International Journal of Knowledge, Innovation and Entrepreneurship
Volume 2 No. 1, 2014, pp. 56—68

Testing Techniques Selection Based on SWOT Analysis

MUNAZZA JANNISAR, RUQIA BIBI & MUHAMMAD FAHAD KHAN
University of Engineering and Technology, Pakistan

Received 02 March 2014; received in revised form 15 April 2014; approved 22 April 2014

ABSTRACT Quality is easy to claim but hard to achieve. Testing is a most essential phase in software development life cycle not only to ensure a project's success but also customer satisfaction. This paper presents SWOT Analysis of three testing techniques—White-Box, Black-Box and Grey Box. The testing techniques are evaluated on the basis of their strengths, weaknesses, opportunities and threats. The analysis in this paper shows that selection of the techniques should be based on a set of defined attributes, context and objective. The findings in this paper might be helpful in highlighting and addressing issues related to testing techniques selection based on SWOT analysis.

Keywords: Black Box Testing, White Box Testing, Grey Box, SWOT

Introduction

Software plays a substantial role in every sphere of life. It is a key reason why software engineering continually introducing new methodologies and improvement to software development. The capacity of organisations to achieve customer satisfaction or to correctly identify the needs of customers can be misplaced, leading to ambiguities and unwanted results[1]. Researchers have suggested many testing techniques to deal with fault identification and removal subjects, before the product [software] is shipped to customer. Despite many verification and validation approaches to assuring product quality, defect-free software goal is not very often achieved. For subjective assessment of software testing strategies SWOT analysis can be used to highlights the strengths, weaknesses, opportunities and threats of available techniques that might eventually improve organization's future decisions regarding testing strategies. The main advantage of using SWOT is that it requires little cost and can be applied to address complex situations.

IEEE defines white box testing as a methodology that examines the internal mechanism of a system or module (IEEE, 1990). White box testing provides an opportunity

1.*Problem Identification: The perfect step to successful problem resolution, available: http://webs.anokaramsey.edu/widdel/ftp/b1221/Problem 20Identification.pdf; accessed: 15.4.14.*

to look into the system, concerning itself exclusively to the detailed view of an application's structure. White box testing approach is applicable at unit, integration and system level of testing (Khan and Khan, 2012). Similarly, IEEE (1990) defines black box testing as a testing that takes into account the external mechanism of a system or module. Goal of this testing is to derive test cases that fully exercise behavioral functionality. This methodology checks the system against given inputs and expected outputs.

Grey Box is as testing methodology which embeds the properties of both black-box and white-box testing approaches. The Grey-box technique is from white-box and black-box testing methodology, but embeds the properties of both testing approaches in that it usability and specification correctness from black box testing, leverage it beside the comprehensive code testing of white box[2]. Three testing approaches—White-box testing, Black- box testing and Gray-box testing—are studied and their SWOT analysis is conducted. If a software system is to achieve its full potential, it would be beneficial to bring under consideration the issues regarding each testing technique to gain a deeper understanding of the impact of each testing approach on software development.

The remaining paper is organized in order of the following sub headings: white-box testing approach, testing steps and its SWOT analysis; black-box testing, Gray-box Testing and SWOT of these two approaches; and concluding remarks.

White-Box Testing

This verification technique is also recognized by various names as clear box, glass box, transparent box and structural testing (Beizer, 1990). The connotations applicably specify that White box testing approach implication gives full understanding of the inner mechanism of the system, the logic and the structure of the application's source code. Awareness with the application's internal working scenarios gives the tester familiarity regarding code structure to uncover implementation errors. Exhaustive correctness checking is performed, all code paths are executed to ensure quality software outcome with reduced defect rate. White-box testing catastrophe results in repetition of all black-box testing and reviewing of all white-box testing paths.

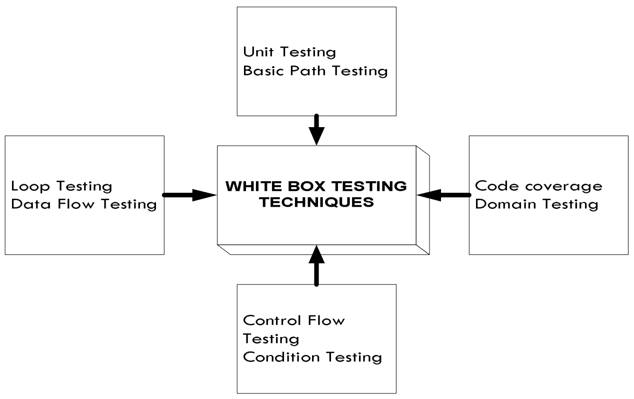

Fig. 1 White Box Testing Techniques

2. Types of software testing, 2014, Available: www.successful-quality-assurance.com; accessed: 15.4.14.

Watch Depth Through Glass Box

Initially test case scenarios are recognized, before writing test cases prioritization of these scenarios is carried out relying on the input fed by the black box testing scenarios. Input from black-box tested scenarios needs more detailed analysis regarding code testing. Second phase contain application block profiling that comprises inspecting source code at runtime. This stage benefits in understanding the resource consumption, time taken by numerous operations and areas of code that could not be accessed. Internal subroutine testing phase ensures that these subroutines can handle all types of data appropriately, that is introduced as an input to the subroutine. Conditional and loops testing phase in code is performed to focus on testing the loops and conditional statements for exactness and efficiency of different data inputs. Conditional testing looks out for the branch statements in the code and aims to test all the paths. Security issues solvation hold major role in any application success. It checks for the possible intrusion and vulnerabilities in the application's whole structure.

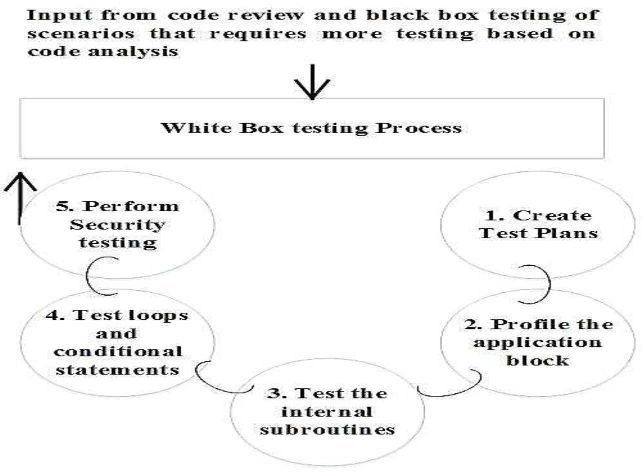
Input from code review and black box testing of scenarios that requires more testing based on code analysis
White Box testing Process
1. Create Test Plans
2. Profile the application block
3. Test the internal subroutines
4. Test loops and conditional statements
5. Perform Security testing

Fig. 2 White Box Testing Steps

Strengths

Full Code Pathway Adept
In white box testing full internal knowledge of system is known, helps in potentially testing all data domains and internal boundaries.

Thoroughness
Use of dig deep phenomenon to find bugs in white box testing is viable to assure that every possible interaction in system has been inspected and tested. The feature of thoroughness helps to inspect code keenly eliminating high probability of error presence, suspecting of correct outcome.

Code Defects
The extra lines of code in a unit or application module might become a cause of non-conformance to functionality. Access provided to source code develops understanding of code behaviour and uncovers unintended output of system components. White box testing reveals such errors by eliminating extra lines of code.

Introspection
Key feature of white box testing technique is to watch out the inner working skeleton, this means that testers can pinpoint the entities programmatically. This is advantageous when the graphical user interface changes frequently or in some scenarios it decreases the fragility of test scripts[1].

Improved Effectiveness
In White box testing, tester performs crosschecking of design decisions contrary to source code to check conformance with the required functionality. This strategy helps in getting a clear perspective of user elicited requirements. The more conformance is, the more it achieves accuracy and the more satisfied customer is.

Experienced team target high quality
White box testing needs experienced developers and testers; this adds to improve quality as high test coverage is ensured by proficient team.

Timely Fault Identification
Evaluating code and making tests those are based on the implementation details, allows testers to remove programming errors before product is delivered to customer.

High Granularity
Test granularity refers to the refinement of a test's focus. A test case that is fine-grained helps the tester dig deep to low-level details, those habitually internal to system.

Weaknesses

Complexity
Clear box testing looks out for detailed knowledge of application and its behaviour. In order to work with it, examining each constituent part of system is essential for tester. This high degree merit requires skilled team of testers to develop test cases.

Maintenance
White box system testing involves test scripts. These test scripts are to be tied to the baseline application's source code. Any change introduced to the code needs high degree of maintenance.

Technical Knowledge
Requires use of specialized tools like code analyzers and debugging tools, this encompasses a detailed technical knowledge of the system.

Staff Training
Moderate or novice testers cannot cope with technical complexity unless a proper training session is conducted; also involves high consumption of cost and schedule.

Practices
Involves familiarity with the organization procedures aimed at achieving less error-prone software. It might be hard for all the programmers to learn these practices as part of their initial education and on-going skills development.

Opportunities

Skill Balancing Act
Since a novice or moderately skilled coders would find it hard to cope with complexity in practices, teaming up with a programmer, subject expert, would ensure a good structural testing. This would help the team to be sufficiently productive, receive training in testing techniques (from the programmer) and be able to detect defects at structural level.

Complexity Measure
Control flow graph ensures complete execution flow of each statement once. Cyclomatic Complexity calculates number of independent paths in the basis set of the program. Complexity measured quantitatively can reduce the complexity level injected in code.

Threats

Fragility
Introspection strategy is designed to look inside the code structure. However, whenever a change is introduced in an application, test scripts needs to be revised. Strong coupling between code and test scripts in white box testing point to a flaw that multiple occurrences of changes in code might eventually break the test cases (Crosschecknet, 2014).

Cultural stress
We see development and tester team as separate teams each with particular work tasks. However, when differentiation begins to blur, that might be a signal perhaps to introduce cultural stress between two teams (RSO, 2014).

Integration
High quality comes with maximum flaws omitted out to have correct specified functionality. White box performs exhaustive testing and in order to reach the merit of high quality introspection, it needs integration with application being tested. This may give birth to platform support and operational problem issues (RSO, 2014).

Large SOA deployments
Service-oriented architectures focus on cooperative work, allows integration of discrete applications among different departments. A deeper understanding of internal structures in white box testing is a prerequisite which makes this approach inappropriate for web services – as it might result in lack of knowledge of operating systems, programming languages and hardware platforms (RSO, 2014).

Black Box Testing

Black box testing verifies whether the behaviour of a program conforms to user expectations or product specifications. This occurs throughout the software development life cycle. This verification technique is also known by names as functional testing, closed-box testing and behavioural testing. Black Box Testing is well suited for quick web service prototyping and for the systems that have enumerated or ranges of inputs to cover (Acharya and Pandya, 2013). It tends to find errors related to usability, performance, initializing, termination and timing (Westfall, 2009). A tester only sees two information streams, entering and leaving the "Big Black Box".

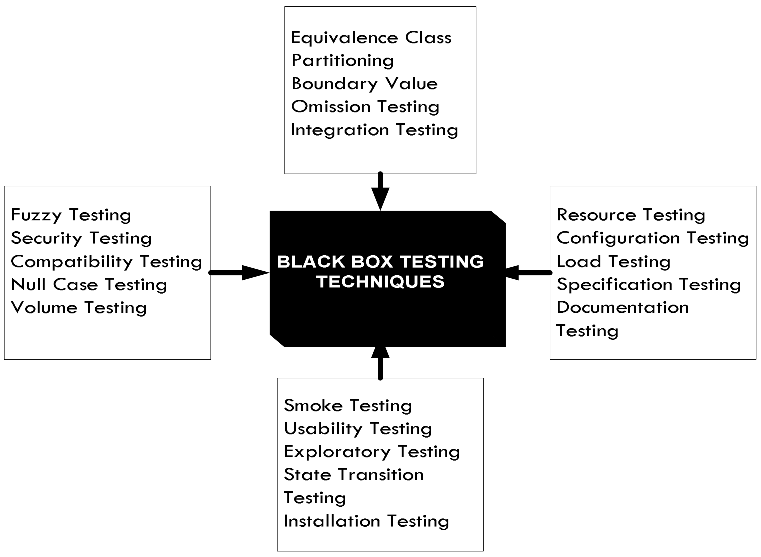

Fig. 3 Black Box Testing Techniques

Functionality Testing through Closed Box

The first step involves creating test cases after prioritizing the requirements; afterwards external interfaces are checked by giving various inputs to ensure that they fulfill all the functional requirements. Furthermore application is checked against resource utilization to ensure that it does not overload the allocated resources. The stress testing phase checks the system's behaviour by pushing it beyond the peak load conditions. Security testing focuses on testing the application for vulnerabilities and loop holes. In the end, the application is put under globalization test to ensure its support for international inputs/services.

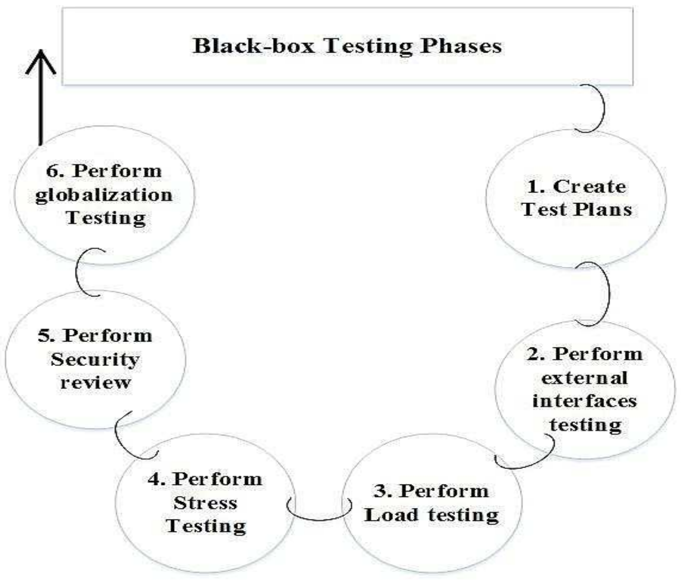
Black-box Testing Phases
1. Create Test Plans
2. Perform external interfaces testing
3. Perform Load testing
4. Perform Stress Testing
5. Perform Security review
6. Perform globalization Testing

Fig. 4 Black Box Testing Phases

Strengths

Reduced time factor
Unlike other testing approaches, black-box testing is considered to be a least time-consuming. It deals with structural level and level details testing without digging deep into the application's source code.

Quicker Test Case development
Completion of functional specifications leads to a quicker development of test cases.

Large Input Domain
Closed-box testing helps to check out the correct behaviour of system, not only for the valid inputs scenarios but also covers invalid input scenarios. This feature increases scenarios coverage and help to capture the values those lay inside or outside the boundaries. A large input domain can be tested; this approach can be advantageous to bug out defects at earlier stage.

No application knowledge required
Testers in black-box only deal with the behavioural aspect of one's application; they don't need a tight coupling with the baseline application code. Less internal knowledge and minor technical skills can help them to be more productive as there would not be the need to have knowledge of specific language or implementation.

Timely Defect Diagnosing
Defects that are impossible or difficult to identify can also be diagnosed e.g. interoperability issues and timing constraints.

Unbiased Testing
In Black Box Testing, roles of developers and users are clearly defined. This reduces dependency between the two and thus results in unbiased testing of the system.

Weaknesses

Exhaustiveness
Removal of defects from every boundary and input values makes it difficult to cope with the given schedule and time.

Ambiguous Specification
Requirements act as a baseline for the later stages in software development life cycle. Test cases derived from vague specifications, results in incorrect output. An error injected in later phases through requirements would result in high costs.

Script maintenance
Maintaining scripts would be difficult if the user interface modify frequently.

Inefficient testing
Tester has limited knowledge about application which leads to inefficient testing.

Opportunities

Cost Effective
Testing done by using automated tools is cheaper than using manual methods. As these tools require fewer resources it could reduce cost associated with testing phase.

Component Testing
As Black Box Testing externally checks the entire system, often it provides with opportunity to check the smaller components which are combined to build the system.

Threats

Blind Coverage
Test cases which are designed considering only boundary values may not be able to test each and every path, resulting in blind code coverage. To improve coverage there is no easy way to specify test case characteristics.

Code Quality
Quality of code may be a critical factor for some business domains. Organizations using

Black Box Testing may not have any means to improve quality of a code and its adherence to the coding standards.

Gray-Box Testing

The term is known as GRAY because it melds the assets of both types. Preceding work outlines gray-box as Gray-box testing = “Black box testing + White box testing + Regression testing + Mutation testing (Coulter, 1999).

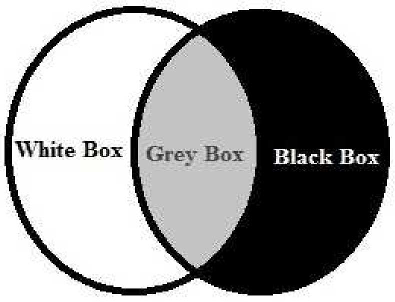

Fig. 5 Gray Box Testing

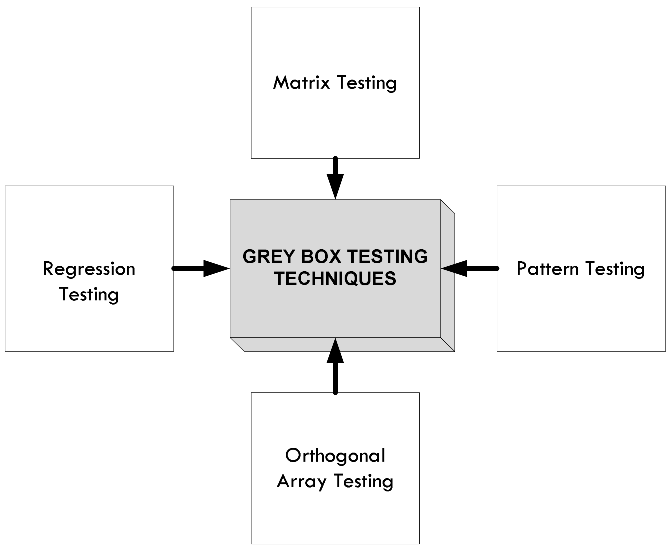

Fig. 6 Grey Box Testing Approach
Source: Gray-box testing technique (Software Testing Genius, 2014).

Strengths

Increased coverage
With Gray-box testing all layers of system can be focused for test case generation through blend of both approaches – i.e. White Box Testing and Black Box Testing. A tester knows the application's internal data structure and algorithm for designing test cases, along with the code of modules being tested. This increases the number of test cases generated, increases the coverage and deals with chances of blunder in user requirements and system functional behavior.

Domain knowledge
A complete knowledge about system's domain always provides an edge over one with limited domain information. A tester in Gray-box has an edge over others, the extent of knowledge helps him or her to produce better test data and test scenarios while 'scheming' the test plan.

Web Applications Support
Web Applications needs support for distributed networks, the absence of code makes it hard to imply WHT approach for testing, and Black Box Testing is a deal between customer and developer for requirements, so its usage might not be beneficial. This is where gray-box testing coins in. It is more advantageous to use gray-box, it supports the Web applications as substantial material is available in WSDL—Web service definition language.

Code Digging
Testers following Gray-box testing methodology do not have to depend on the application's source code like white box testing; rather they rely on functional specifications of system to be tested.

Non bias
Grey-box testing maintains a separation of tasks between tester and developer teams. This minimizes the personnel conflicts and aids the team's productivity.

Intelligent Test Authoring
Tester in Gray-box grips intelligent test scenarios e.g. Data-type handling, communication protocols and exception handling.

Weaknesses

Partial Code Coverage
Limited test coverage issue in gray-box testing arises because of unavailability of the source code, produces less productive application with probability of errors in the application's source code.

Exhaustive Testing
Like White Box testing checking conformance with expected for every possible input is unrealistic. This results in consumption of unreasonable amount resources, time and schedule slipping.

Defect identification
Difficult to associate defect identification in distributed applications.

Recurrence
Testing the same module for interfaces and data structures creates chances of having repetition of tests that are already done by programmer.

Test Case Execution Problem
Using gray-box testing in inspection of back-end modules activities, breeds problems during test case execution. First, when a failure is encountered, it causes the whole operation to be aborted by the component. Secondly, test case execution goes smoothly but the output produced would be incorrect[3].

Opportunities

SOA Support
Web service's distributed nature provides Gray-box testing with an opportunity, to be beneficial in removing defects from SOA. As far as web services are concerned, white box testing considered unsuitable because of its requisite of internal knowledge. Test cases generated by state art methods e.g. message mutation helps in identifying exception handling states and flow without source code for large arrays. White Box Testing proved beneficial for these methods. This approach can make gray-box testing to be more constructive and yield outcome like white-box testing.

Threats

Unidentified Bug
As the tester does not deals in the application's source code, it may leave some major bug undetected in code thus limiting the correct output chances

Test Cases Redundancy
A designer works on user's point of view in gray-box testing. The tests, once executed for the system interface, might create redundancy if designer has already run particular test, which would increase cost and schedule.

Conclusion and further study

The paper delivers detail insight into three popular testing methodologies—White-

3. Available: www.extremesoftwaretesting.com; accessed: 15.4.14.

box, Black-box and Grey-box texting. The selection of best approach should not only depend on the appropriateness each of these three techniques but also on the needs of the organisation. White Box Testing is considered as an expensive and exhaustive approach as compared to other stated techniques because of its detailed internal knowledge, limited test conduction without modifying program and impossibility of looking at every bit of code to find hidden error. The selection of Black Box Testing would help testers to identify errors without getting deep into the program. Besides these benefits, however, black box lacks functionally-specified test cases because of limited internal knowledge as some paths are left unidentified during testing. Bearing in mind the pros and cons of White box testing and black box testing, gray box testing characteristics are considered effective and more beneficial testing strategy in the light of the preceding analysis. To consummate reliable and quality software production, the concept presented can be further carried to develop a better understanding of each approach-type, by conducting their SWOT analysis. A separate study focusing on each testing technique would be beneficial to gain a deeper understanding of the issues surrounding these techniques as well as their implications for software development.

Acknowledgment
The authors would like to thank Eng. Muhammad Fahad Khan, Assistant Professor at department of Software Engineering, University of Engineering and Technology Taxila, Pakistan, for his guidance and dedication to our research.

Correspondence

Munazza Jannisar
Software Department
University of Engineering and Technology
Taxila, Pakistan
Email: munazzajannisar@yahoo.com

Ruqia Bibi
Software Department
University of Engineering and Technology
Taxila, Pakistan
Email: ruqia.kibria@yahoo.com

Muhammad Fahad Khan
Software Department
University of Engineering and Technology
Taxila, Pakistan
Email: fahad.khan@uettaxila.edu.pk

References

Acharya, S. & Pandya, V. (2013) *Bridge between Black box and White box- Gray box Testing Technique*, IJECSE, 2, (1), pp175-185; [Online], http://www.ijecse.org/wp-content/uploads/2012/12/Volume-2Number-1PP-175-185.pdf; accessed: 15.4.14.

Beizer, B. (1990) *Software Testing Techniques*, 2nd Edition, New York: Van Nostrand Reinhold Co.

Crosschecknet (2014) Patented Technology for Service and API based SOA [Online], www.crosschecknet.com; accessed: 15.4.14.

Coulter, A. (1999), *Gray-box Software Testing Methodology – Embedded Software Testing Technique*, IEEE, *Proceedings* of the *18th Digital Avionics Systems Conference* (DASC99), vol. 2, pp. 10 5-2.

IEEE (1990) Standard Glossary of Software Engineering Terminology (610.12-1990).

Khan, M. & Khan, F. (2012) *A Comparative Study of White Box, Black Box and Grey Box Testing Techniques*, IJACSA, 3, (6).

Khan, M. (2011), *Different Approaches to White Box Testing Techniques for Finding Errors*, IJSEIA, V5, (3) pp. 1-13.

RSO [REDStone software organization] (2014); [Online], www.redstonesoftware.com; accessed: 15.4.14.

Software Testing Genius (2014) Techniques of Continuous Improvement used by the Expert Test Managers [Online], Available: www.softwaretestinggenius.com; accessed: 15.4.14.

Westfall, L. (2009) *The Certified Software Quality Engineer Handbook,* American Society for Quality, Milwaukee: Quality Press.