Volume No: 1(2014), Issue No: 5 (May) ISSN No: 2348-4845

UNDERSTANDING THE TERMINOLOGY AND TEST PLAN – A SURVEY ANALYSIS

Uday kumar J, Asst.prof, Jagadeeswara Rao G, Asst.prof. AITAM.

Quality is proportional to testing in software engineering. Quality is not only the justification of requirements but also the presence of values. In this new era of software engineering , test phase is ought to be viewed as separate phase in software development lifecycle. At the same time we can't separate these two words "FIXING and FINDING". Tester will find the defect later developer will fix it.

Inspection of code by a programmer itself turned as testing. So, how can we find a thin layer between Fixing and Finding? This paper attempted to address the understanding of terminology and difference among those roles and phases along with good test plan.

1. INTRODUCTION

As software industry's nature is "Do all most anything", it leads to one thing that makes complex. Therefore good software is a big challenge; thereby we can get it by good test process. Software testing is indispensable in ensuring Software quality [54].

Testing is a process in which the defects are identified, isolated , subjected for rectification and ensure that the product is defect free in order to produce quality product in the end .Hence customers satisfaction. In spite of this recognition, software testing activities have been slow to move from the end of the development life cycle and into all phases of the development process.

Here lot of approaches are there in test process like Inspect code, analyze cause for errors. Inspecting the detail design and code is much better way to find errors than testing. Walkthroughs can catch sixty percentage of errors. Generally, Walkthroughs and other form of human inspection are good at catching surface problems and style issues. Few humans are good at reviewing even first order semantic issues in a code segment. It is far more cost effective to reduce the affect of an error by preventing it than it is to find and fix it.[53]

In 1945, a moth became trapped between the points of a relay in the Mark II Aiken Relay Calculator, causing the computer to perform incorrectly; this was the first computer bug[2].

Today, new development models (e.g., progressive, iterative, and agile methods) have emerged and been proposed to tackle some of the criticisms of the SDLC methodology [36].

In 1985,the Therac-25 radiation therapy device malfunctioned and delivered lethal radiation doses. This was due to a software bug. In 1988, more than 2,000 computers were infected by the first computer worm, which was made possible because of a buffer overflow bug in the Berkeley Unix finger daemon program[30].

2. DEFECT VERSUS BUG

These days we people facing lot of failures in software because of several reasons. Testers felt that their job is to find defects whereas programmers felt that they have to build bug free application. Here a conflict is there between tester and developer and also defect vs. bug.

Defect is defined as deviation from the requirement. At the same time defect is a notation used by test engineer. Whereas Bug is a defect which is accepted by

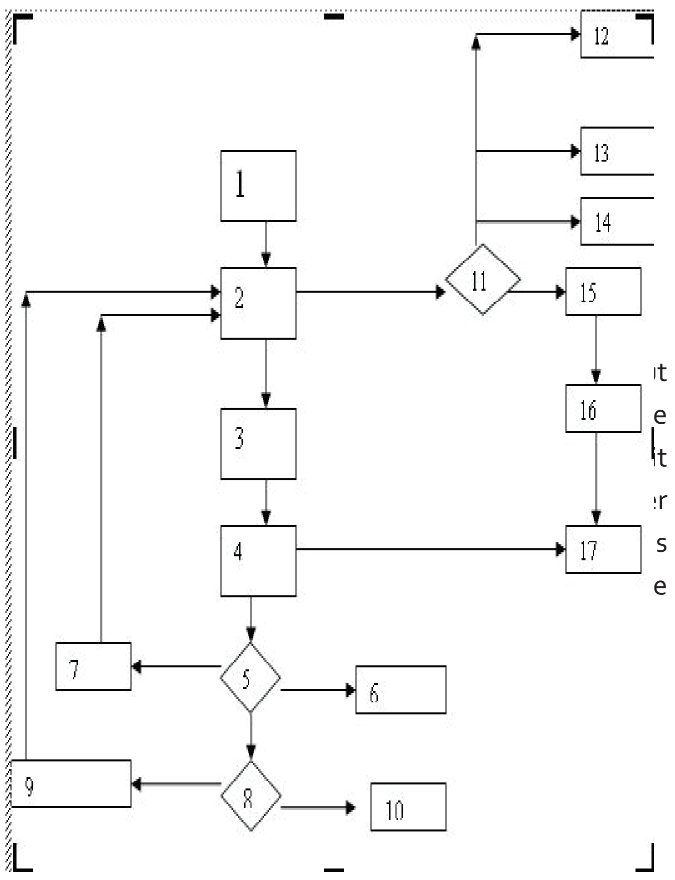

fig1: Bug life cycle

1.Requirements 2.development 3.application 4.testing team 5.if defects 6.stop 7.new 8.if defect really rectified 9.reopen 10.closed 11.if it really a defect 12.hold

13.reject 14.as per design 15.open 16.rectification

17.fixed

Status :

New: Whenever the defect is newly identified for first time then the test engineer set the status as new.

Open: Whenever the developer accepts the defect then set the status as Open.

Fixed: Whenever the developer rectified the defect then he will set the status as Fixed.

Differed: Whenever the developer accept the defect but want to rectify it later then he will set the status as Differed.

Closed and Re-open: Whenever the next build is released the test engineer will check whether the defect is really rectified or not. If at all they feel really rectified then they will set the status as Closed. Otherwise Re-Open.

Hold: Whenever the developer is confused to accept or refused then it will be Hold. Whenever the defect is in hold status they will have one meeting on Defect. It gives solution.

Rejected: Whenever the developer feels it is not at all defect then he will set the status as rejected.

As per Design : Whenever the developer feels, the test engineer raised defect without knowing about latest request then he will set the status as per Design.

4.PRIORITY AND SEVERITY OF DEFECTS

Software bugs can cause a range of problems, ranging from minor glitches to loss of life or significant material loss[30]. The use of incorrect intermediate results due to undetected bugs has been known to lead to catastrophes in mission critical or even safety critical situations[6][7].Conventional testing is a sort of testing in which testing engineers will check the developed applications and its related parts are working on base of requirements or not. Conventional testing starts from coding phase to last phase comes under quality control.

Unconventional testing starts from initial phase and continues for last phase comes under quality assurance. Quality assurance people check each and every person is doing his or her work according to company's process guidelines or not.

Priority describes the sequence in which defects need to be rectified. Priority classified into four types

P1. Critical

P2. High

P3. Medium

P4: Low

Severity describes the seriousness of the defect. Severity classified into four types.

S1: Fatal

S2: Major

S3: Minor

S4: Suggestion

Testing is a technology, as well as art. In some situations we need to handle least severe with high priority and high severe bug with least priority , always challenging. Generally we use a process of debugging to find and remove the bugs from the program. Debugging is a frequent, tedious, and time-consuming task for software developers.[31,32]

5. STLC:

Royce [31] is widely recognized for introducing the first formal methodology for software development, now known as the waterfall methodology. Royce's waterfall model introduced a sequential process that emphasized systematic development and divided software development processes into separate and distinct phases, including requirements analysis, program design, coding, testing, and operations.

Software Testing Life Cycle contains Six phases.

1. Test planning
2. Test Development
3. Test Execution
4. Result analysis
5. Bug Tracking
6. Reporting

1. Test planning

Plan: Plan is strategic document which contains some information that describes how to perform a task in a effective, efficient, optimized way.

Optimization: It is a process of utilizing the available resources to the level best and getting maximum possible output.

Test Plan: It is a strategic document which contains some information that describes how to perform testing on application in an effective, efficient, optimized way.

Test Lead will prepare test plan document.

Contents of Test plan:

1.0 Introduction

1.1. Objective:

Purpose of this document will be clearly described here in this section

1.2. Reference document:
The list of all the document referred by the test lead while preparing test plan document will be listed out in this section. For example SRS, Project plan .

2.0 .Coverage of testing:

2.1. Features to be tested
List all the features that are to be tested which are within the scope will be listed out here in this section

2.2. Features not to be tested:
List all the features that are not planned for testing will be listed out here in this section. It may contain out of scope features , features that are planned to be incorporated in feature, low risk features, features that are skipped based on time constraints.

3.0.Test Strategy:
It is a organization level term that is common for all projects in the organization
Level of testing, types of testing , test design techniques, configuration management ,test metrics , terminology, automation plan, list of automated tools are important aspects in test strategy.

4.0.Basic criteria: It deals with acceptance and suspension criteria .Acceptance criteria tells us when to stop testing on that application, suspension criteria tells us when to suspend testing on that application.

5.0.Test deliverables and Test Environment: The list of all documents that are prepared and delivered, details of environment that is about to be used for testing during the test process will be listed out.

6.0.Resource Planning and Scheduling: Who has to do what? Is clearly mentioned in this section. The starting dates and ending dates of every task also mentioned here.

7.0.Risk and Assumptions :The list of all potential risks and corresponding solution plans will mentioned here .For example Employee may leave the organization in the middle of the project. Contingency is bench strength. One more example is if tester enable to test all the features within the time then Contingency is priority based execution The list of all assumptions that are to be made by test engineers will be listed out here.

8.0.Approval Information: Who has approved the document and when it is approved will clearly mentioned in this section.

6. TYPES OF TESTING:

The traditional debugging process includes code review, using debugging tools such as GDB[33].

Sanity testing is a type of testing in which one will conduct overall testing on the released build. For example whether required connections like JDBC,ODBC are properly established or not.

Whether one can navigate to all the pages of application or not .Some people call it as smoke test.

Regression Testing Perform testing on already tested functionality again and again is called regression testing. Test the new features for first time is not regression testing .It means Regression testing starts from second build. Random testing is also comes under Regression testing.

Perform testing on same functionality again and again with multiple set of values to achieve conclusion. Retesting starts from first build and ends to last build.

Installation Testing is to test that one should install the application into environment by following the guidelines provided in the deployment document in order to get suitable installed application. .To do testing in this situation , whatever the ideas we get are known as test cases. Test cases are broadly categorized into three types 1.GUI test cases 2.Functional test cases 3. Non functional test cases also called as performance test cases. Functional test cases are divided into two types 1.positive test case 2.negative test cases .Some guidelines for writing positive test cases are test engineer should have positive mind set and at the same time he must consider positive flow of application .Test engineer must use only valid inputs from point of functionality. .Some guidelines for writing negative test cases are test engineer should have negative mind set at the same time he must consider positive flow of application .Test engineer must use only invalid inputs from point of functionality. Port testing is a testing in which one will install the application into original customer environment and check whether it is compatible with it or not. In End-to-End testing one will perform testing on all the end-to-end scenarios of the application. That comes under transaction flow testing. Authentication testing is a basic testing in all web applications, it checks different combinations of user name and password in order to conform whether authorized persons are using application or not. Direct URL testing and firewall leakage testing are known testing types belongs to security testing. Ad-hoc testing is the best testing to the tester as per his psychology.

7. BUG TRACKING AND REPORTING:

This process should need because, access must be firm to authorized people and report must contain these issues like number of cycles of execution, number of test cases executed in this cycle, number of defects found, duration of each cycle etc. The situation where we feel that we should do some testing is known as test scenario. To do testing in this situation , whatever the ideas we get are known as test cases .
Traceability matrix is a document which contains table of linking information used for tracing back .

Here we can get some nice aspects like defect age, latent defect and the goodness of test suit and test bed This general model is often refined and each of these phases made more or less granular, by breaking them into additional or fewer phases [18].Test suite is a combination of different types of test cases also called as test sweet. Test bed is a combination of test environment plus test suite. The good thing in bug tracking and reporting is finding to important issues in this phase. Those are defect age and latent defect.

The time gap or duration between opening date and closing date of defect is called as defect age. The defect that is found late after some religious is known as latent defect .Even inspections and reviews are not good enough to find them. Therefore in this paper I am stressing importance of test plan along with bug tracking and reporting.

8. PSYCHOLOGY OF TESTING

One of the primary causes of poor program testing is the fact that most programmers begin with a false definition of the term. They might say:

"Testing is the process of demonstrating that errors are not

present." Hence, don't test a system to show that it lives up to expectations; rather, you might as well begin with the presumption that the system holds mistakes (a quality presumption for very nearly any system) and after that test the project to discover whatever number of the mistakes as could reasonably be expected.

9. FUTURE WORK

As we have already mentioned the role of test plan is very crucial and due to evaluatory conditions still test plan can be planned in optimized manner .So, one of our further thought is involving the analysis of latent defect and defect age we can produce best plan by that way we can increase the success rate of software .Our sense is a good test plan can decrease failure rate of software.

10. CONCLUSION

If at all most of the bugs are detected at early stages that can help companies to improvise quality in software. But metrics is always a complex word when we are discussing about software quality .Benchmarks are needed at the same time finding benchmarks are tough .Our attempt is to get fine understanding of terms and content in test plans. Latent defect shows its multifaceted behavior .Accomplished test case means which test case can finds the latent defect. Sense of prediction should cropped between test cases and defects. Those test cases should decrease the defect age. Even with utmost attention if a test engineer write a test case , it may not help to find defect. Here negative flow would essential. This paragraph pointing the need of good test plan.

11. REFERENCES

[1] Salfner, F., Lenk, M., and Malek, M. 2010. A survey of

online failure prediction methods. ACM Computer. Survey.

42, 3, Article 10 March2010, 42 pages.

[2] J. S. Huggins, "First Computer Bug," 2006,

http /www.jamesshuggins.com/h/tek1/_rst computer bug.htm.

[3] Shinde V. "Bug life cycle" .Sept 2007.

http://www.softwaretestinghelp.com/buglifecycle/

[4] Zimmermann T, Nagappan N., and Zeller A. "Predicting bugs from his-tory". Springer, 2008.

[5] Figure 1.1 "Bug lie cycle" The bugzilla guide- 2.18.6 release chapter 6. using bugzilla

http://www.bugzilla.org/docs/2.18/html/lifecycle.html

[6] European Space Agency. Arianne?5 ight 501 Inquiry Board Report. http://ravel.esrin.esa.it/docs/esa?x? 1819eng.pdf

[7] N. Leveson and C.S.Turner. An Investigation of the Therac?25 Accidents IEEE Computer, Vol. 25, No. 7, July 1993.

[8] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object- riented design metrics as quality indicators," IEEE Trans. Software Eng., vol. 22, no. 10, pp. 751761, 1996.

[9] N. Ohlsson and H. Alberg, "Predicting fault-prone software modules in telephone switches," IEEE Trans. Software Eng., vol. 22, no. 12, pp. 886-894, 1996.

[10] L. C. Briand, J. W. Daly, and J. Wust, "A unified framework for coupling measurement in object-oriented systems," IEEE Trans. Software Eng., vol. 25, no. 1, pp. 91-121, 1999.

[11] K. E. Emam, W. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," Journal of Systems and Software, vol. 56, no. 1, pp. 63-75, 2001.

[12] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," IEEE Trans. Software Eng., vol. 29, no. 4, pp. 297-310, 2003.

[13] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object- oriented metrics on open source software for fault prediction," IEEE Trans. Software Eng., vol. 31, no. 10, pp. 897-910, 2005.

[14] N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect density," in Proceedings of ICSE 2005. ACM, 2005, pp. 580-586

[15] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in Proceedings of ICSE 2006. ACM, 2006, pp. 452-461.

[16] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in Proceedings of ICSE 2005. ACM, 2005, pp. 284-292.

[17] A. E. Hassan, "Predicting faults using the complexity of code changes," in Proceedings of ICSE 2009, 2009, pp. 78-88.

[18] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in Proceedings of ICSE 2008, 2008, pp. 181-190.

[19] A. Bernstein, J. Ekanayake, and M. Pinzger, "Improving defect prediction using temporal features and non linear models," in Proceedings of IWPSE 2007, 2007, pp. 11-18. [20] S. Kim, T. Zimmermann, J. Whitehead, and A. Zeller, "Predicting faults from cached history," in Proceedings of ICSE 2007. IEEE CS, 2007, pp. 489-498.

[21] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," IEEE Trans. Software Eng., vol. 31, no. 4, pp. 340-355, 2005.11

[22] A. E. Hassan and R. C. Holt, "The top ten list: Dynamic fault prediction," in Proceedings of ICSM 2005, 2005, pp. 263-272.

[23] R. Robbes, D. Pollet and N. Lanza , "Replaying IDE Interactions to Evaluate and Improve Change Prediction Approaches" ,in proceedings of MSR 2010, IEEE , 2010, pp.161-180.

[24] T. Zimmermann, P. Weigerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in ICSE. IEEE Computer Society, 2004, pp. 563-572.

[25] A. Ying, G. Murphy, R. Ng, and M. Chu-Carroll, "Predicting source code changes by mining change history," Transactions on Software Engineering, vol. 30, no. 9, pp. 573-586, 2004.

[26] J. Sayyad-Shirabad, T. Lethbridge, and S. Matwin, "Mining the maintenance history of a legacy software system," in Proceedings of ICSM 2003, 2003, pp. 95-104.

[27] L. C. Briand, J. Wust, and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems," in Proceedings of ICSM 1999, 1999, pp. 475-482.

[28] F. G. Wilkie and B. A. Kitchenham, "Coupling measures and change ripples in c++ application software," Journal of Systems and Software, vol. 52, no. 2-3, pp. 157-164, 2000.

[29] J. Ferzund, S. Nadeem Ahsan, and F. Wotawa , "Software Change Classification using Hunk Metrics" in Proceedings of ICSM 2009,IEEE, 2009.

[30] S. Gar_nkel, "History's Worst Software Bugs," 2005, http://wired.com/news/technology/bugs/0,2924,69355,00.htm.

[31] "Software bug," 2006, http //enw_ ikipediao_ rg/wiki/Computer bug.

[32] A. Zeller, Why Programs Fail: Elsevier, 2006.

[33] Free Software Foundation Inc., "GDB: The GNU Project Debugger," vol. 2006, 2006, http://www.gnu.org/software/gdb/.

[34] M. Ernst, J. Cockrell, W. Griswold and D. Notkin, "Dynamically discovering likely program invariants to support program evolution", IEEE TSE, Vol.27, No. 2, Feb 2001.

[35] M. Ernst, A. Czeisler, W. Griswold and D. Notkin, "Quickly detecting relevant program invariants", ICSE, 2000.

[36] S. Hangal and M. Lam, "Tracking down software bugs using automatic anomaly detection", ICSE, 2002. 12

[37] P. Zhou, W. Liu, L. Fei, S. Lu, F. Qin, Y. Zhou, S.Midki_ and J. Torrellas, "AccMon: Automatically detecting memoryrelated bugs via program counter-based invariants", MICRO- 37, 2004.

[38] A. Barr, "Find the Bug", Addison-Wesley, 2004.

[39] M. Dimitrov and H. Zhou , "Anomaly-based bug prediction, isolation, and validation: an automated approach for software debugging" , The 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XIV), 2009.

[40] S. Shivaji, E. James Whitehead, Jr., R. Akella and S. Kim, "Reducing features to improve bug prediction" , International conference on automated software engineering., IEEE, 2009.

[41] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object oriented design metrics as quality indicators," IEEE Trans. Software Eng., vol. 22, no. 10, pp. 751-761, 1996.

[42] T. Zimmermann, R. Premraj, and A. Zeller. "Predicting defects for eclipse". in Proceedings of PROMISE 2007. IEEE CS, 2007, p. 76.

[43] S kim. "Adaptive bug prediction by analyzing project history". Computer science thesis, University of California, santa cruz , 2006.

[44] M.stoerzer, B.G. Rayder and F. Tip. "Finding Failure-Inducing Changes in Java Programs using change classi_cation". ACM, 2006.

[45] B. Bergman and B. Klefsjo, Kvalitet, _dn behov till anvdndning, Stu-dentlitteratur, Lund, 1991.

[46] James Kloeppel,"Researchers Develop Intelligent Bug Detection Software", 2006, http://www.hpcwire.com/hpcwire/20060714/researchers develop intelligent bug detection software-1.html

[47] P. L. Li, J. Herbsleb, M. Shaw, and B. Robinson, "Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc," in Proceedings of the 28th ICSE, Shanghai, China: ACM, 2006.

[48] M. Lyu, Handbook of Software Reliability Engineering. McGraw-Hill, 1996.

[49] V. R. Basili and B. T. Perricone, "Software Errors and Complexity: An Empirical Investigation." Communications of the ACM, vol. 27, pp. 42-52., 1984.

[50] G. Denaro and M. Pezze, "An Empirical Evaluation of Fault-Proneness Models," in Proceedings of the 25th International Conference on Software Engineering (ICSE2002), Miami, USA, May 2002., 2002.

[51] A. Mockus, D. Weiss, and P. Zhang, "Understanding and Predicting Effort in Software Projects," in Proceedings of the 26th ICSE, IEEE 2003.

[52] Benediktsson, O., Dalcher, D., and Thorbergsson, H. "Comparison of Software Development Life Cycles: A Multiproject Experiment," *IEE Proceedings Software*, Volume 153, Number 3, 2006, pp. 87-101.

[53] Royce, W.W. "Managing the Development of Large Software Systems," *Proceedings of the IEEE WESCON*, Los Angeles, California, August 1970, pp. 1-9.

[54] Cohen, C.F., Birkin, S.J., Garfield, M.J., and Webb, H.W. "Management conflict in software testing," *Communications of the ACM*, Volume 47, Number 1, 2004, pp. 76-81.

[55] Boehm, B. "Software and Its Impact: A Quantitative Assessment," *Datamation*, Volume 19, Number 5, 1973, pp. 48-59.

[56] Boehm, B. "A View of 20th and 21st Century Software Engineering," *Proceedings of the 28th International Conference on Software Engineering*, May 20-28, 2006, Shanghai, China, pp. 11-29.

[57] Boehm, B. and Basili, V.R. "Software Defect Reduction Top 10 List," *Computer*, Volume 34, Number 1, 2001, pp. 135-137.