. https://doi.org/10.30534/ijatcse/2019/20822019