IJCSC 0973-7391 Vol- 6 • Issue-1 Sep - Mar 2015 pp.148-154 available at www.csjournalss.com

Software Quality Assurance through 'Testing': A Review

Vikash Yadav

Research Scholar, Suresh GyanVihar University, Jaipur

Bright Keswani, Associate Professor, Dept. of Comp. Application, Suresh GyanVihar University, Jaipur

Abstract: High quality software cannot be done without high quality testing. Software testing is an ultimate obstacle to the final release of software products.Software testing is also a primary cost factor in the overall construction of software products.The development and testing of software-based systems is an essential activity for the automotive industry. Software-based systems with different complexities and developed by various suppliers are installed in today's premium vehicles, communicating with each other via different bus systems. The integration and testing of systems of this kind of complexity is an extremely difficult task. Software quality is directly related to software testing as better tests will result in error free software which ultimately results in better software quality. This paper is an attempt to justify that one can assure easily Quality of Software through 'Testing' process easily.

Keywords: Software Quality Assurance,Functional Testing, Software Testing,Structural Testing, Verification and Validation etc.

1. Introduction

Software testing is one of the major activities in development. To make the testing more effective many attempts have been made. Methodologies like extreme programming have emphasised software quality and as the complexity of many software projects grows, software development processes are forced into more testing and quality assurance. On the one hand, model-based testing techniques are new testing methods aimed at increasing the reliability of software products, and decreasing the cost by automatically generating a test suite from a formal behavioural model of a system. On the other hand, the architectural specification of a system represents a gross structural and behavioural aspect of a system at the high level of abstraction. Formal architectural specifications of a system also have shown promises to detect faults during software back-end development. The prime objective of testing is to detect faults in the systems under test and to convey confidence in the correct functioning of the systems if no faults are found during complete testing. Faults not found in the different testing phases could have major consequences that range from customer dissatisfaction to damage of physical property or, in safety relevant areas, even to the endangering of human lives. Therefore, the thorough testing of developed systems is essential. Evolutionary Testing tries to improve the effectiveness and efficiency of the testing process by transforming testing objectives into search problems, and applying evolutionary computation in order to solve those [16]. *"Too little testing is a crime – too much testing is a sin".* The risk of under testing is directly translated into system defects present in the production environment. The risk of over testing is the unnecessary use of valuable resources in testing systems that have no or very few defects.

Apart from introduction, the paper is well divided into various sections for better understanding. Section 2 contains basics about software testing. Section 3 contains the details about various

software testing approaches. Section 4 contains the importance of software testing in various program scenarios and in section 5 variousaspects of software quality are discussed.

2. Fundamentals of Software Testing

The basic purpose of the software testing is to detect errors that may be present in the program. So the concentration in the testing should not start with the intent of showing that a program works perfectly but the intent should be on the negative side i.e. to show that a program does not work perfectly. Primary cause of poor program testing is the fact that most programmers begin with the false definition of the term. They might say "Testing is the process of demonstrating that errors are not present in the program" or "The purpose of testing is to show that a program performs its intended function correctly" or "Testing is the process of establishing confidence that a program does what it is supposed to do". These definitions are upside-down. A More Appropriate Definition for software testing is that "Testing is the process of executing a program with the intent of finding errors" [11]. There are two fundamental strategic issues that software test designs must accommodate: one is the problem of defining when a test case has shown an accurate outcome or has shown a fault. This is known as the oracle problem. The other is the problem that it is seldom practical to test the complete range of possible inputs and outputs for any given real world software application. The standard approach to this test scope coverage problem is to use some techniques to narrow the range of test case inputs and outputs to a representative and manageable number. The challenging task of software testing is making use of limited testing resources for selecting test cases that effectively detect failures.

3. Approaches of Testing

The practice of testing software has become one of the most important aspects of the process of software creation.When software is tested the first and potentially most crucial step is to design test cases. Developing effective and efficient testing techniques has been a major problem when creating test cases. There are several well-known techniques associated with creating test cases for a system. Test design strategies are chosen that are appropriate to the type of application under test and the types of bugs sought. Each strategy has a distinct scope, assumptions and limitations. Basically there are two approach of software testing namely Black-Box Testing or Functional Testing and White-Box Testing or Structural Testing.

3.1 Functional Testing

Functional testing is a method of software testing that tests the functionality of an application as opposed to its internal structures. This testing strategy is based on the view that any program can be considered to be a function that maps values from its input domain to the values in its output range. Many times, human being operates very effectively with black box knowledge; in fact, this is central to object orientation. This method of testing can be applied to all levels of software testing: unit, integration, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

In this kind of testing the test cases are designed on the basis of the clients need or the specifications of the program rather than the internal structure of the program. The most understandable functional testing approach is exhaustive testing but it is not practical.

Functional test cases have two distinct advantages:

1) They are independent of the fact that how the software is implemented. So if the implementation is changes, the test cases remain unaffected and are still useful.

2) Test case generation can be started in parallel with the implementation, hence saving the time of overall project development.

Functional testing also has a major disadvantage of redundant test cases. Significant redundancies may exist among test cases which is responsible for wastage of effort and time.

3.2 Structural Testing

The functional testing is concerned with the function that the program under test is supposed to perform and does not deal with the internal structure of the program responsible for actually implementing that function. The structural testing is concerned with the functionality of the software under test rather than the actual implementation of the program. Structural testing, on the other hand is concerned with testing the actual implementation of the program. The intent of this testing is not to exercise all the different input or output conditions but to exercise the different programming structures and data structures used in the program.

4. Importance of Testing

Extensive testing can only be carried out by an automation of the test process claimed [19]. The benefits are reduction in time, effort, labor and cost for software testing. Automated testing tools consist in general of an *instrumentator*, *test harness* and a *test data generator*.

Static analyzing tools analyze the software under test without executing the code, either manually or automatically. It is a limited analysis technique for programs containing array references, pointer variables and other dynamic constructs. Experiments show that this kind of evaluation of code inspections (visual inspections) are very effective in finding 30% to 70% of the logic design and coding errors in a typical software, [4]. *Symbolic execution* and *evaluation* is a typical static tool for generating test data.

Many automated test data generators are based on symbolic execution, [7], [15]. Symbolic execution provides a functional representation of the path in a program and assigns symbolic names for the input values and evaluates a path by interpreting the statements and predicates on the path in terms of these symbolic names, [9]. Symbolic execution requires the systematic derivation of these expressions which require more computational effort. The values of all variables are maintained as algebraic expressions in terms of symbolic names. The value of each program variable is determined at every node of a flow graph as a symbolic formula (expression) for which the only unknown is the program input value. The symbolic expression for a variable carries enough information that, if numerical values are assigned to the inputs, a numerical value can be obtained for the variable, this is called symbolic evaluation. The characteristics of symbolic execution are:

a. Symbolic expressions are generated and show the necessary requirements to execute a certain path or branch, [2]. The result of symbolic execution is a set of equality and inequality constraints on the input variables; these constraints may be linear or non-linear and define a subset of the input space that will lead to the execution of the path chosen.
b. If the symbolic expression can be solved, then the test path is feasible. And the solution corresponds to a set of input data which will execute the test path. If no solution can be found then the test path is infeasible.

c. Manipulating algebraic expressions is computationally expensive, especially when performed on a large number of paths.
d. Common problems are variable dependent loop conditions, input variable dependent array (sometimes the value is only known during run time) reference subscripts, module calls and pointers, [10].
e. These problems slow down the successful application of symbolic execution, especially if many constraints have to be combined, [3] and [6].

Some program errors are easily identified by examining the symbolic output of a program if the program is supposed to compute a mathematical formula. In this kind of event, the output has just to be checked against the formula to see if they match.

In contrast to *static analysis*, *dynamic testing* tools involve the execution of the software under test and rely upon the feedback of the software (achieved by instrumentations) in order to generate test data. Precautions are taken to ensure that these additional instructions have no affect whatever upon the logic of the original software. A representative of this method is described by [6] who used instrumentation to return information to the test data generation system about the state of various variables, path predicates and test coverage. A penalty function evaluates how good the current test data is with regard to the branch predicate, by means of a constraint value of the branch predicate. There are three types of test data generators; *path wise*, *data specification* and *random test data generator*.

Random testing is the simplest technique of test data generation**.** It could be used to generate data for any type of program, since every data is a string of bits. But random testing mostly does not perform well in terms of coverage, since it merely relies on probability. It has quite low chances in finding semantically small faults [12], and thus accomplish high coverage. A fault that is only revealed by small percentage of program input is called semantically small fault. For example, in following code:

```
void function1(int x, int y)
{
        if (x ==y)
                        print("ONE");                  // statement 1
        else
                        print("ZERO");                 // statement 2
}
```

The probability of executing *statement 1* is 1/n, where n is the maximum integer, since to execute *statement 1*, both x and y must be same. So random testing can generate this type of test data with very less probability.

The distribution of selected input data should have the same probability distribution of inputs which will occur in actual use (operational profile or distribution which occurs during the real use of the software) in order to estimate the operational reliability [5] [13, [18].

"To err is human; to find the error quickly & correct it is divine" [17]. During any phase of s/w development, the chances of errors getting introduced are in plenty. Thus the need arises for verification of the products of S/W development. So

(a) S/W Testing is a process of executing a program with the intent of finding errors [11].

(b) A good test case is one that has a high probability of finding an as yet undiscovered error;

(c) A successful test is one that uncovers an as yet undiscovered error; &

(d) Testing is the process to prove that the S/W works correctly [14].

Testing is done because programmers are human, and human is to err, this is a true fact in the domain of software and software controlled systems. Errors tend to propagate; a requirement error may be amplified during design and amplified still more during coding process. A fault is the result of an error. It is more precise to say that a fault is the representation of an error, where representation is the mode of expression, such as narrative text, dataflow diagram, hierarchy charts and source code. A failure occurs when a fault executes. An incident is the symptom associated with a failure that alerts the user to the occurrence of the failure. A test is the act of exercising software with test cases. Test case occupies a central position in testing [19].

Random testing selects test data randomly from the input domain and then test the program with these test cases. The automatic production of random test data, drawn from a uniform distribution, should be the default method by which other systems should be judged, [8].

5. Software Quality Aspects

Software Quality Assurance (SQA) consists of a means of monitoring thesoftware engineering processes and methods used to ensure quality. It does thisby means of audits of the quality management system under which the softwaresystem is created. These audits are backed by one or more standards, usuallyISO 9000.

It is distinct from software quality control which includes reviewing requirementsdocuments, and software testing. SQA encompasses the entire softwaredevelopment process, which includes processes such as software design, coding,source code control, code reviews, change management, configuration management,and release management. Whereas software quality control is a control ofproducts, software quality assurance is a control of processes.

Software quality assurance is related to the practice of quality assurance inproduct manufacturing. There are, however, some notable differences betweensoftware and a manufactured product. These differences stem from the factthat the manufactured product is physical and can be seen whereas the softwareproduct is not visible. Therefore its function, benefit and costs are not as easilymeasured. What's more, when a manufactured product rolls off the assemblyline, it is essentially a complete, finished product, whereas software is never finished.

Software lives, grows, evolves, and metamorphoses, unlike its tangiblecounterparts. Therefore, the processes and methods to manage, monitor, andmeasure its on-going quality are as fluid and sometimes elusive as are the defectsthat they are meant to keep in check. [1]

SQA is also responsible for gathering and presenting software metrics.For example the Mean Time Between Failure (MTBF) is a common softwaremetric (or measure) that tracks how often

the system is failing. This SoftwareMetric is relevant for the reliability software characteristic and, by extension theavailability software characteristic.

SQA may gather these metrics from various sources, but note the importantpragmatic point of associating an outcome (or effect) with a cause. In this waySQA can measure the value or consequence of having a given standard process,or procedure. Then, in the form of continuous process improvement, feedbackcan be given to the various process teams (Analysis, Design, Coding etc.) anda process improvement can be initiated.

6. Conclusion

As high quality software is required in every company irrespective of whether it is a product based company or service based company. So developing good quality software is always a dream for developers. Software testing provides an mechanism to test the software for its completeness and other quality attributes. Also SQA activities helps in making the product better. So, the relationship between these concepts and software has been described in this paper. In future, we can explore them to build a model which makes better quality products.

References

1. Brad Clark, Dave Zubrow, "How Good Is the Software: A review of Defect Prediction Techniques", sponsored by the U.S. department of Defense 2001 by Carnegie Mellon University, version 1.0, pg 5.
2. Clarke L. A.: 'A system to generate test data and symbolically execute programs', IEEE Trans. on Software Engineering, Vol. SE-2, No. 3, pp. 215-222, September 1976.
3. Coward, P. D.: 'Symbolic execution systems - a review' Software Engineering Journal, pp. 229 - 239, November 1988.
4. DeMillo R. A., McCracken W. M., Martin R. J. and Passafiume J. F.: 'Software testing and evaluation', 1987.
5. Duran, J. W. and Ntafos S., 'A report on random testing', Proceedings 5th Int. Conf. on Software Engineering held in San Diego C.A., pp. 179-83, March 1981.
6. Gallagher M. J. and Narasimhan V. L.: 'A software system for the generation of test data for ADA programs', Microprocessing and Microprogramming, Vol. 38, pp. 637-644, 1993.
7. Howden W. E.: 'Symbolic testing and the dissect symbolic evaluation system', IEEE Transactions on Software Engineering, Vol. SE-3, No. 4, pp. 266-278, July 1977.
8. Ince, D. C.: "*The automatic generation of test data"*, The Computer Journal, Vol. 30, No. 1, pp. 63-69, 1987.
9. King J. C.: 'Symbolic execution and program testing', Communication of the ACM, Vol. 19, No. 7, pp. 385-394, 1976.
10. Korel B.: 'Automated software test data generation', IEEE Transactions on Software Engineering, Vol. 16, No. 8, pp. 870-879, August 1990.
11. Myers, G.J. (1979): *The Art of Software Testing*, John Wiley & Sons, Inc., New York.
12. Offutt J. and Hayes J., "*A semantic model of program faults*". In International Symposium on Software Testing and Analysis (ISSTA 96), pages 195{200. ACM Press, 1996.

13. Ould, M. A.: 'Testing - a challenge to method and tool developers', Software Engieering Journal, pp. 59-64, March 1991.
14. Prasad K.V.K.K.(2006): S/W Testing Tools with case studies, Dreamtech Press.
15. Ramamoorthy C. V., Ho S. F. and Chen W. T.: 'On the automated generation of program test data', IEEE Transactions on Software Engineering, Vol. 2, No. 4, pp. 293-300, December 1976.
16. Reza, H. Lande, S. (2010): Information Technology: New Generations (ITNG), 2010 Seventh International Conference. Las Vegas ISBN: 978-1-4244-6270-4, INSPEC Accession Number: 11402724, Digital Object Identifier: 10.1109/ITNG.2010.122, Date of Current Version: 01 July 2010pp188 – 193.
17. Shingo Shigeo, Zero Quality Control: Source Inspection & the poka-yoke system, Productivity Press, 1986.
18. Taylor R.: '*An example of large scale random testing*', Proc. 7th annual Pacific North West Software Quality Conference, Portland, OR, pp. 339-48, 1989.
19. Sonia Bhargava, Bright Keswani, "Generic ways to improve SQA by meta-methodology for developing software projects", International Journal of Engineering Research and Applications, Vol. 3, Issue 4, pp 927-932, July 2013.