Technique of Finding the Defect in Software Testing

VIJAY PRATAP KATIYAR

Computer Science Department, Faculty of Engineering and Technology, Rama University, Kanpur, Uttar Pradesh, India

---

ABSTRACT:- Software testing is a process of executing a program or application with the intent of finding the software bugs. It can also be stated as the process of validating and verifying that a software program or application or product: Meets the business and technical requirements that guided it's design and development.

Software testing is the most significant stage of the Software Development Life Cycle. Projects underneath testing goes through different stages such as test analysis, test planning, test case, test case review process, test execution process, requirement traceability matrix (RTM), defect tracking bug logging and tracking), test execution report and closure.

In terms of software, defects means whenever expected results not meet actual results. Generally defect is known as a bug. lt talks about the complete life cycle of a bug right from the stage it was found, fixed, re-test, and close.

This paper basically deals with entire process of bug life cycle and how to avoid the bug. To avoid the bug, Test Engineer should prepare the bug report template which consists of various steps. Complete the bug free task.

Keywords

Test Plan, Bug Life Cycle, Classification of Defect, Black-box Technique.

1. INTRODUCTION

Software testing is completed with different stages such as test analysis, test planning, test case, test case review process, test execution process, requirement traceability matrix (RTM), defect tracking (bug logging and tracking), test execution report and closure. Software testing is the major activity of estimating and accomplishing product with an observation to find out faults. It is the procedure where the system constraints and system modules are implemented and estimated manually or by using automation tools to figure out whether the system is fulfilling the specific requisites or not. [1]

The term ”bug” refers to software defects, and the term has been a publicly accepted term for software defect since first referred to as such in Harvard Mark I days [2].

Debuggers are crucial tools needed in the development process, actually during the development of the average software project at least half of the time is spent debugging. [2][3]. In spite of that, the debuggers most commonly used to debug software aren’t taught to students to any great extent. Debuggers are in general studied very little, for example compared to compilers. Most debuggers offer more sophisticated functions such as running a program ”step-by-step”, pause application execution to examine its current state, and tracking the value changes of variables A debugger is a Complex piece of software that can be is used to test and “debug” other applications. Their internals require sophisticated algorithms and data structures to be able to perform their objective. Debuggers are used to analyze and find out why software doesn’t behave as expected. They help developers to understand the software and to find the cause of a software glitch. The developer can with the help of the debugger find and repair the glitch to allow the software to work according to its original intent. The debugger can control the software being debugged so it can allow the developers to follow the execution flow of the software, and in that way verify that the software executes as expected.

A Test plan is a document describing the scope, approach, objectives, resources, and schedule of a software testing effort. It identifies the items to be tested [4, 5], items not be tested, who will do the testing, the test approach followed, what will be the pass/fail criteria. It is also a document we share with the Business Analysts, Project Managers, Development teams. This is to enhance the level of transparency into the QA team’s [5] working to the external teams. It is documented by the QA Manager/QA Lead based on the inputs from the QA team members. Test plan is not static and is updated on an on demand basis.

2. BRIEF LITERATURE SURVEY

2.1 Test Plan

Test plan is a dynamic document that derives entire testing activities. It is prepared in the beginning as soon as the requirement is gathered. Following are the attributes of test plan.

A. Objective: It shows that, what is the aim of writing the test plan. This test plan is written to test the functionality of product.

B. Scope: It consists of two sub part:

(i) In scope i.e., features which needs to be tested.

(ii) Out of scope i.e., features which need not to be tested.

C. Test Methodology: It defines the types of testing which needs to be performed by test engineer for start particular release. Testing types are shown as below:

(i) Start with smoke testing (test basic and critical features of an application.)

(ii) Functional testing

(iii) Integration Testing

(iv) System Testing

(v) Regression Testing

(vi) Compatibility Testing

D. Approach, Test Environment, Templates, Roles and Responsibilities, Effort estimation, Entry and exit criteria, Schedule, Automation, Defect tracking, Assumption, Deliverables, Risks, Mitigation/Backup/Contingency plan.

E. Test Plan Identifier ,Test Items, Software Risk Issues ,Features to be Tested ,Features not to be Tested ,Approach ,Item Pass/Fail Criteria , Suspension Criteria and Resumption Requirements , Test Deliverables ,Remaining Test Tasks ,Staffing and Training Needs ,Responsibilities , Schedule ,Approvals ,Glossary[6]

Table 1

TEST CASE TEMPLATE	
HEADER	
Test Case Name Id	
Test Case Type	Functional/Integration/Systems
Requirement	
Module	
Status	
Severity	Critical/Major/Minor
Release	
Version	
Pre-Condition	
Test Data	
Brief Description	
BODY	
Steps	
Inputs	
Description	
Expected Result	
Actual Result	
Status	

Comments	
FOOTER	
Author	
Date	
Approved By	
Reviewed BY	

[9][10]

2.2 Bug Life Cycle

Bug life cycle is a complete life cycle of a bug. Whenever a bug is fixed, we retest the bug and change the status accordingly. It might be closed or re-open. We get bugs because of

- **Wrong code** i.e., logic of the program is wrong or functionality does not work according to the requirement.
- **Missing code** i.e., developer has forgotten to develop that particular feature and it is not available in the application.
- **Extra coding** i.e., developer has created a feature which is not available in the requirement but exists in application.

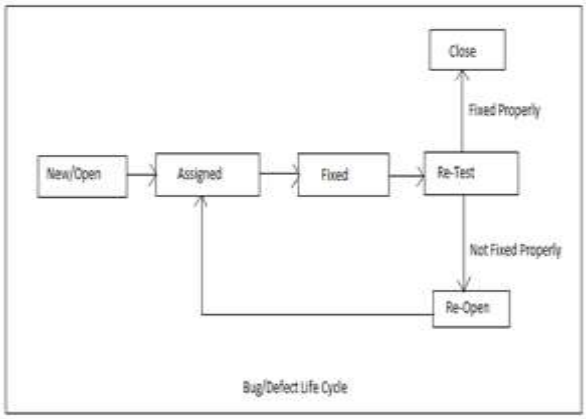

Fig-1

Following are the entire process of bug life cycle:

STEPS:-

- As soon as we get a bug status of bug is "New/Open".
- The bug is reported to the concerned person by changing the status as "Assigned".
- Once developer gets the bug, first he is going to go through the bug i.e., check if it is valid or not then if it is valid bug start re-producing the bug i.e., perform the actions on the application according to steps mentioned in the bug report and then change the code.
- Once the code changes are done, the developer changes the status to "fixed".
- The test Engineer starts re-testing the bug and if it is fixed properly then changes the status to "closed". Else if bug is still exists then status is change to "reopen" and assign it back once again to the developer.

This process continues until the bug is "fixed" or "closed". [7][8]

2.3 Classification of Defect

A defect in software testing is an error in programming or logic that causes a program to failure or to produce wrong/unexpected outcome.

Following are the different types of defects/bug:

A. Invalid or rejected bug

Whenever scenario is wrong and developer does not accept it as a bug then it is called invalid/rejected bug.

This bug generally occurred due to the following cases:

Case 1:- if Test Engineer misunderstood the requirement and may log the bug then developer will change the status to invalid. Then again Test Engineer go through it and if he also feel that it's invalid and closes the bug by keeping the status as close**.**

Case 2:- if developer misunderstood the requirement and changes the status to invalid after go through. The Test Engineer will go through the bug and he also may reproduce it and if the bug exists really he changes the status from invalid to re-open.

B. Duplicate bug

When same bug is found by different Test Engineer then it is known as duplicate bug. This bug is occurred because of following reasons:

- Common features (modules) which is access by both the Test Engineer.
- Dependent features.

SOLUTION FOR AVOID DUPLICATE BUG

i. Search in bug repository, if bug already exists, do not log (report) a bug. If bug does not exist, log (report) a bug and store in bug repository.

ii. Send it to developer and carbon copy to all test engineers.

C. Not-Reproducible bug

Developer accepts the bug but not able to find the same bug after following the navigation steps mentioned in the bug report.

Reasons for not-reproducible bug:

- **Platform mismatch**

i. Server mismatch

Test Engineer tests the application in one server and developer may reproduce the bug in another server, to avoid this mention server name in bug report.

ii. Environment mismatch

Test Engineer tests the application in different operating system and browser and developer may reproduce the bug in different operating system and Browser, to avoid this mention platform name in bug report.

- **Data mismatch**

Scenario may be correct, but for testing an application, Test Engineer use different data and developer may use different data for reproducing the bug, to avoid this mention test data in bug report.

- **Build mismatch**

Test Engineer got a bug in one build and developer reproduce the same bug in different/another build due to time constraint it is known as build mismatch.

D. Can't fix bug

Developer accept the bug, also able to reproduce it but not able to perform code changes due to some reasons. These reasons are as follows that's why developer can't fix the bug:

1. Core of code (bug is in the core of code)

2. No technology support

If there will be major changes in bug then developer can't say that can't fix that bug. Can't fix bug should be minor bug but all minor bug cannot be can't fix bug.

E. In-consistent bug

In first time Test Engineer found a bug but after that he is not able to find the same bug in next time, so to avoid the inconsistency Test Engineer should take the screenshot and follow the following steps:

1. As soon as Test Engineer got the bug first to take the final screenshot of that bug.

2. Re-confirm the bug whether it is consistent or not.

3. If bug is consistent then search the bug in bug repository. If bug report is not found then prepare the bug report and send it to the developer.

F. Deferred/Postpone bug

Even though there is a bug, it is postpone to the future releases due to time constraint. Due to time constraint developer going to make it to deferred i.e. he will fix it in next release. In the initial builds, bugs can't be deferred but in the later stages due to the time limit he can deferred the bug. But the test engineer will check that bug can be really deferred or not. Deferred bug can be minor bug but the entire minor bug cannot be deferred. We cannot close the bug until it is fixed in next release.

G. Requests for Enhancement

These are the suggestion given by the test engineer towards the enhancement of the application.

Table 2

BUG REPORT TEMPLATE	
1. Date: 2. Reporter : 3. Assigned to : 4. Status : 5. Severity : 6. Priority : 7. Server : 8. Platform : 9. Test Data : 10. Build # : 12. Brief Description : 13.Steps to reproduce : 14. Observation : 15.Expected result 16. Attachments	

[9][10]

3. METHODOLOGY

Black-Box Design Technique

Equivalence partitioning - Equivalence partitioning or **equivalence class partitioning** (**ECP**) is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed. An advantage of this approach is reduction in the time required for testing software due to lesser number of test cases.

Equivalence partitioning is typically applied to the inputs of a tested component, but may be applied to the outputs in rare cases. The equivalence partitions are usually derived from the requirements specification for input attributes that influence the processing of the test object**.** [11]

Boundary Value Analysis- Boundary value analysis (BVA) is based on testing at the boundaries between partitions.

Here we have both valid boundaries (in the valid partitions) and invalid boundaries (in the invalid partitions).

As an example, consider a printer that has an input option of the number of copies to be made, from 1 to 99. To apply boundary value analysis, we will take the minimum and maximum (boundary) values from the valid partition (1 and 99 in this case) together with the first or last value respectively in each of the invalid partitions adjacent to the valid partition (0 and 100 in this case). In this example we would have three equivalence partitioning tests (one from each of the three partitions) and four boundary value tests. Consider the bank system described in the previous section in equivalence partitioning.

Because the boundary values are defined as those values on the edge of a partition, we have identified the following boundary values: -$0.01 (an invalid boundary value because it is at the edge of an invalid partition), $0.00, $100.00, $100.01, $999.99 and $1000.00, all valid boundary values. So by applying boundary value analysis we will have six tests for boundary values. [11]

Decision Tables - Decision tables are a concise visual representation for specifying which actions to perform depending on given conditions. They are algorithms whose output is a set of actions. The information expressed in decision tables could also be represented as decision trees or in a programming language as a series of if-then-else and switch-case statements.

Each decision corresponds to a variable, relation or predicate whose possible values are listed among the condition alternatives. Each action is a procedure or operation to perform, and the entries specify whether (or in what order) the action is to be performed for the set of condition alternatives the entry corresponds to.

To make them more concise, many decision tables include in their condition alternatives a don't care symbol. This can be a hyphen [12] or blank although using a blank is discouraged as it may merely indicate that the decision table has not been finished. One of the uses of decision tables is to reveal conditions under which certain input factors are irrelevant on the actions to be taken, allowing these input tests to be skipped and thereby streamlining decision-making procedures**.** [13],

Aside from the basic four quadrant structure, decision tables vary widely in the way the condition alternatives and action entries are don't care symbol represented.[14] Some decision tables use simple true/false values to represent the alternatives to a condition (similar to if-then-else), other demonstration tables may use numbered alternatives (similar to switch-case), and some tables even use fuzzy logic or probabilistic representations for condition alternatives.[16] In a similar way, action entries can simply represent whether an action is to be performed (check the actions to perform), or in more advanced decision tables, the sequencing of actions to perform (number the actions to perform).

A decision table is considered balanced [13] or complete [18] if it includes every possible combination of input variables. In other words, balanced decision tables prescribe an action in every situation where the input variables are provided.[13]

State transition table - In automata theory and sequential logic, a state transition table is a table showing what state (or states in the case of a nondeterministic finite automaton) a finite semi automaton or finite state machine will move to, based on the current state and other inputs. A *state table* is essentially a truth table in which some of the inputs are the current state, and the outputs include the next state, along with other outputs. [23]

State Transition testing, a black box testing technique, in which outputs are triggered by changes to the input conditions or changes to 'state' of the system. In other words, tests are designed to execute valid and invalid state transitions

When to use?-

- When we have sequence of events that occur and associated conditions that apply to those events
- When the proper handling of a particular event depends on the events and conditions that have occurred in the past
- It is used for real time systems with various states and transitions involved

Example: A System's transition is represented as shown in the below diagram:

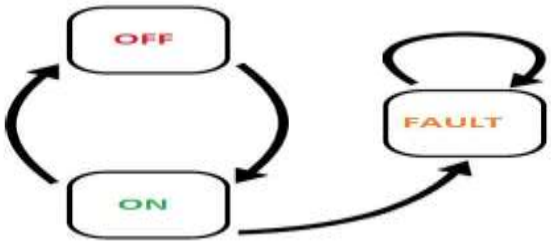

Fig-2

The tests are derived from the above state and transition and below are the possible scenarios that need to be tested.

Table 3

Tests	Tests 1	Tests 2	Tests 3
Start State	Off	On	On
Input	Switch ON	Switch Off	Switch Off
Output	Light ON	Light Off	Fault
Finish State	ON	OFF	On

[15]

4. RESULT

With the help of Testing we can reduce the bug from the software but it cannot prove that there is no remaining bug or software is bug free. The main reasons for bug rejections are improper bug reports and inadequate information of test engineer over the developed project. To avoid such kind of bug we should prepare a proper bug report. To start the testing process firstly test engineer should have a clear knowledge of requirement. After that he can clearly identify the bug. He should prepare the test document and test plan as per the requirement. Based on these test plan further test engineers can prepare the test case where all the positive as well as negative scenarios are included. And prepare test case, test scenario, bug report template in such a way that any other person can easily reproduce the same scenario or identify the bug without wasting time over reviews and rework. Development team as well as testing team should be a part of requirement gathering team for enhanced understanding of software requirements. It will help the project team to have a proper proposal of scope and restrictions to avoid rework.

5. CONCLUSIONS

Requirement Analysis forms the backbone of software. Software is bug free with the help of software testing processes like as testing design techniques, software test plan, test case execution, test strategies etc.

If we prepare the proper bug report then the entire bug should be resolved by developer easily.

Development team also understands the requirement and they resolved the bugs and send the report to the Tester. By this process client get the bug free software.

Today, testing is the most challenging and dominating activity used by industry, therefore, improvement in its effectiveness, both with respect to the time and resources, is taken as a major factor by many researchers The purpose of testing can be quality assurance, verification, and validation or reliability estimation.

6. REFERENCES

[1] Itti Hooda, et al. "Software Test Process, Testing Types and Techniques", International Journal of Computer Applications (0975 –8887) February 2015, Volume 111 – No 13.

[2] Jonathan B. Rosenberg. How Debuggers Work. John Wiley & Sons, London, 1996

[3] Quantify the time and cost saved using reversible debuggers. Survey done on Cambridge Judge Business School, 2012.

[4] Roger S. Pressman " Software Engg. A Practitioner's Approach ", 6th Edition.

[5] Si Huang, Myra Cohen, and Atif M. Memon(2010) "Repairing GUI Test Suites Using a Genetic Algorithm, "In Proceedings of the 3rd IEEE International Conference on Software Testing Verification and Validation(ICST).

[6] Cirstian Cadar et al," Symbolic Execution for Software Testing in PracticePreliminary Assessment ", ICSE, May, 2011.

[7] Aranda, J., & Venolia, G., et al. "The Secret Life of Bugs: Going Past the Error and Omissions in Software Repositories", 31st IEEE International Conference on software Engineering (ICSE), 2009, pp298-308.

[8] Qin, F., Tucek, J., & Zhou, Y., et al. "Treating Bugs As Allergies: A Safe Method for Surviving Software Failures", Proceedings of the10th conference on Hot Topics in Operating Systems (HOTOS), 2005, Vol. 10, pp.19-19.

[9] Breu, S., Premraj, R., Sillito, J., & Zimmermann, T., et al. "Information needs in bug reports: improving cooperation between developers and users", Proceedings of the 2010 ACM conference on Computer supported cooperative work(CSCW), 2010, pp301-310.

[10] Giovanni Denaro, Bernhard Scholz, Zhi Quan Zhou, et al. "Automated Software Testing and Analysis: Techniques, Practices and Tools", 2014 47th Hawaii International Conference on System Sciences, p. 260, 2007, doi:10.1109/HICSS.2007.96.

[11] Burnstein, Ilene (2003), Practical Software Testing, Springer-Verlag, p. 623, ISBN 0-387-95131-8

[12] Ross, Ronald G. (2005). "Decision Tables, Part 2 ~ The Route to Completeness". Business Rules Journal. 6 (8). Retrieved 11 November 2017.

[13] Snow, Paul (19 July 2012). "Decision Tables". DTRules: A Java Based Decision Table Rules Engine. Retrieved 11 November 2017.

[14] Rogers, William T. "Decision Table Examples: Medical Insurance". Saint Xavier University Systems Analysis and Design. Archived from the original on March 29, 2007.

[15]https://www.tutorialspoint.com/software_testing_dictionary/state_transition.htm

[16]Breen, Michael (2005), "Experience of using a lightweight formal specification method for a commercial embedded system product line" (PDF), Requirements Engineering Journal, 10 (2), doi:10.1007/s00766-004-0209-1