**Testing - Verification Types**

**Black Box Testing** **(Functional testing aka Behavioural Testing)**

Is the testing of application functionality using software specifications and requirements (Ref 4) instead of internal structures and system design (Ref 12). The tester only has knowledge of what the software is supposed to do but not how (Ref 14). Black box testing can apply to unit (class in OO development ) (Ref 1), integration testing but is mainly used at the higher levels of testing such as system and acceptance levels of testing (Ref 17). Tests are carried out with the point of view of the user (Ref 5).

The goal of white boxtesting is to make sure the class meets requirements as specified in the specification (Ref 1).

Black box testing is done bychecking that for valid or invalid input that the output that is produced is as expected (Ref 14)**.**

Test cases are created from CRC cards, class diagrams, contracts and method specifications (Ref 1). Tests can be functional or non functional but are mainly functional (Ref 14).

Black box testing is used for normal unit testing (Ref 1) and it is efficient in the testing of large systems. Black box testing has the benefit that it can be used to identify discrepancies in the specification and any vagueness (Ref 5).

Test design techniques (Ref 17) include decision table testing, [all-pairs testing](http://en.wikipedia.org/wiki/All-pairs_testing), [state transition tables](http://en.wikipedia.org/wiki/State_transition_table), [equivalence partitioning](http://en.wikipedia.org/wiki/Equivalence_partitioning) (dividing input domain of program into classes of data which test cases can be written for) (Ref I) and [boundary value analysis](http://en.wikipedia.org/wiki/Boundary_value_analysis) (Ref 17).

Limitations of black box testing include firstly the likelihoodof repeating the same tests. Secondly all the inputs might not be identified in the testing process. Thirdly the test cases depend on a good specification.

There are tools available which are used to create the black box tests. There are also tools available to automate the running of black box tests.

**White Box Testing** **(Structural testing)**

Testing of the internal workings of a program based on knowledge of HOW system is implemented and not the functionality that the user can access (Ref 17, Ref 4). Access to source code is required (Ref 4) . Paths through the code are chosen by the tester for testing with inputs to these paths and expected outputs (Ref 17)

The goal of white box testing is coverage of the specification (Ref 11) in the code via covering as many statements, branches, decision points as possible during testing (Ref 2) by reviewing code and discovering errors or assumptions that would not be apparent to anyone treating the code as a black box (Ref 1).

White box testing involves testing paths, decision points (true/false), execute loops and checking internal data structures and applies mainly within the unit at unit testing level. It also applies to integration testing level (between components interacting via their interfaces) and system testing levels (between subsystems). Non implemented parts of specification or missing requirements is not tested (Ref 17, Ref 2).

It is a technique to prevent software failures and make software more robust.(Ref 4). Is used to test the intended and unintended behaviour of software, the design (actual implementation against logical flow), functionality, and uncover any faults, syntax checking and typographical errors (Ref 4, Ref 11)

A pre-requisite knowledge of the programming language and the program logic is required as well as the code and specification (Ref 11).

Three steps need to be followed together. Firstly design documentation, source code and design artefacts need to be analyzed and understood. Secondly tests are created to exploit the software based on this knowledge. Thirdly tools and techniques used for white-box testing need to be understood to carry out the testing effectively (Ref 4).

Common testing techniques include code coverage, fault injection, mutation and static testing methods (Ref 17), coding practices, exception and error handling (Ref 4)

If unit tests cover all the execution paths and data structures as required from the list below, it is a white box test otherwise it is a black box test if it does not (Ref 7)

The test plans are created from method specifications (Ref 1)

Test case design techniques include, [control flow](http://en.wikipedia.org/wiki/Control_flow) testing, data flow testing, branch testing, path testing, statement coverage and decision coverage (Ref 17).

White box testing is used when development is complex(Ref 1).

It is Impossible to test all execution of all paths and data structures in large systems through exhaustive testing . Best practice is to test important paths and data structures to make testing effective (Ref 11).

The white box testing tools include dynamic test tools such as JUnit to write and execute white box tests to cover the test case design techniques above and static code coverage tools such as Ecorbetura and EclEmma to determine how effective the coverage of the test case techniques has been. Eclipse IDE and other IDEs allow these software plug-in's to be installed and used from within the development environment (Ref 6).

**Grey Box Testing (Combination of White Box and Black Box testing)**

Grey box testing is executed at the user level (Black Box) but testers have knowledge of the internal workings including data structures and algorithms from documentation about code or program operation that would normally be out of view to design tests that will run at black box level (Ref 17, Ref 15).

The goal of grey box testing is to allow the tester to have better knowledge to make informed decisions when testing the system. (Ref 17).

Testing is done to the system under test (within black box), hence testing actual inputs and outputs outside the black box is not required. The tester will have better knowledge to make informed decisions when testing the system. (Ref 17). Tests are designed by studying programming code units (white box testing) and the actual tests are conducted like black box approach by using the exposed interfaces (black box testing) (Ref 10) .

A test environment and checking of the state of the system after performing actions to see if it behaves in the way expected are both allowed (Ref 17). Examples of actions may include seeding a database or observing database after running a SQL query etc.

This type of testing generates test scenarios with limited information. Typically used for testing exception handling, data type handling etc. (Ref 17)

For the test plan the tester does not require access to source code (Ref 17) but does require access to both high level and detailed documentation about the application to formulate test cases (Ref 15).

Techniques includematrix testing, regression testing (re-rerunning test cases if changes are made) and Pattern testing (verifying design patterns and architecture).

Grey box testing is used when the advantages of both white box and black box testing is required. Grey box testing is unbiased and allows testing boundary between developer and tester to be maintained. It is also used when intelligent test scenarios are required such as for data type handling, communication protocol and exception handling (Ref 15). Grey box testing is suited to functional or business domain testing or web applications. For web applications there may be an absence of source code and hence it is not possible to use white box testing. There may also be the need to main the contract between developer and tester and hence black box testing cannot be used also and hence grey box testing is the ideal (Ref 15).

This type of testinghas the limitation that it will give partial code coverage because of limited access to code and defect identification is difficult in distributed systems (Ref 15) .

**How the verification types fit into Software Design Lifecycle**

White box testing can be performed any time in the life cycle after the code is developed, and tends to be done during the unit testing phase as good practice, though it can be done also at the integration and system testing levels (Ref 4).

Black box testing is applicable to unit, integration, and mainly system and acceptance testing (higher levels of testing(Ref 17). It is also used in regression testing (Ref I).

Grey box testing can be used in other levels of testing but it is primarily used in integration testing (Ref 10).

**Tools that implements White Box, Black Box or Grey Box testing**

**JUNIT V4**

JUNIT is an open source unit testing framework that is available as standalone version or as in plug-in's for various IDE's such as Eclipse. It allows the testing of Java development code. There are versions developed for all popular programming languages and it is part of the xUNIT family (Ref 8 and Ref 16)

The goal of JUNIT is to allow the writing and running of tests (Ref 8).

JUNIT can be used to create white box and black box tests or a combination of both (grey box testing). This is done by creating a class in JUNIT for the class under test and writing tests as methods of the class. Annotations such as @Test tag is used to annotate the test methods and gives instructions to JUNIT compiler and runtime processing (Ref 13 and Ref 6). These are data about the program that is not part of the program itself (Ref 6). JUNIT has various functions that can be called instead of using print output statements to compare actual and expected results (Ref 13).

In black box testing the tests methods in the JUNIT test class are created to cover the different states that objects may have that are instantiated from the class under test (Ref 13). Objects can have multiple states and they don't have to be binary (Ref 13) It is best that the tester identifies as many states relevant to the specification. Tests should also be written to cover boundary conditions which is the testing of the in between of states (Ref 13). The tester has to keep determining if there are more states that need to be identified and formulate more tests to cover these. This process is aided by observing behaviour from the method specifications. The aim is to test each method once for every state that is identified by the tester.

Semantics of methods from the class under test can also be tested by testing object properties and creating test methods in the test class within JUNIT to do this (Ref 13) e.g. if order of output is important in the semantics then testing for ordering of output etc.

Tests can and should be written to cover special cases including trying to make the program crash. This is usually done by testing various allowed and non allowed input values (Ref 13).

For white box testing in JUNIT4 tests cases are written to ensure all internal operations are performed according to the specification and that the code in the methods have been adequately exercised as per white box testing design techniques previously mentioned under white box testing techniques (Ref 11).

For grey box testing JUNIT tests are designed by studying programming code units (white box testing) and the actual tests are conducted like black box approach by using the exposed interfaces (black box testing) (Ref 10).

**Central features** (Ref 8)

* Allows developer or tester to build test suites incrementally to monitor their progress through testing and to see unintended side effects
* Tests can be run continuously
* Results are immediately available to tester
* Test progress is shown in a progress bar and is red for fail or green for a passed test.
* A list of failed tests is available to view at bottom of screen
* Multiple tests can be run concurrently via test suite(s)
* Bugs can be easily corrected when found
* Tests can be ignored (Ref 3).
* Can do setup before running ALL tests e.g. database via SetUpBeforeClass()
* Can have methods running before a test e.g. to set up test environment via SetUp()
* Can add a method to be run after ALL tests have been performed via TearDownAfterClass()
* Can add a method to be run after a test performed via TearDown()

**Special technologies used (if any)**

JUNIT 4 can be extended with third party add on extensions to aid testing for a particular verification type. (Ref 3).

Davison Bullock MSc Information Technology Year 2 23/11/2012

**References**

1. Dennis A, Tegarden D, Wixom B. (2009). Chapter 13 - Development. In: John Wiley & Sons *Systems Analysis and Design with UML*. 3rd ed. Hobeken, USA: John Wiley & Sons. p512-520.

2. Heckman S, Williams L. (2008). *White Box Testing.* Available: http://openseminar.org/se/modules/46/index/screen.do. Last accessed 22nd November 2012.

3. Irriger, A. (2010). *JUnit.* Available: http://www.methodsandtools.com/tools/tools.php?junit. Last accessed 22nd November 2012.

4. Janardhanudu G, Van Wyk, K. (2009). *White Box Testing.* Available: https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/white-box/259-BSI.html#dsy259-BSI\_howto. Last accessed 22nd November 2012.

5. Koundinya. (2010). *Black Box Testing, Its Advantages and Disadvantages.* Available: http://www.codeproject.com/Articles/5579/Black-Box-Testing-Its-Advantages-and-Disadvantages. Last accessed 23rd November 20112.

6. Lachish, O. (2012). *Session 4 - Unit Testing.* Available: http://www.dcs.bbk.ac.uk/~oded/Tools2012-2013/Sessions/Session-4/Session-4.pdf. Last accessed 23rd November 2012.

7. Rosencruft, E. (2012). *Does “Unit Testing” falls under white box or black box testing?.* Available: http://stackoverflow.com/questions/9892963/does-unit-testing-falls-under-white-box-or-black-box-testing. Last accessed 23rd November 2012.

8. Rouse, M. (2010). *JUnit (definition).* Available: http://searchsoftwarequality.techtarget.com/definition/JUnit. Last accessed 22nd November 2012.

9. Softwaretestingfundamentals. (2007). *Black Box Testing: Types and techniques of BBT.* Available: http://www.softwaretestinghelp.com/black-box-testing/. Last accessed 22nd November 2012.

10. Softwaretestingfundamentals. (2011). *Gray Box Testing.* Available: http://softwaretestingfundamentals.com/gray-box-testing/. Last accessed 22nd November 2012.

11. softwaretestingfundamentals. (2007). *White box testing: Need, Skill required and Limitations.* Available: http://www.softwaretestinghelp.com/white-box-testing/. Last accessed 22nd November 2012.

12. softwaretestingfundamentals. (2007). *Types of software Testing.*Available: http://www.softwaretestinghelp.com/types-of-software-testing/. Last accessed 22nd November 2012.

13. University of British Columbia. (2007). *CPSC 211: Black Box Testing and JUnit.* Available: http://www.ugrad.cs.ubc.ca/~cs211/labs/JUnitLab/JUnitLab.html. Last accessed 22nd November 2012.

14. Various. (2012). *Black-box testing.* Available: http://en.wikipedia.org/wiki/Black-box\_testing. Last accessed 22nd November 2012.

15. Various. (2012). *Gray box testing.* Available: http://en.wikipedia.org/wiki/Gray\_box\_testing. Last accessed 22nd November 2012.

16. Various. (2012). *JUNIT.* Available: http://en.wikipedia.org/wiki/JUnit. Last accessed 22nd November 2012.

17. Various. (2012). *Software testing.* Available: http://en.wikipedia.org/wiki/Software\_testing. Last accessed 22nd November 2012.