ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

ICITCSA 2017

Pioneer College of Arts and Science, Coimbatore

Vol. 6, Special Issue 1, January 2017

Quality Assurance for Business Needs using Software Testing Concepts

R. Jaya Kumar[1], A. Senthil Kumar[2]

Assistant Professor, Department of Computer Science, Sankara College of Arts and Science, Coimbatore, Tamilnadu.[1,2]

Abstract: Software testing is a process of verifying and validating that a software application or program 1. Meets the business and technical requirements that guided its design and development, and 2. Works as expected. Software testing also identifies important defects, flaws, or errors in the application code that must be fixed. During test planning we decide what an important defect is by reviewing the requirements and design documents. An important defect is one that from the customer's perspective affects the usability or functionality of the application. Assuring quality is not a responsibility of the testing team. The testing team cannot improve quality; they can only measure it, although it can be argued that doing things like designing tests before coding begins will improve quality because the coders can then use that information while thinking about their designs and during coding and debugging.

Keywords: Software, Quality.

I. INTRODUCTION

Software testing is not a one person job. It takes a team, but the team may be larger or smaller depending on the size and complexity of the application being tested. The programmer who wrote the application should have a reduced role in the testing if possible. The concern here is that they're already so intimately involved with the product and "know" that it works that they may not be able to take an unbiased look at the results of their labors.

A good developer does not necessarily make a good tester and vice versa, but testers and developers do share at least one major trait, they itch to get their hands on the keyboard. As laudable as this may be, being in a hurry to start can cause important design work to be glossed over and so special, subtle situations might be missed that would otherwise be identified in planning. Like code reviews, test design reviews are a good sanity check and well worth the time and effort.

II. THE V-MODEL OF SOFTWARE TESTING

Software testing is too important to leave to the end of the project, and the V-Model of testing incorporates testing into the entire software development life cycle. In a diagram of the V-Model, the V proceeds down and then up, from left to right depicting the basic sequence of development and testing activities. The model highlights the existence of different levels of testing and depicts the way each relates to a different development phase

a) Like any model, the V-Model has detractors and arguably has deficiencies and alternatives but it clearly illustrates that testing can and should start at the very beginning of the project.

b) The business requirements are also used to guide the user acceptance testing. The model illustrates how each subsequent phase should verify and validate work done in the previous phase, and how work done during development is used to guide the individual testing phases.

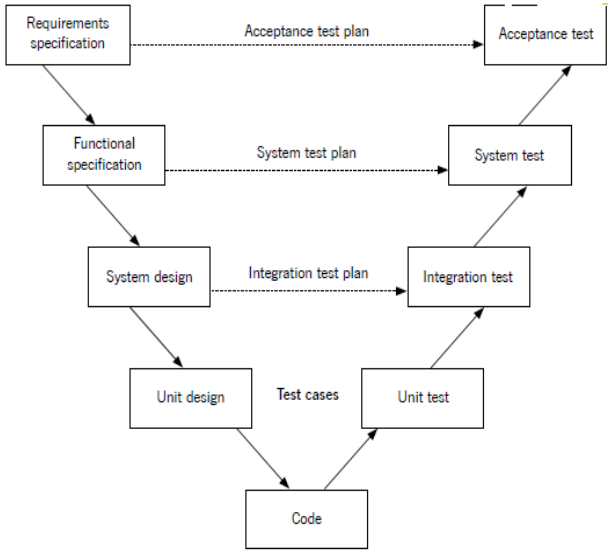

Fig 1.1 Model Of Sofware Testing

c) V-model means Verification and Validation model. Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development. Requirements like BRS and SRS begin the life cycle model just like the waterfall model. But, in this model before development is started, a system test plan is created. The test plan focuses on meeting the functionality specified in the requirements gathering.

The high-level design (HLD) phase focuses on system architecture and design. It provides overview of solution,

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

ICITCSA 2017

Pioneer College of Arts and Science, Coimbatore

Vol. 6, Special Issue 1, January 2017

platform, system, product and service/process. An integration test plan is created in this phase as well in order to test the pieces of the software systems ability to work together.

The low-level design (LLD) phase is where the actual software components are designed. It defines the actual logic for each and every component of the system. Class diagram with all the methods and relation between classes comes under LLD. Component tests are created in this phase as well. The implementation phase is, again, where all coding takes place.

III. THE SYSTEMATIC APPROACH

System Testing tests all components and modules that are new, changed, affected by a change, or needed to form the complete application. The system test may require involvement of other systems but this should be minimized as much as possible to reduce the risk of externally-induced problems. Testing the interaction with other parts of the complete system comes in Integration Testing. The emphasis in system testing is validating and verifying the functional design specification and seeing how all the modules work together. The first system test is often a smoke test. This is an informal quick-and-dirty run through of the application's major functions without bothering with details. The term comes from the hardware testing practice of turning on a new piece of equipment for the first time and considering it a success if it doesn't start smoking or burst into flame.

IV. TESTIMONY METRIC

In the software testing literature, people often talk about white-box testing and black box testing. Black-box testing treats the program under test as a "black box." No knowledge about the implementation is assumed. In white box testing, the tester has access to the details of the program under test and performs the testing according to such details. Therefore, specification-based criteria and interface-based criteria belong to black-box testing. Program based criteria and combined specification and program based criteria belong to white-box testing. Another classification of test adequacy criteria is by the underlying testing approach. There are three basic approaches to software testing:

(1) Structural testing: specifies testing requirements in terms of the coverage of a particular set of elements in the structure of the program or the specification;

(2) Fault-based testing: focuses on detecting faults (i.e., defects) in the software. An adequacy criterion of this approach is some measurement of the fault detecting ability of test sets.

(3) Error-based testing: requires test cases to check the program on certain error-prone points according to our knowledge about how programs typically depart from their specifications. The source of information used in the adequacy measurement and the underlying approach to testing can be considered as two dimensions of the space of software test adequacy criteria.

V. BOUNDARY VALUE TESTING

Consider some mapping (function) that has an int input variable with the interval of values $a \leq x \leq b$, where the boundary values for x are a and b. One basic boundary value analysis approach is to select test values for an input variable, such as x above, as follows: a, $a + \varrho$, nominal, $b - \varrho$, and b, where "nominal" represents some "middle" or typical value within x's range, and ϱ denotes some small deviation.. Generalizing the approach to deal with more than one variable can be straightforward. Consider, as an example, a mapping that involves two input variables with the following ranges:

$$a \leq x \leq b$$
$$c \leq y \leq d.$$

A generalization of the boundary value analysis approach to handling this example is easy if we assume that failures are seldom the result of simultaneous faults in the input variables.

VI. CONCLUSION

Software testing provides a means to reduce errors, cut maintenance and overall software costs. Numerous software development and testing methodologies, tools, and techniques have emerged over the last few decades promising to enhance software quality. While it can be argued that there has been some improvement it is apparent that many of the techniques and tools are isolated to a specific lifecycle phase or functional area. One of the major problems within software testing area is how to get a suitable set of cases to test a software system. This set should assure maximum effectiveness with the least possible number of test cases. There are now numerous testing techniques available for generating test cases

REFERENCES

[1] G. Bernet, L. Bouaziz, and P. LeGall, "A Theory of Probabilistic Functional Testing," Proceedings of the 1997 International Conference on Software Engineering, 1997, pp. 216 –226

[2] B. Beizer, "Software Testing Techniques," Second Edition, Van Nostrand Reinhold Company Limited, 1990, ISBN 0-442-20672-0

[3] J.B. Good Enough and S. L. Gerhart, "Toward a Theory of Test Data Selection," IEEE Transactions on Software Engineering, June 1975, pp. 156-173

[4] E. Engström and P. Runeson. A qualitative survey of regression testing practices. In Proceedings of the International Conference on Product-Focused Software Process Improvement, pages 3–16, 2010.

[5] K. Adamopoulos, M. Harman, and R. M. Hierons. How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution. In GECCO (2), Volume 3103 of Lecture Notes In Computer Scienc, pages 1338–1349. Springer, 2004

[6] D. Amalfitano, A. R. for Android mobile application testing. In Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops, pages 252–261, 2011.

[7] Fasolino, and P. Tramontana. A GUI crawling-based technique

[8] A. Arcuri and X. Yao. Coevolving programs and unit tests from their specification. In In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, pages 397–400, 2007.