ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 4, pp: (280-283), Month: October - December 2015, Available at: www.researchpublish.com

Software Testing Based on Assumed Systems Specifications

Osaghae E. O.

Department of Computer Science Federal University, Lokoja, Kogi State, Nigeria

Abstract: **Software testing is used to guarantee the quality of software products however, software testing researchers continue to rely mainly on using the documents from the systems specification of the software product. Over the years, software testing researchers attempts to improve the quality of the software products through testing process. Still, modern software testing techniques rely on having a reasonable knowledge of either the codes, systems designs or the modules/units. In this paper, attempt is made to derive Assumed Systems Specifications from the output results generated by the software product being tested. The proposed Assumed System Specifications, will aid the software tester, to capture the functions of software product, using the output results. The merits of the proposed software testing approach are to deriving Assumed System Specifications from output results generated, and this can be used to recommends areas of improvement in the quality of software.**

Keywords: **Software Testing, Systems Specifications, Software Development Life Cycle, Quality Software.**

I. INTRODUCTION

Software testing is the process of executing software to determine if it works correctly and according to the specified requirements. Software testing is a set of activities conducted with the intent of finding errors, produce highly reliable systems and for achieving high quality software. Testing also ensures that the system is working according to the specification. Without measuring, we cannot be sure of the level of quality in software. The two important goals of software testing are to ensure system being developed is according to the customer requirement and to reveal bugs. More than half of budget of a software project spend on testing even though, it is not guaranteeing the correctness of software. Testing is laborious, expensive and time consuming job, so the choice of testing must be based on the risk to the system. A software failure occurs when a piece of software does not perform as required and expected. Traditionally Software testing techniques can be broadly classified into *black-box testing* and *white box testing*. Black box testing is whereby the aim of testing is to have access to the internal source code itself. Black box testing is also called as *functional testing*; a functional testing technique, test designs test cases based on the information from the specification [5]. White box testing techniques is one of the most important and prevalent software testing techniques, it is typically very effective in validating design, decision, assumptions and finding programming errors and implementation errors in software. There are some advantages of white box testing such as it reveals error in hidden code, its side effects are beneficial and it helps in removing extra lines of code. The main disadvantage of white box testing is that, it is very expensive and some of the codes omitted in the code could be missed out [1], [2], [3], [4], [6], [7], [8], [9] and [11].

Test cases are derived from analysis and design stages of software development processes. However, test case generation from design specifications has the added advantage of allowing test cases to be available early in the software development cycle, thereby making test planning more effective. Model based testing (MBT), as implied by the name itself, is the generation of test cases and evaluation of test results based on design and analysis models. This type of testing is in contrast to the traditional approach that is based solely on analysis of code and requirements specification. In traditional approaches to software testing, there are specific methodologies to select test cases based on the source code of the program to be tested. Test case design from the requirements specification is a black box approach, where as code-based testing is typically referred to as white box testing. MBT, on the other hand is referred to as the gray box testing approach. Software testing is involved in each stage of software life cycle, but the way of testing conducted at each stage of software development, is different in nature and it has different objectives. *Unit testing* is a code based testing which is performed by developers, this testing is mainly done to test each and individual units separately. This unit testing can be done for small units of code or generally no larger than a class. *Functional Testing*: the software program or system under test is observed as a *black box*. The choice of test cases for functional testing is based on the *requirement* or *design specification* of the software entity under test [1], [3] and [9].

Software Testing Metrics are quantifiable measures that could be used to measure different characteristics of a software system or the software development process. Test metrics are known as an important indicator of the effectiveness of a software testing process. However, in practice, software testing and analysis tools are typically not powerful enough to address complications in testing or analysis of real-world software, which is growing increasingly larger and more complex over time [5] and [10]. It is observed that functional testing experts should be provided with design specification documents of the software systems being tested. In some real-life situations, the software tester may not have access to the design specification documents rather, the experts only have to test the software using the output results generated. In this paper, there is an attempt to test software, by firstly, deriving assumed system specification using output results generated by the software product, before other testing activities continues.

II. RELATED WORKS

In this section, related works to this paper will be discussed with the intention revealing the contributions of some authors in software testing, using systems specifications and designs. The related research works are as followed:

[1] Proposed a tool called Spec Explorer used to formally define system requirements and show how test cases can be automatically generated from this model. As a sequential system with several states, and constraints, cruise control system case showed that once requirements are fully collected and correctly defined, a formal model can be very effective in automatically generating test cases to evaluate an application. They claimed that formal models can be used to create the design and possibly find weakness in the design, before reaching the implementation. They asserted that using formal models could be expensive in terms of time and resources [1].

[11] reported various types of testing techniques that can be applied in measuring various quality attributes of software and which of the testing techniques are related to various phase of Software Development Life Cycle (SDLC). They observed that in all Software Development Life Cycle (SDLC) models, testing is applied after a particular stage and not in all the phases. Armed with this idea, they proposed that testing should be applied in all the phases of SDLC and not at a particular stage [11].

[3] observed that output data produced by the execution of the software with a particular test case provides a specification of the actual program behavior. Test case generation in practice is still performed manually most of the time, since automatic test case generation approaches require formal or semi-formal specification, to select test case to detect faults in the code implementation. He further observed that code based testing technique is not an entirely satisfactory approach to generate guarantee acceptably thorough testing of modern software products. He then proposed an alternative approach to generate test cases from requirements and specifications. These test cases are derived from analysis and design stage itself. He claimed that test case generation from design specifications has the added advantage of allowing test cases to be available early in the software development cycle, thereby making test planning more effective [3].

Having reviewed some of the related research work to this paper, we could not see any attempted of existing research works, trying to build an assumed system specification of software being tested. In this paper, there is attempt to test a software and then assume the system specifications, based on the output results generated by the software product.

III. SOFTWARE TESTING APPROACH

A software tester may not have the system specifications documents rather, the software product and output results generated by the software product. Figure 1 shows architecture for deriving system specifications using Output results. The architecture is divided into two tiers, whereby, there software tester has access to the tier 1 and does not have access to tier 2. Tier 1 is made up of Output Results Documents (Result of functions) and Assumed System Specifications (Assu_S_Specs). Tier 2 is made up of Software Functions (Software-functs) and System Specifications (S_Specs). In tier

2, a software developer uses the System Specification S_Spec$_1$, S_Spec$_2$, S_Spec$_3$, …, S_Spec$_{k-1}$, S_Spec$_k$, to code the software system in a particular programming language and k is the number of System specifications. The System Specifications are derived from the properties of the system being developed. The Functions are software constructs in the form of Software Functions: Software-funct$_1$ Software-funct$_1$, Software-funct$_2$, Software-funct$_3$… Software-funct$_{k-1}$,Software-funct$_k$ and k is the number of software functions in the software, after coding. When the output results from the software functions are generated, they are represented as results of functions, in the output results documents (in tier 1). These results of functions are used to derive Assumed System specifications, because the software tester could not have access to the original System Specification documents..

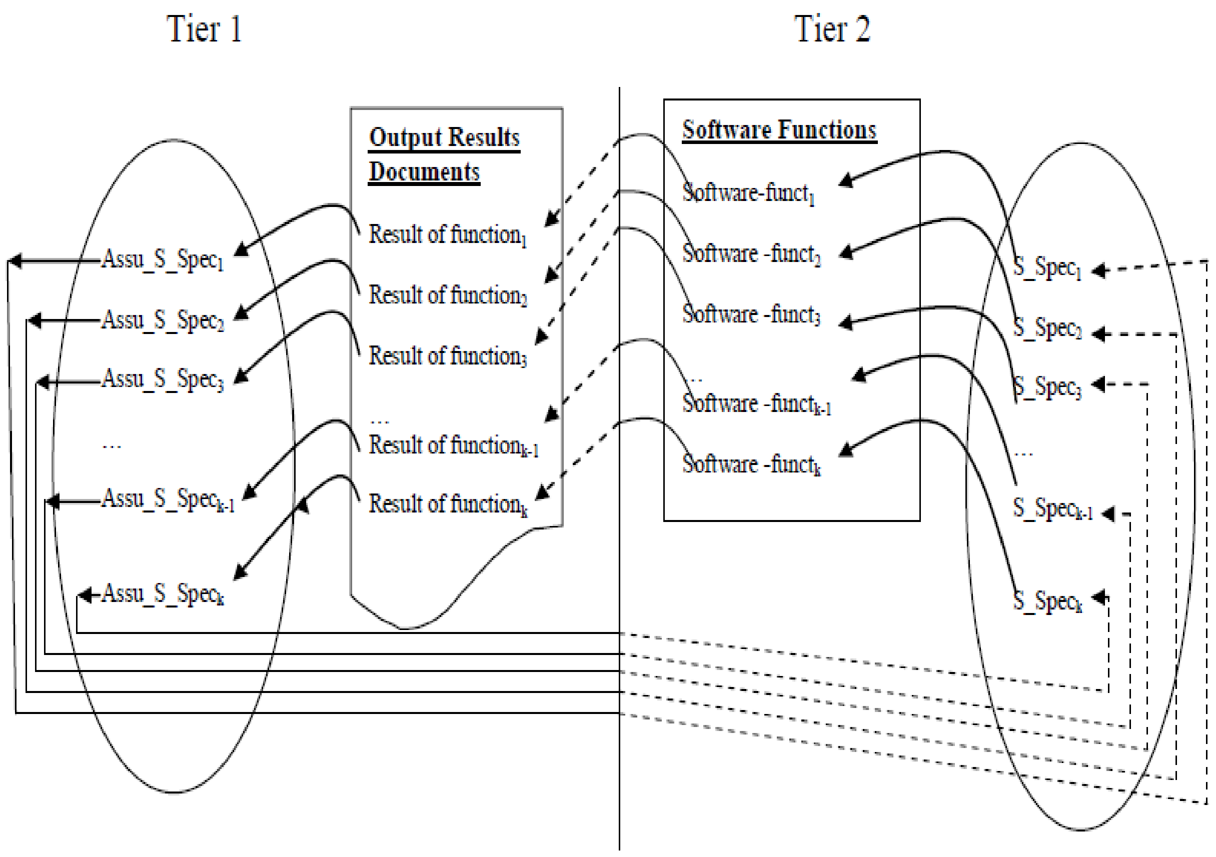

Fig 1: Architecture for Deriving Assumed Systems Specification using Output results

The behaviour of Figure 1 starts from the result of functions displayed in Output Results Documents. These result of functions displayed in the output results documents, to derive Assumed System Specification that is similar to the initial System Specification, used to develop the software product. The Assumed System Specifications cannot act as the initial System Specification, it is a guide to the software tester, to test a faulty software product, whenever the system specifications documents are not available. The changing of directional lines, to dotted lines used by Assumed System Specifications, is an indication that the software tester can only assume the original System Specifications used to develop the software product. In other words, the software tester is can only figured out the System Specification of the software product by assumption, using result of functions displayed from the output result documents. The directional arrows from Assumed System Specifications change when they move from tier 1 to tier 2. Similarly, the software product, displays its results in the output result documents as results of functions and it cannot dictate what the results will be used for. To show that the software functions have no control of how the output results are displayed, hence, their directional lines change to doted lines, when they move from tier 2 to tier 1.

The proposed derivation of Assumed Systems Specification using output results will help the software tester to test the general functionality of a software product. This Assumed System Specifications will help the software tester to correct the faults in the software product and at the same time, it this can also recommend areas of improvements to the software quality.

VI. CONCLUSION

In this paper, I proposed the use of output results to derive Assume System Specifications of a software product. It is observed that before a software product is coded, its system specifications are used to code the software functions. These software functions are what the software product uses to output its results documents, in the form of the results from each software function. The main problem of this software testing approach, is that it could wrongly assume System Specifications, when the software development process, does not adopt the modular software development approach. Another problem that could result from this approach, is an attempt to replace the original System Specifications, with the Assumed System Specifications because, what the software tester has is an assumption of the real specifications. The software testing aided by deriving Assumed System Specifications from the output results documents, only captures what is assumed to the system specification. In other words, Assumed System Specification cannot be a reflection of system properties of the software product.

REFERENCES

[1] S. Al-Emari, and I. M. Alsmadi, “Using for Automatic Checking for Constraints in Software Controlled System”, Informattca Econmica, Vol. 15, No. 3, pp. 5-14, 2011.

[2] M. E. Khan, “Different Approaches to White Box Testing Techniques for Finding Errors” International Journal of Software Engineering and Its Applications, Vol. 5, No. 3, ppA Short Survey, International Journal of Computer Science Issues, Vol. 8, Issues 1, pp. 1-13, 2011.

[3] S. K. Swain, S. K. Pani and D. P. Mohapatpatra, “Model Based Object-Oriented Software Testing”, Journal of Theoretical and Applied Information Technology, Vol. 14, No. 1, pp. 30-36, 2010.

[4] M. E. Khan, “Different Forms of Software Techniques for Finding Error”, International Journal of Computer Science Issues, Vol. 7, No. 3, pp. 11-16.

[5] T. Xie, L. Zhang, L. Zhang, X. Xiao and D. Hao, “Cooperative Software Testing and Analysis: Advances and Challenges”, Journal of Computer Science, and Technology, Vol. 29, No. 4, pp. 713-723, 2014.

[6] ‚A. Misra, R. Mehra, M. Singh, J. Kumar, S. Mishra, “Novel Approach to Automated Test Data Generated for AOP”, International Journal of Information and Technology, Vol. 1, No. 2, 2011.

[7] S. Batra, and R. Rahul, “Improved Quality using Testing Strategies”, Joiurnal of Global Research in Computer Science, Vol. 2, No. 6,International Journal of Advanced Research in Computer Engineering and Technology, Vol. 2, No. 6, pp. 113-117, 2011.

[8] C. Sharma, S. Sabharwal and R. Sibal, “A Survey on Software Testing Techniques using Genetic Algorithm”, International Journal of Computer Science Issues, Vol. 10, No. 1, pp. 381-393, 2013.

[9] S. Nidhra and J. Dondeti, Choudhary, “Black Box and White Box Testing Techniques- A Literature ReviewFeature”, International Journal of Embedded Systems and Applications, Vol. 2, No. 2, pp. 29-50, 2012.

[10] V. Verma and S. Malhotra, “Applications of Software Testing Metrics in Constructing Models of The Software Development Process”, Journal of Global Research in Computer Science, Vol. 2, No. 5, pp. 96-98, 2011.

[11] M. Tuteja and G. Dubey, “A Research Study on importance of Testing and Quality Assurance in Softare Development Life Cycle (SDLC) Models”, International Journal of Soft Computing and Engineering, Vol. 2, Issues, pp. 251-257, 2012.