UiO : Department of Informatics
University of Oslo

Comparing Black- and White-box testing environments using mixed research methods and AHP

Amiryasin Fallah Daryavarsari
amiryasf@ifi.uio.no

Master Thesis
Network and System Administration- spring 2013

Comparing Black- and White-box testing environments using mixed research methods and AHP

Amiryasin Fallah Daryavarsari
amiryasf@ifi.uio.no
Master Thesis

23rd May 2013

"At leve er krig med trolde i hjertets og hjernens hvælv.
At digte, det er at holde dommedag over sig selv."

"To live is to battle the demons in the heart as well as the brain.
To write is to preside at judgement day over one's self."

Henrik Johan Ibsen

Dedicated to my lovely aunt, Shaghayegh

Abstract

With the intent of cost reduction, flexibility in management, decreasing complexity, improving efficiency and more, the company for which this thesis work was performed, decided to investigate on the two radio unit software testing environments (black- and white-box) which were presumed to overlap. Since removing one environment or integrating them into one can lead to saving a lot of resources, a decision was made to assess their feasibilities. In order to do this study, quantitative and qualitative research methods are mixed, two questionnaires are done for revealing the testers opinions and neglected issues. The questionnaires helped the author of this thesis to find a suitable gateway through a large amount of internal documents and establish two interviews with key persons inside the R&D department. Moreover, the Analytical Hierarchy Process (AHP) is applied to this multi-criteria decision making problem in order to prioritise these two testing environments.The thesis strives to supply a fair judgment in comparing these two environments and their integration feasibility. It also tries to disclose their intangible properties.

Acknowledgment

It is an honor for me to express my appreciation to the following people and recognise their support in many different ways:

- Foremost, I would like to express my sincere gratitude towards my supervisor Hærek Haugerund for his support , motivation and encouragement. In addition to his guidance and efforts to overcome the challenges faced through out the thesis, thanks to him that trust me to be the teaching assistant at Intrusion detection and firewalls course as well as being positive and helpful.

- My sincere appreciation to Æleen Frisch. Her unique knowledge in many fields not only in system administration motivated me in many ways. Her excellent way of teaching and kindness like a friend are unforgettable. I feel great that I met such great person with such great personality.

- Thanks a lot to Ismail Hassan for making us a lot of problems in his courses to show us: no pain, no gain!

- I would like to take the opportunity to thank University of Oslo and Oslo University College for offering this Master degree program.

- Thanks to Kyrre Begnum, although I did not have chance to attend to any of his course, he gave me hope for the throughput of this thesis in the mid-term presentation.

- I am very grateful that I worked on my thesis beside helpful and friendly people at the company: Fredrik for his support, positive energy and attitudes. Team Lava: Hanieh, Sepehr, Anne, Xenia, Ahmad, Aisheng, Alexander, Per and Eva for their helps and kindnesses. Team Victorinox: Jurgen and especially Per for their information and time as well as helps.

- My Immeasurable thanks to Mahsa Samavati not only a classmate but

a great friend. Her cleverness and kindness is unrivaled. Thanks for her pure collaboration during this program.

- My deep gratitude to Vangelis Tasoulas because of his frankly friendship and helps. His enormous knowledge in Network and system administration seems to be unbeatable.

- To all my friends in Stockholm, especially to Pooyeh, she would be a lifelong friend and I look forward to reciprocate her unforgettable support. To Pooria, no one can be like him, thanks to make 5 months in Stockholm like a day for me. And finally to Mina, although she is a new friend, her kind heart and friendship forced me to state her name here.

- I am very grateful to be in a good company of fellow classmates. They have been an excellent companions throughout the program. I would to like to thank them for their nice collaboration.

- Thanks to all my friend and family around the globe, those who made and make all impossibles to possibles for me on the earth.

- Thanks to Shahrzad, she always motivates me in many ways. She assistes me in many issues with her unique sense of humor and patience.

- Last but not least, I would like to thank my beloved family including mother, father and brother. I could not be who I am and at the place I am right now without their total support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

All activities and tasks that a developer applies to design, develop, deploy and maintain a software application, can be considered as a part of the software development process.[1] This process begins by analysing the requirements and then planning based on defined requirements. The next activities are implementation and testing to cover the predefined requirements. Testing is one of the most vital activities in the software life cycle. Additionally it is considered as an important activity in most information systems and environments in which network and system administrators are responsible. It has direct impact on the development, maintenance cost and quality assurance. In fact, this activity ensures quality of the product and mitigate costs in maintenance. The main aim in testing is to find whether predefined requirements are met or not. Almost 50 percent of software cost is dedicated to the testing with intent of a reliable software.[2] It should be mentioned that uncovering errors in pre-delivery phases is more economical than being revealed after launching the product.[3, 4]

1.1 Motivation

This thesis work was performed in a company which provides telecommunications networks, television and video systems, IP networking equipments and all other services that are related to ICT. The thesis was offered by the radio unit (RU) software development section within the R&D de-

partment. In order to shorten the trouble report (TR) cycle between the sections which are involved in radio unit software, it was decided to merge units into some cross functional teams. In other words, design and formal verification units are merged into one section to reduce time that could be wasted in a broad TR cycle. Previously, the design section had its own environment to test radio unit software. This environment was designed for software designers by software designers and aimed to test the software from unit testing level up until a part of sub-system level testing which is called sub-system integration.[Figure3.1] All these tests were done from a white-box perspective. On the other hand, the formal verification teams were responsible for the formal verification. Their tasks were started when the designers had finished the sub-system integration tests and delivered the radio unit software to them. The formal verification section also had its own environment to test the radio unit software with the intent of executing sub-system verification tests and another testing environment for node level testing before final delivery to the next department. These tests were done from a black-box point of view. In this testing structure, if the functional verification would face any trouble which could not be resolved, a TR was sent to the prior level teams and this procedure was totally time and resource consuming. Hence it was decided to combine the cross functional teams to be responsible for the whole radio unit software development from the early stages in design to last stages in the node level. The sub-system level testing is the level in which the main conflict between these two white-box and black-box environments would happen in the new cross functional teams. In addition, It has been seen that some test cases seem to be redundant. Since it was not completely visible from which testing approach and by which testing environment for an instance capability should be checked, it was decided to make investigations about these two environments to reveal which one can fulfill the sub-system level goals or if the integration can be a solution for this conflict.

It is good to know that suitable testing reduces further cost, although it is a costly activity. The key principle that has to be considered in the testing is "Shift Left". It means that the quality measure should be shifted earlier in the life cycle or in other words, shifted left along the timeline. The product failures have to be unveiled in the earlier phase and shifting to left should be avoided. Actually, most of the efforts have to be put in the early stages and especially in the analysis stages. It should be noticed that finding error late in the cycle is going to cost quite a lot more and also it takes longer time for the feedback loops and the trouble report (TR) cycle. In order to reduce

the cost of testing without affecting quality, exhaustive testing should be prevented. Redundant testing increases resource usage in companies. This redundancy has higher likelihood to happen in levels where both black box and white box testing can be applicable. Hence, it make sense to compare test cases which are presumed to perform exhaustive testing and studying the integration feasibility of black box and white box testing efforts with intent of decreasing resource(human resources, time, cost and so forth) usage.

Moreover, the Analytical Hierarchy Process (AHP)[5] method can be applied to assist comparing these two environments with respect to their multiple criteria. The focus can be on the properties of these two environments which use black- and white-box testing approaches with the intent of disclosing less-known specifications of testing.

This thesis tries to be a step ahead and explores testing approaches and their pros and cons in addition to their integration feasibility. It is aimed at analysing the testing systems and their computing environments in a comprehensive way in order to allow the company in which this thesis was carried out, to provide appropriate business decisions in order to minimize resource usage while maintaining adequate testing accuracy for their products. Quantitative and qualitative survey methods are mixed to provide better understanding over two testing environments. Since the population of this study are real testers which are engaged in testing in their daily works, their ideas and opinions are distinguished as quite valuable to assist in the analysis.

Normally, network and system administrators face a pretty vast range of tasks to do such as security, monitoring, configuration management and so on. Network and system administrator's duties are quite human dependent which can cause errors in any level. Obviously, all of the assigned tasks need to fulfill service-level agreements (SLA) which is often a neglected issue by network and system administrators. Moreover, today, usage of the cloud environments is added to their tasks as well. Cloud computing and especially cloud testing should be addressed by network and system administrators. This requires proper understanding of testing approaches. Unfortunately, most network and system administrators do not know basic testing concepts and issues. Due to extension of new fields in their daily work such as cloud computing and also traditional work load like security, testing should not be an unknown or less-known matter any

more.

Additionally, testing in a information system or testing a information system itself may raise new confusion which can be even more complicated for those who has limited knowledge of it. A network and system administrator should learn methods to decompose the system under the test that can be a security, cloud or other architecture, into its components, test these components one by one, finally integrate these components and testing the whole architecture. Two major testing approaches which are involved in the testing from software to other information systems are black box and white box testing methods. The black box testing method is crucial when a network and system administrator has no control over the cloud and its components or can not see clearly what is happening inside the system. On the other hand, the white box testing can help network and system administrators as testers to address issues inside the system when it is completely visible for instance. These concepts can be also be applicable in testing an intrusion detection systems for instance, either we need to run a black box penetration tests on the intrusion detection system as holistic architecture or it is required to run a white box intrusion tests which follow the packets inside the system to reveal errors which happens on each node or component. Merging these two approaches could be addressed. Although, merging can reduce the costs beside the other benefits, it can raise some other disadvantages which need to be known.

Furthermore, in the case of cloud computing, as stated in the[6, 7, 8, 9, 10, 11], there are some challenges in the cloud-based testing as follows: Lack of standards, Security in the public cloud, SLAs, Usage, Planning, Performance. Since network and system administrators are mainly responsible for the cloud environments, it seems necessary for them to express their ideas about these challenges. This cannot be covered if they would not be aware of testing approaches and challenges. In this thesis, the potential challenges will be outlined and additionally two testing approaches will be described and evaluated. Since the topic is roughly new for network and system administrators, an outlook for the testing will be provided by comparing the white- and black-box environments under assessment. The main goal will be discussing different scenarios that are possible to occur while a network and system administrator is going to test a system or service.

1.2 Problem Statement

This study focuses on comparing two radio unit software test environments (the WBTest and BBTest) that each of them uses different testing approaches(black- and white-box testing). It tries to apply a proper way with the intent of comparing and assay their integration feasibility as well. During this report, it is strived to address the following questions:

- What would be the consequence of relying on only one of the two systems in an effort to reduce their significant resource consumption?
- What is a proper way to grade these testing environments in order to make an appropriate business decision?
- What would be the result in the case of merging these two environment into an integrated environment? Is it feasible?
- What are the redundancies in the current two-part testing efforts?
- To what extent may the lessons learned from radio unit software testing be applicable in network and system administration?

1.3 Thesis Structure

The layout of the thesis follows the background chapter where the two testing approaches are described and some examples of these methods are provided. Then the approach chapter comes to give explanations of the test strategy, the structure of the two testing environments which will be called the WBTest and BBTest environments, qualitative, quantitative and mixed methods, the analytical hierarchy process (AHP) and finally the plan of the study. The result chapter includes the results of the quantitative part of the study. The result chapter is followed by the analysis chapter which contains an overall analysis of the results, qualitative part of the study and applying the AHP method and document reviews of these two testing environments.

Chapter 2

Background

2.1 Software Development Process

A Software development process includes all tasks, activities, procedures which developers apply to develop and maintain a software and its related products such as codes, design documents, test cases and user handbooks.[1] It also known as software development life-cycle (SDLC). For a couple of years, Some have tried to investigate predictable and repeatable processes in order to meliorate and formalize productivity and also quality in software products. In the other hand, others have tried to use project management methods to prevent late delivery and meet expected cost and functionality on time in producing softwares.

A Software development process contains following activities: planning, implementation, testing, documentation, deployment and maintenance. It is obvious that every activity needs to be planned and a software developer does the same. In fact, they look for the requirements and analyse them precisely in order to design desired software based on requirements.[12] Normally, customers have no idea what software should do, but they know what they want at the end. In the other hand, a software developer distinguishes unclear and ambiguous requirements which should be redefined or even be out of the current project. After the planning phase, implementation will be carried out by programming the code of the software. Either parallel or next activity (based on the software development model) in the software development process is testing which

is considered as important activity in this process. This activity ensures that the software defects are detected and fixed as much as possible in order to have less bug in the final release. More detail information about testing, its methods and strategies will be provided in the rest of the background chapter.

Furthermore, maintenance is considered as a part of software development and it is how to deal with discovered failures or new requirements , therefore, documentation is noticeable activity in order to fast discovery and maintenance. Quantity of the documentation is based on the software development model (will not be covered in this thesis) which is chosen by the software team. Less documentation in the agile models and more in the waterfall models is expected. The remained activity in the software development life cycle is deployment which happens after testing phase and contains training, support and installation and custom configurations in the production environment.

2.2 Software Testing Basis

> "A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item."[ANSI/IEEE 1059 standard]

Testing is the activity in order to find if the specified requirements are met or not. Furthermore, one of the most vital activities in the software development life cycle(SDLC) is the testing which has direct effect on the development cost and quality assurance.[13] In the early years of system development, "hardware" had more than 80 percent of overall cost in a system development and only 20 percent and even less than it, was belonged to "software".[14] This trend was inverted in the 1990's. In this decade, the software cost by more than 80 percent got the hardware cost place by less than 20 percent in the total cost.[15] In the other hand, The testing cost is too much. Nearly 50 percent of a software cost is dedicated to testing in order to achieve a reliable software.[2] This cost can be even 2 to 3 times more in maintenance phase and normally 70 percent of this cost is because of the errors in the pre delivery phases.[3, 4] Furthermore, there are two opinions to guarantee reliability and ensure the quality of

software. The first and actually traditional aspect indicates that the main goal in software testing is detecting more defects, not proving software correctness.[16] This aspect seems to be not reasonable enough. The second one which is more contemporary, says that software testing is not only related to the distribution of defects but also related to how we use the software. [17] In order to decrease cost of the delivered software beside ensuring its quality, early defects detection is recommended. Although, in the software development life cycle, testing can be conducted in all phases, from requirement analysis until software deployment, it depends on the chosen model for software development. In the waterfall model, it should be done in the testing phase. In the other hand, in the incremental model, it can be executed in the end of each increment. Obviously, there is no ending point for testing and no one can say that the software under the test is completely tested. But, some aspects such as testing deadline, management decision, test case execution, code coverage, functional completion up until pre-defined point as well as reaching to a certain point of error rate, can finish the testing process.[18]

2.2.1 Testing Primary Objectives

According to [19, 20] , three major testing objectives are:

1. Testing in an activity in which a program will be executed in order to find errors in the software.
2. A test case that has high likelihood to find defects which is not discovered yet,is evaluated as proper test case.
3. A test can be considered as successful, when it unfolds still-existing-errors.

These three mentioned objectives, make it clear that a successful test is not a test with no error found in a software. In addition to detect defects in the software, testing shows that whether a software meets design and behavioral specification and its function is based on the defined specification or not. The noticeable point is that testing cannot represent the inexistence of the defects. In the other word, it can only show that errors are in the software not lack of errors.[20]

2.2.2 Testing Basic Principles

The suggested principles for testing that guide testers to perform successful testing are mentioned in the[21] and some of them are highlighted in the[20] and written in the following lines. As it said in the previous part, the main objective of testing is disclosing the errors. In addition, from customer viewpoint those defects which prevent the defined requirements to be met are most critical. Next principle indicates that the planning and designing of the tests can be started before code generation and also testing phase. In addition, "small to large" manner should be implemented in testing, which means that testing should be started in small components, then step by step to larger components and integrated components, finally in the whole system. Hence, testing has been divided into "unit testing", "integration testing" and "system testing". Furthermore, it is completely impossible to examine all possible path combinations even in the normal size program. It is totally time consuming and nonsense to test all possible paths. In order to cover this goal, logic of the conditions and program should be checked not all path combinations. And finally, based on primary goal of testing to find errors as many as possible in the program, software developers who wrote the program, are not best persons to test the created software. In the other word, testing should possibly be performed by an independent software tester.

2.2.3 A Good Test

As it suggested in the [22] "a good test" should have following characteristics:

1. A test can be considered as "a good test" when it had high likelihood to detect defects in the software. To reach this goal, tester should think backward and find the way to force the program to fail. Then find the way to catch the errors.

2. "A good test" is a test that is not redundant. The two or more test should not test the same things. In the other word, two or more tests should not detect the same defects. This aspect prevent to use extra resources and spending extra time and cost. In addition, every test should have different aims. This attribute becomes much more

important when there are several teams working on a large-sized software product.

3. The test with its "best of breed"[22] is "a good test". It means that with respect to resources and time shortage, only a subset of or only one group of tests could be sufficient. In the other word, a test with higher probability to find more errors or more important errors should be conducted in such occasions.

4. "A good test" is test which is neither too simple nor too complicated. It is possible to combine several tests into one in order to save testing time and cost. But, it should be prevented to have too complex and hard to understand tests. Generally, simpler tests are more efficient to be executed.

2.3 Testing Design Techniques

As it emphasized in the previous parts, the main intent of testing is design tests with high probability to find errors with respect to time and resources. There are several methods for designing tests but most efficient test case should fulfill the main goal of testing in the system which is disclosing errors and defects.

Two following basic ways can be used in testing process of a software product. First one cares about specified functions in the software. In this way test cases will be designed to test operability of functions based on the defined specification for the software. Errors in functions can be revealed at the same time that it tests the operability of them. The second way concerns about internal parts of the software. Tests are carried out to check if the internal components work correctly and to find errors in them. It should verify that the internal parts can satisfy the design specification or not. These two aspects are called black box and white box testing respectively.[20]

The black box testing is done at the interfaces of the software under the test. It tests that the system under the test accepts a valid input and based on this input, it produces an expected output or not. This process is done without any knowledge about internal components and interior structure

of the program. The most important approach in this method is testing external interfaces in the software.

In the other hand, white box testing is investigating the internal operation and logic of the software under the test. The internal structure of the software is also a matter in this aspect. In this type of testing, program structure such as conditions, loops and the other data structures of the software will be examined to satisfy design specifications. The tester should test logical paths through data structures to check if they operate appropriately. It seems, this type of testing can uncover all possible errors and result will be completely a correct software. But after taking a deeper investigation over it, it show that performing such test is a huge task and takes a lot of time. The calculation in the[20] indicates that it will be 10^{14} possible paths for a small program with 100 lines code which contains two nested loops each from 1 to 20 in addition to four if-then-else conditions in the internal loop. Now just presume an imaginary (perfect, high-performance) processor which has been not produced yet, if it only takes 1 millisecond for executing a test case and preparing the results, it would take 3170 years to test such program with such redundant testing. To solve this problem and prevent redundant testing, some important paths and data structures should be selected and examined.

2.3.1 White Box Testing

The white box testing methods which also called glass box testing, are used to examine all independent paths in the software at least once as well as all logical choices(true or false) in addition to perform loops in their boundaries and finally validity check of data structures. These tasks are mostly based on the "coverage". This coverage can be on statements, loops, paths, conditions and so on.

Tom McCabe[23] proposed "Basis Path Testing" as one of the white box testing techniques in the 1976. Basis path testing is a simple but not complete technique in control structure testing which is going to be explained in following lines.

2.3.1.1 Flow Graph and Cyclomatic Complexity

The flow graph and cyclomatic complexity are two concepts which are used in explaining basis path testing and will be described in the following lines. As it is shown in the figure 2.1, each data structure has its own symbol in the flow graph and these flow graph symbols are used to illustrate the control flow.

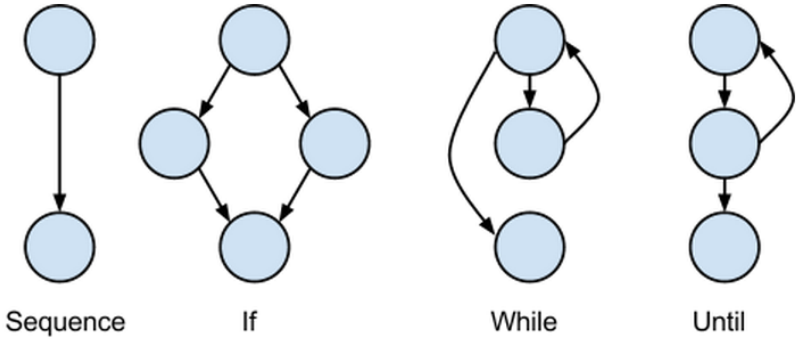

Figure 2.1: Flow graph symbols for data structures.

The following simple pseudo code which calculate average of the ten numbers, is used as an example to depict flow chart, flow graph and further explanations about deriving test cases by using graph theory.

```
Begin
i and sum are equal 0
        while i < 10
                input x
                if x is negative then
                        x = -x
                sum = sum + x
                increment i by one
        end of while
if sum less than 100 then
        avg = sum / 10.0
        print avg
else print sum is too large
End
```

As it depicted in the figure 2.2, the flowchart is resulted from pseudo code.

Each circle in the flow graph is called flow graph node and all arrows are called flow graph edges which represent flow of control. Each edge in the graph should be ended at a node. As it can be seen in the converting 3,4 and also 9,10 from flow chart to flow graph in the figure 2.2, it is possible to merge sequence boxes together or with a decision or condition diamond and conclude a single circle(node). All areas bounded by nodes and edges construct a region. In addition the external area is also called a region(region 4 in the figure 2.2). The nodes which have a condition are called predicate nodes. These nodes normally have two or more outgoing edges. Moreover, when there are multiple conditions in a decision box(that is called compound condition), each condition must be represented by a node.

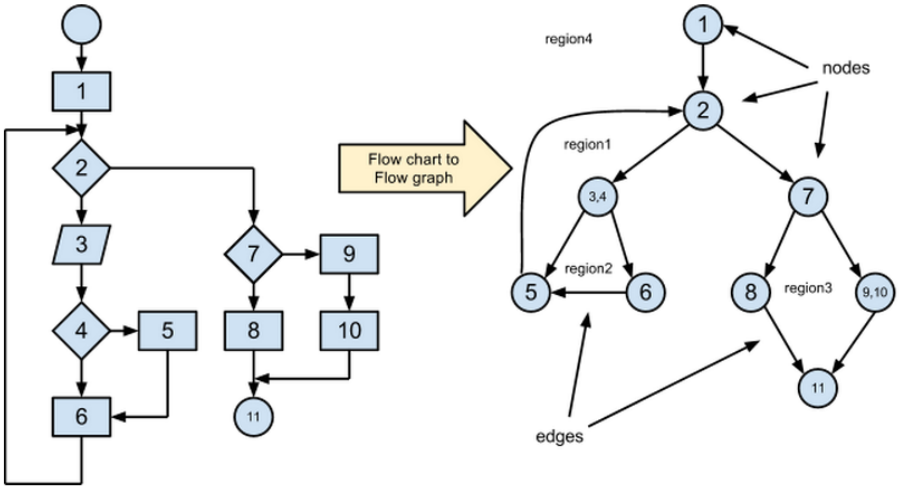

Figure 2.2: Converting flow chart to flow graph.

After short description about flow graph and its terms, it is time for cyclomatic complexity. The cyclomatic complexity is a measure for the logical complexity of a software or program.[23] The computed value by the cyclomatic complexity represents that how many independent paths exist in the program. Furthermore, The maximum number of tests in order to cover that all statements are performed at least once, is equal to the cyclomatic complexity measure. For making it a bit more clear, an independent path is a path which passes over a new set of statements or a new condition. The set of all independent paths build basis set for the flow graph and program. In the given pseudo code and concluded flow graph, independent paths are:

- Path1: 1, 2, 7, 8, 11
- Path2: 1, 2, 7, 9, 10, 11
- Path3: 1, 2, 3, 4, 5, 2,...
- Path4: 1, 2, 3, 4, 6, 5, 2,...

As it can be seen in these four concluded paths, each new path introduce a new edge. In addition, three points at the end of path 3 and 4 shows that those are covered by another independent path. In this example, those three points can be filled either path 1 or path 2. Therefore there is no need to add both paths and consider them as two independent paths. Moreover, all combinations of these four path cannot be considered as a new independent path due to exhaustive nodes in the paths.

By concluded paths, we can see what tests should be designed with the intent of examining all possible paths and executing all conditions in both their false or true responses at least once to disclose errors in the procedural program. These four paths are shaped a basis set for depicted flow graph and tests should be designed for execution of these paths.

Implementation of the graph theory for computing the cyclomatic complexity will give us exact number of independent paths. The cyclomatic complexity can be computed by one of the upcoming ways.

1. The number of the regions which bounded by edges and nodes in the flow graph is equal to cyclomatic complexity. As it can be seen in the mentioned example, there are four regions and as result, four independent paths in the flow graph.

2. The cyclomatic complexity for the flow graph can be computed by following formula:

$$v(G) = e - n + 2p$$

   Which e refers to number of edges, n refers to the number of nodes and p is the number of connected components which is one in the example. There are eleven edges and nine nodes in the flow graph 2.2. The result by applying the formula will be four.

3. And finally, cyclomatic complexity can be computed by using the number of predicate nodes plus one. In the mentioned example, three predicate nodes can be seen. Hence, the cyclomatic complexity is four again.

As conclusion of applying graph theory and cyclomatic complexity in this example, there are four independent paths which can ensure execution of all statement in the program. In addition, based on [23], there are six properties for the cyclomatic complexity that should be noticed:

1. $v(G) \geq 1$
2. $v(G)$ is the upper bound number of independent paths in the graph G and the value of it is equal to the size of a basis set.
3. Removing or adding the functional statements(nodes) to the flow graph G, will not change $v(G)$.
4. Flow graph G has just one path if and only if $v(G)$ is equal to one.
5. Adding a new edge to the flow graph G increases $v(G)$ by unity.
6. $v(G)$ is only based on the decision structure of the flow graph.

It is time for deriving test cases from the independent paths. Distinguishing predicate nodes is the main help to derive test cases. In the other hand, the path of the program is decided in these nodes. All test cases are performed and their result will be compared with the expected results. Execution of all test cases can guarantee testing of all statements at least on time.

There is another data structure that is called graph matrix which can help testers to find the cyclomatic complexity and also is useful in basis path testing. The graph matrix is a matrix that its number of columns and rows is equal to the number of nodes. In this matrix, each row and column represent a specific node and the matrix entries are the connections between nodes which are edges. A simple graph flow example and its matrix are shown in the figure 2.3.[20, 24] The nodes are shown by numbers and edges by letters. These numbers and letters are mapped into a corresponding matrix. By adding weight to each entry in the matrix, control structure testing will be much more powerful. It means that each

weight link can contain further information about the control flow. This weight can also indicate execution probability of the weighted edge, time spending for processing between two nodes or also resources such as memory that are needed during traversal between two nodes.[20]

Simplest demonstration can be based on incident matrix, which no connection and connection is shown by zero and one respectively. The corresponding connection matrix is also depicted in the figure 2.3.

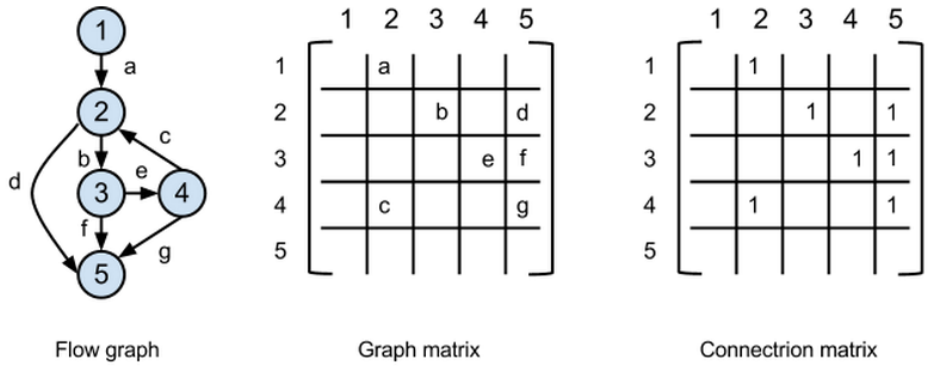

Figure 2.3: Graph matrix and connection matrix

All letter in the graph matrix, has been replaced by 1 to show connectivity and existence of a link and build up connection matrix. In addition, rows with at least two entries, indicate that there is a predicate node in the graph. Three rows with such specification can be seen in the connection matrix. Hence, based on the third method to find cyclomatic complexity, $v(G)$ will be number of predicate nodes plus one which will be four in the flow graph depicted in the figure 2.3.

In order to improve white box testing quality, we need to add some other testing techniques to the basis path testing. In the other word, it is not enough to test the software with basis path testing. Other technique such as condition testing and loop testing will be shortly discussed in the following lines to widen coverage in the white box testing method.

2.3.1.2 Condition testing[25]

Condition testing is a technique that is used to test logical conditions in the code. These conditions can be a true/false (boolean) variable or a relational statement in the program we are testing. This technique has two major benefits. First, it can simplify testing of conditions. Second, Condition testing can provide assistance in creating additional test cases for examining the program. Furthermore, condition testing is not only for detecting defects in a condition statement, but also for detecting other possible errors in the code. In the other word, effective condition testing strategy will also lead to effective testing in the other part of the program.

The simplest condition testing strategy is branch testing. In case of having a compound condition, both true and false branches are required to be executed and examined.[26] The other strategy is domain testing which needs three or four tests for a relational expression.[27] Testing greater than, equal to and less than possibilities are three tests which is mentioned in the previous sentence.[28] Obviously, these three tests will uncover errors in the relational expressions.

Branch and Relational Operator(BRO) testing technique which is suggested in the [25] as a condition testing strategy, guarantees uncovering a condition errors in its branch and operator operators, but this should be noticed that all true/false(boolean) variables and also relational operators in the condition under test should occur only one time and there should be no similar variables.

2.3.1.3 Loop testing

Loop testing(a white box testing technique) concentrates on examining loops in the program and checking their validity. As it stated in the [24], there are four classes for the loop structure. They are simple loops, nested loops, concatenated loops as well as unstructured loops. For each class, there are some tests that can be derived and executed.

Simple loop. Tests that can be used for simple loops are:

1. Skipping the loop.

2. One time pass via the loop.
3. Two times pass via loop.
4. m times pass via the loop ($m < n$)
5. $n-1, n, n+1$ times pass via the loop

notice that n is the maximum number of valid passes via the loop

Nested loops. extending the simple loop testing to nested loops testing will lead to increasing the number of tests for nested loops. This growth would be even geometrically by increasing the level of nesting in loops. There is an aspect that is suggested in the [24] which can help us to decrease the number of possible tests in the program.

1. Running mentioned simple loop tests on the innermost loop. All other loops should be set with minimum values as their loop counter. Other tests which contain out of range and excluded values should be added as well.
2. In the next step, after testing innermost loop, rest of the tests should be executed one level out while outer loops are at their minimum values.
3. Perform these three steps until testing all loops.

Concatenated loops. There are two types in such loops. First, loops that are concatenated but independent which can be tested with simple loop testing method. Second, loops that are concatenated but with dependency. For example, second loop would need last value of previous loop as it initial loop counter. In such cases, the method for nested loops testing is suggested.

Unstructured loops. Obviously, as it can be known by its name, there is no guarantee for suitable tests in such kind of loops. In order to test these loops, software designer should change the structure of such loops and then testing them.

2.3.2 Black Box Testing

Behavioural testing as known as black box testing method, is a testing technique without having any information about internal structure or procedure of a software. In addition, it cares about functional requirement. This method is being used by software testers in order to provide suitable

inputs for checking if the software satisfies functional requirements or not. As it stated in the [19], the following criteria will be satisfied by test cases which use black box testing method. First, those test cases that reduce(more than one test case) the amount of extra test cases that must be derived for proper testing. Second, those test cases which reveal not only an error with a specific designed test, but several classes of errors in the software. The lines that follow, provides some well-known black box testing methods and techniques.

2.3.2.1 Graph-based Testing

Discovering program objects which can refer to the data objects and modules in the program and connections between objects that show their relationships, is the initial step in the black box testing. After this step, set of test cases can be derived with intent of verifying predefined relationships between objects. [29] This goal can be reached easier if we create graph based on program objects and their relationships and then examining the graph by series of test cases to detect defects.

The mentioned graph has following specification and notion : nodes that show program objects, node weights that contain attribute of nodes, links that show relationships between objects and link weights that keep properties of links. As it can be seen, stated properties are similar to the graph properties of basis path testing method in the white box testing. This graph is an extended model of the flow graph in the white box testing. The simple demonstration of this graph is in the figure 2.4.

As it can be seen in the figure 2.4, nodes(objects) are illustrated by circles and links(relationships) are shown by three different shapes. First, directed link that shows one way relationship. it means that the relationship moves from one object to another one based on the direction of the arrow. Second, Bidirectional(Undirected) link which is illustrated by a simple line and shows both direction relationships. This link known as symmetric link as well. And finally, parallel links which is shown by two or more parallel lines and indicates different relationships between two objects.

In the [29], Beizer explained some methods in the black box testing which use the previously explained graphs to derive test cases:

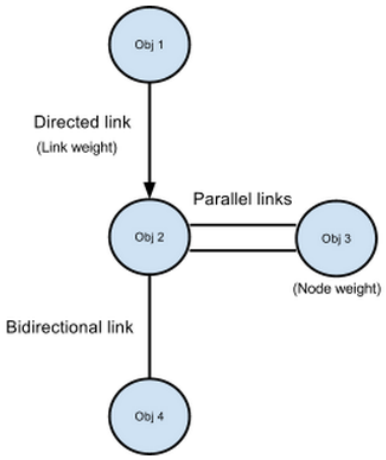

Figure 2.4: Graph properties

Transaction flow modeling.

In this type, transaction steps are indicated by the nodes. For example, steps to register in a web site. In addition, links are representative of logical relationships between nodes. The data flow diagram can help testers creating this graph.

Finite state modeling.

Various user observable states are shown by nodes and links show the required transitions in order to move to different states(nodes). The state transition diagram can be helpful for creating this type of graphs.

Data flow modeling.

The nodes represent data objects and the links indicate the required transformation for translating data objects during their traversal.

Time modeling.

The nodes are representative of program objects and the links show the sequential relationships between the objects(nodes). In this method, link weights are applied to show the execution time.

In the graph based black box testing, first step is identifying the objects and their attributes to build up nodes and nodes weights. Next step is discovering the relationships and their properties to create links and links weights. In case of any loop in the graph, the loop testing method mentioned in the white box testing can be applied. For identifying the effect of the relationships on the graph objects(nodes), transitivity between

objects should be studied. As an example, if "A is required to compute B" and "B is required to compute C". The transitive relationship is "A is required to compute C". It means that the test case must contain different values of B and also A.[20] Furthermore, the symmetric link should be considered for testing reflexive relationships. It should be noticed, there are two main aspects in designing test cases which should be addressed, node coverage and link coverage.

2.3.2.2 Equivalence Partitioning

It (known also as Equivalence Class Partitioning) is a black box software testing technique that divides the input data of a program to the partitions of data that are equivalent from which test cases can be derived.[20] This technique tries to disclose "classes" of errors with intent of reducing number of test cases and shortens the testing time. In addition, designing test cases in this method is depends on the assessment of equivalence classes for the input. It should be noticed that existence of symmetric, transitive and reflexive relationships between objects shows that there is an equivalence partition for the input.[29] In addition, a serie of valid and invalid input states. The upcoming guidelines provided in the [20] , assist us for distinguishing a equivalence class.

1. A specific numeric value as input: one valid and two invalid equivalence classes are defined.

2. A range of values as input: one valid and two invalid equivalence classes are defined.

3. A set of related values as input: one valid and one invalid equivalence classes are defined.

4. A boolean condition as input: one valid and one invalid equivalence classes are defined.

By using these guidelines, the equivalence partitions will be derived and then based on these equivalence classes which are reduced the number of test cases, suitable test cases will be created.

2.3.2.3 Boundary Value Analysis

For unknown reasons, most of the errors happen at the input boundaries not in the center. In the other word, testing around the input boundaries is more valuable for testers to reveal errors. Boundary Value Analysis as outcome of this trend, helps to create test cases in order to execute black box testing on software under the test. The boundary value analysis technique is a complement for the equivalence partitioning. It assists to design test cases not for any member in a equivalence class but for those inputs which are at the boundaries or edges of the equivalence class. As it stated in the[19], in addition to deriving test cases from input domain, boundary value analysis can derive test case from the output conditions.

The following guidelines are provided to derive boundary value analysis test cases.[20] As it mentioned in the previous paragraph, the boundary value analysis technique complements the equivalence partitioning technique, so the guidelines seem to be optimized for boundary value analysis technique.

1. A specific numeric value as input: proper test cases should be designed to examine maximum and minimum values in the class. Test will be completed after deriving test cases for the values above and below the maximum and minimum values which are already tested.
2. A range of values as input: in this situation, the values which bound the interval, should be considered in test case development and the values above and below these boundaries.
3. Guidelines 1 and 2 can be applied for output domain as well. It means that the boundary value analysis technique which is mentioned in the first two guidelines can reduce the number of test cases for examining output conditions.
4. In case of having internal data structure with predefined boundaries like a limited size array, test cases should be designed to examine the data structure at its boundaries.

By Using these guidelines, probability of detecting defects will increase by having such boundary testing.

2.3.2.4 Orthogonal Array Testing

Orthogonal array testing is a black box testing technique which can happen in case of having number of input parameters but this number is relatively small but too large for executing redundant testing.[20] There are some applications which have several input parameters that all the parameters accept limited number of values. Although it is possible to perform redundant testing and examining all possible input combinations in case of limited number of input values, when the number of parameters increase, the exhaustive testing will be completely impossible. As it mentioned above, in such cases(small number of input parameters but large to prevent redundant testing), the orthogonal array testing is applicable. This technique is so suitable for detecting region faults which represent faulty logic in the software.[20]

following example helps to describing orthogonal array testing more clear. Imagine a software with four input parameters: A, B, C, D. Each of these parameters can take three values 1 , 2, 3. Number of possible test cases will be $3^4 = 81$. It is large and seems impossible to have this amount of exhaustive test cases. The following testing would be defined, If testers use "one input at the time" testing technique:

$$(A, B, C, D) = (1,1,1,1), (2,1,1,1), (3,1,1,1), (1,2,1,1),$$
$$(1,3,1,1), (1,1,2,1), (1,1,3,1), (1,1,1,2), (1,1,1,3)$$

This technique has limitation in error detection. In fact, it is helpful, if the input parameters have no interaction with each other. It can detect single mode faults(when single parameter causes faulty functionality in the software). In case of faults caused by two or more parameters, it is disable to disclose logic faults.[30]

The orthogonal array testing assist testers to reduce number of test cases and perform more proper tests. In the orthogonal array testing, an L9 orthogonal array will be created. The corresponding L9 orthogonal array of the stated example can be seen in the table 2.1.

Following results in detecting different kind of faults can be obtained by orthogonal array testing technique. [30]

Test Case				
	A	B	C	D
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Table 2.1: L9 orthogonal array

Single mode faults. This technique disclose and isolate all single mode faults in the software. If value 1 be as source of error, first three test cases will fail and show the faults. Failure of first three test cases can assist to analyse the source of error. It can be seen that this error happens when A gets the value 1. In this step, the logical process which leads to this failure will be isolated for fixing.

Double mode faults The double mode faults can be detected by this technique. Occurrence of a problem when two specific items happen together, is called double mode fault. It shows inappropriate interaction between those two specific parameters.

Multimode faults Some multimode faults can be detected by orthogonal array but it cannot guarantee it.

As it mentioned above based on the results in the[30] , Orthogonal array testing assures detecting single and double mode faults but not multimode faults.

2.4 Testing Strategies

In order to achieve a well constructed software, it is required to derive test cases by mentioned method and execute test cases in an organised

way. Such organised way builds up the testing strategies which indicate a couple of steps for testing. Testing strategies assist testers to distinguish that how they execute test cases. For example, if a new component is added to program, how should it be tested? Should we run the tests on the small part or on entire software? The responses to such question are provided by testing strategies.

Testing can be designed and planned in early stages of software development. Such planning can conclude a testing template in which testing methods and design techniques of test cases will be declared. This template composes testing strategy in the software process.

2.4.1 Verification and Validation

Verification and validation(V&V) which can also be referred to the software quality control, are the process of checking if the software satisfies predefined specifications or not. In addition, Software testing is one of the activities inside the V&V. Verification and validation are not the same. A precise definition which shows their difference is stated in the [31]:

- Verification: Are we building the product right?
- Validation: Are we building the right product?

Verification and Validation contain a wide set of tasks in the software quality control. As it stated in the [32], each of the software process phases contain several verification and validation task such as requirements validation in the requirement definition phase, design evaluation in the design phase and so on. Although testing is the last task which uncover errors in the V&V process to determine quality of software, other tasks are also necessary. In addition, it cannot guarantee the quality. In the other word, testers cannot examine the quality by testing. In fact, there are several factors that are important in the software quality control. The connection between quality control and testing is described by Miller: "The underlying motivation of program testing is to affirm software quality with methods that can be economically and effectively applied to both large-scale and small-scale systems." [33]

2.4.2 A Testing Strategy

In order to describe a testing strategy shortly, assume that spiral model is a chosen software development process model. Software engineers begin the software development process by defining the software role and system engineering. Moving one step inside the spiral, requirement analysis will be the next step which contain function, performance, validation and other factors and specifications that should be analysed. After requirement analysis of the software, it will be time for design and then coding at. A testing strategy is to move outward along the spiral model. It means that the first step for testing the software is unit testing which execute tests on each unit in the written program. Next step will be integration test that concentrates on the design and how the software architecture is constructed. One more turn outward leads testers to validation tests in which predefined requirements is validated. Finally, system test will be executed to test entire software system and all other factors together.Unit testing mostly uses white box testing techniques. Obviously, the goal of unit testing is code coverage and detecting maximum code defects in the program. After unit testing components, it is time for the integration test which addresses problems with both verification and program structure. Although integration test widely uses black box testing techniques to uncover functional and behavioural errors, white box testing techniques will be used to disclose errors in the major control paths as well. After successful integration tests which result an integrated software, validation tests will be performed. Validation test cares about functional, behavioural as well as performance requirements. Hence, its condition fits to the black box testing techniques. And finally system tests will be executed to verify that the software is working properly with its system component such as database, hardware and so on. [20]

2.4.3 Completion of Testing

here is no clear answer to the questions about completion of testing by the software testers. If you ask a tester about this issue, you may receive two responses. First, “There is no final point for testing.” which refers the rest of testing to the customer. In fact, every execution of software by customers can be considered as a testing in the real environment. Second, “Testing will be done when there is not enough resource(time, money and so forth) to

perform more tests." which means the completion point of testing depends on resources.

These responses may give the testers a quick overview about the completion testing point of the software, but software testers require a scientific answer to this question. A statistical response to this question can be found in the [34]. They suggested: "No, we cannot be absolutely certain that the software will never fail, be relative to a theoretically sound and experimentally validated statistical model, we have done sufficient testing to say with 95 percent confidence that the probability of 1000 CPU hours of failure free operation in a probabilistically defined environment is at least 0.995." A function of execution time for software failure models can be conducted by applying statistical modeling as well as software reliability theory. Logarithmic Poisson execution time model as a function for failure model which is suggested in the [34] presents as follow:

$$\mu(\tau) = (1/\theta)ln(\lambda_0\theta\tau + 1)$$

$\mu(\tau)$ refers to the cumulative amount of expected failures once the program has been examined in the execution time of τ.
λ_0 characterizes to the initial program failure intensity in the start of the testing period.
θ refers to the exponential mitigation in the failure intensity as defects are detected and repairs are made.

Derivative of the function $\mu(\tau)$ results to the instantaneous failure intensity.[20]

$$\lambda(\tau) = \lambda_0/ln(\lambda_0\theta\tau + 1)$$

This formula assists testers for predicting the reduction of errors while testing the software. It should be noticed that the logarithmic poisson execution time model can be applied for predicting required testing time with suitable failure intensity, if the data gathered during the test and predicted by logarithmic poisson execution time model are close enough. By gathering proper factors and using such statistical and

reliability modeling, answer to the "when to stop testing?" can be provided. Furthermore, in the [35], a Bayesian approach in order to predict the number of failures in a software, by using the logarithmic-poisson model, a nonhomogeneous poisson process (NHPP) is applied to describe failures in the software.

In the following lines, different testing strategies will be described shortly. The main focus will be on the Integration testing which both white box and black box testing techniques are responsible and some overlaps may happen.

2.4.4 Unit Testing

Unit testing as known as component testing focuses on code coverage and is done by white box testing technique. By applying component level design description, main control structures and paths will be tested in their boundary to detect errors in them. All the methods which are explained in the white box testing techniques such as basis path testing, loop testing, will be used to test software components and reveals error in the program structure. In the unit testing, test cases should be derived to detect defects due to wrong control flow, incorrect comparisons specially in the conditions and also fault computations.

2.4.5 Integration Testing

Following statement may be expressed by a beginner, “If the software components work correctly one by one, obviously they will also work correctly together.” The main issue arises by “working correctly together”. It should be noticed that there is no guarantee that if a component work individually, it would work in integration with another component and reverse. Some components may affect the other ones, global data structure may lead to a problem and integrated subfunctions may not present the demanded functionality. Goal of the integration testing strategy is to construct the software while performing tests to reveal errors. In fact, by this strategy, individual units will be put together to build the software based on the design specification. It also can be called component integration stage.

There are two type of integration testing. First one follows "Big Bang" aspect. it means that putting all components together at once and then trying to test and disclose errors in the software. As it can be imagined, the result will be a messy software which is so hard to reveal failures in it. In this approach, debugging an error may arise another error and the testing process will be trapped in a endless loop. Second aspect is incremental integration which is opposite to the big bang aspect. In this approach, small components build the software incrementally. It results to simpler error detection, error isolation and finally error correction. As it can be seen, this aspect gives a systematic view in developing and testing the software.[20]

2.4.5.1 Top-down Integration

One of the incremental aspects in the integration testing is the top-down integration. In this approach, the integration direction is downward and is started from the highest level via control hierarchy. There are two manners with intent of subordinating modules to the main control module. Depth first and breadth first are the stated manners.

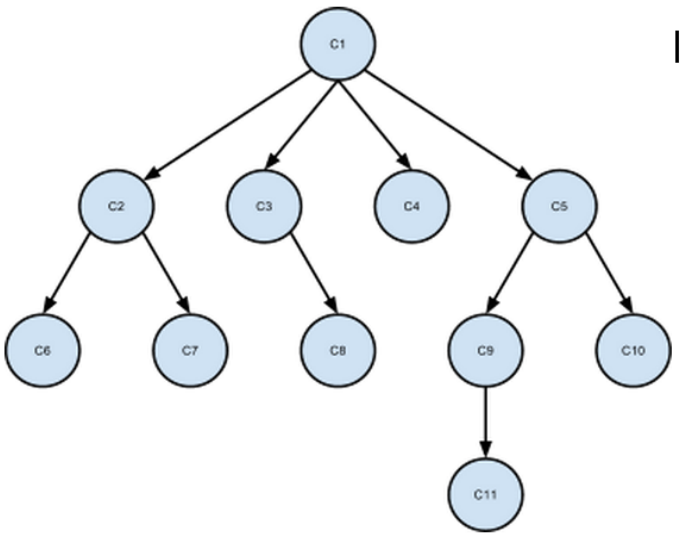

Figure 2.5: Top-down integration

In the depth first, developer may merge the components on a major path control path which can be different based on the specification of the application. Figure 2.5 can assist to understand these manners better. For using depth first integration, by choosing the right path in the figure 2.5 , components C1,C5,C9 are merged. Then C11 would be integrated.

In addition, for suitable functioning of the C5, component C10 may also be integrated. After integration of the the right path, other path will be integrated one by one in the same way. In the other manner, breadth first integration, Components which are going to be integrated, will be chosen in each level horizontally. Referring to the figure 2.5 , components C2,C3,C4,C5 will be chosen in the first step of integration. Then, the next level of components (C6,C7,C8,C9,C10) would be integrated. It will be carried on until complete integration. The integration steps are stated in the[20] and are as follow:

First The test driver for integration testing is the main control module and all the stubs subordinate to this driver.

Second Based on the integration manner (depth first or breadth first), stubs will be substituted with real modules and components.

Third After each integration tests will be executed.

Forth Next stub will be substituted by the actual component when the previous test is completed.

Fifth And finally, regression tests which will be explained in the next part, will be executed to reveal any expected errors after integration of a new component.

This guideline will be done from second step up until completely constructed structure. Due to nature of top down integration, major control problems which are important to be distinguished early, can be detected at the higher level. Although, top down strategy seems to be simple, some logical issues can happen while doing the integration. One of the issues happens when low level components need to test upper level components. As it explained in the guideline, actual components have been substituted by stubs at the start of the top down integration, hence, significant data cannot flow to higher level. In order to solve such problems, there are three options. First, testers would postpone so many tests and waiting until stubs being substituted by real components. Difficulty in finding the reason of errors and also changing the nature of the top down integration are common side effects of the first option. Second, testers would develop new stubs that can perform limited functionalities in order to represent real components. Obviously this option causes high overhead especially when developing stubs

which are more complicated. Third, testers would use bottom up integration which merge components from bottom level to higher level.[20]

2.4.5.2 Bottom-up Integration

In the bottom-up integration, construction and testing the software will be upward. It means that integration begins from low level components. One of the advantages in the bottom up integration is that stubs are useless and therefore, no need to have them any more. The upcoming steps show the guideline for the bottom up integration. [20]

First Components in the lower level which perform a subfunction, compose a cluster.

Second A control program which is called test driver is developed for coordinating input and output data.

Third The created cluster is tested by the written driver.

Forth And finally, the written driver is removed and new cluster will be composed upward.

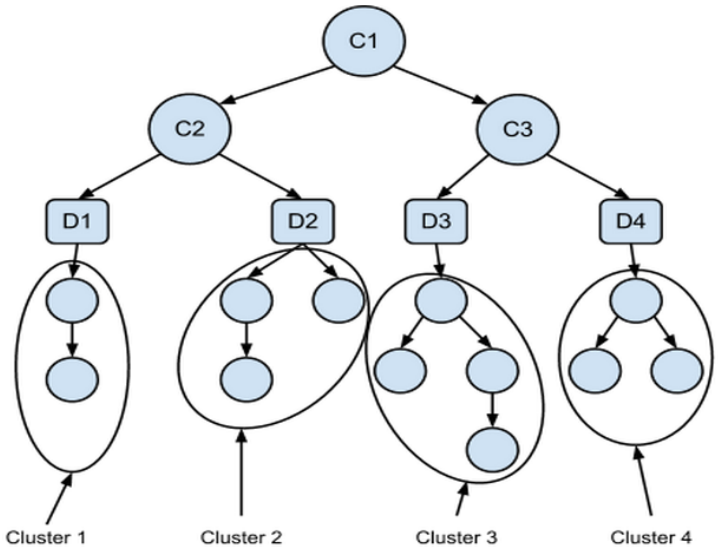

Figure 2.6: Bottom-up Integration

The steps will be executed until the whole structure be integrated. Referring to the figure 2.6, modules with similar interests form clusters.

Each cluster will be tested by a driver. For example in the figure 2.6, clusters 1 and 2 are tested by drivers D1 and D2 respectively. Then, drivers D1 and D2 will be removed and the clusters will be connected to the C2. The same manner happen for the cluster 3 and cluster 4. And finally C2 and C3 will be merged with C1. Although, each cluster requires it own driver to be tested, moving upward and integrating forward mitigate need of new drivers. As it stated in the [20], "if the top two levels of program structure are integrated top down, the number of drivers can be reduced substantially and integration of clusters is greatly simplified."

2.4.5.3 Regression Testing

Adding every new component to the software for testing can lead to new control paths, new control logic or new input/output. Previously tested functions can be affected by such new changes in the software. Regression testing ensures that the integration of new module does not cause new functionality violations. In fact, regression testing assists testers to uncover new errors. Furthermore, this strategy is really useful and important in order to reduce side effects that occur by changes. In the other hand, as it discussed, the main goal of testing is detection of defects in the software, but these detected bugs have to be fixed. This process can cause new errors(because of new changes while correction occurs) in the software. Another benefit of regression testing is to certify that unexpected error or functionality will not happen after such modification in the the software. Regression testing can verify whether the new components affect on the previously integrated components and violate predefined specifications.[19, 20, 36]

This testing can be performed by running a subset of the previously executed tests and as it mentioned in the [19], using some capture and playback tool which captures test cases and their results for comparing these results with outcome of next playback.

Three different types of test cases are in the regression [20]:

1. A delegate test which can examine all functions in the software.
2. Tests that examine functions which are more probable to be failed by

recent changes.

3. Tests which examine those components which have been modified.

Following lines contain the regression test process which is stated in the [37].P refers to a program or component, P' refers to the modified version of P and T, T' refer to test suites for P and P' respectively. In addition, T'' refers to the new test suite that should be derived to test P' when it is necessary.

1. Select $T' \subseteq T$, T is a subset of test cases for execution on P'.
2. Test P' with T' and verifying correctness of the P' with respect to testing with T'.
3. Create T'' which contains new functional or structural test cases in order to test P' when it is required.
4. Test P' with T'' and verifying correctness of the P' with respect to testing with T''.
5. Create a new test suite,T''', for testing P' from T, T' and T''.

It is not proper testing way to re-execute all test cases for all software functions after every change. The tester should design regression tests in a way to reveal one or more error classes in every major functions with intent of reducing number of regression tests. All these attempts for reducing the number of test are because of high cost of regression tests. These attempts are resulted to several regression test selection (RTS) techniques. Five regression test selection techniques for reusing written test case are experimented and compared in the[37] by focusing in the capability to reduce regression testing cost and disclose new errors.

2.4.5.4 Smoke Testing

Smoke testing also known as build verification testing, is one of the integration testing approach and applies to time critical projects. It contains a non-redundant set of tests in order to test the most important and critical functionalities in the software frequently. Integrated software components

in addition to required libraries, files and modules encompasses a build. Then a set of non-exhaustive tests with intent of revealing defects in functions which are so important more probable to fall the project behind the schedule, are designed and conducted. And finally a daily smoke test will be run on the whole product that includes all integrated builds. As it mention by Jim McCarthy in the[38]: "Treat the daily build as the heartbeat of the project. If there is no heartbeat, the project is dead.", daily smoke test on daily build is so important in all software projects. The outcome of smoke testing ensures that this build is stable enough to carry on other tests. In should be noticed that smoke test should cover most critical function but not in depth. In addition daily smoke test assists developers to reveal errors and integration problems early in the development and testing cycles, therefore, it reduce the integration risks which can lead to delay in the schedule. Furthermore, smoke tests is capable of finding both functional and component design errors. It is simply because of its integration oriented structure. Hence, the result will be better quality product. Because of its daily result, smoke test can help developers and also managers to keep track of progress.

Both regression and smoke testing should concentrate on the critical module in the software. A critical module is a module which represent several program requirements, tends to be failed and has predefined performance requirements.[20]

2.4.5.5 Top-down vs. Bottom-up

Obviously, both bottom-up and top-down integration testing have their own advantages and drawbacks. The need of stubs which lead to difficulties in the testing and integration, is one of the most important drawbacks in the top-down aspect. But, revealing errors in the major control function as early as possible(top down integration nature) is one of its benefits. In the other hand, as it mentioned in the [19], existence of a program depends on the integration of last module to the program, therefore, because of the bottom-up integration nature, it is considered as its disadvantage. But, main benefit of bottom-up integration is that it does not require stubs, so, designing test cases will be simpler. Generally, software developer will chose combination of these two approaches which is called sandwich testing. In the sandwich testing, top-down integration

testing is applied to the higher level components and bottom-down integration is applied to the lower level components.[20]

It should be noticed that following tests should be designed and derived in the integration testing:[20]

Interface integrity test.
After each integration, both interfaces(internal and external) should be tested.

Functional validity test
Executing of tests with intent of detecting functional defects.

Information content test
Global or local data structure errors should be uncovered.

Performance test
performance bounds should be tested in order to be verified.

Integration test documentation also known as test specification which includes test plan, test strategy, test procedure, test schedule, previous test result, testing problems and so on, helps testers to perform proper maintenance whenever it is needed. In fact, better test specification leads to better maintenance and better quality product.

2.4.6 System Testing

System testing is another level of software testing and is done after complete integration of components and their tests. When all components are examined by unit testing and then integrated and tested, software should be tested by system testing with intent of examining whole system with its elements such as hardware, software and so on. The goal is to verify proper integration and requested functionalities.

Some of the worthwhile testing in the software testing is described in the[39] and will be explained shortly in the upcoming lines.

Stress testing.
The main aim in the stress testing is to put system in the abnormal

condition to find its failure threshold. It means that the tester tries to overload system with high quantity, frequency and volume of resources.

Performance testing.
This testing is being done to test the run time performance of the system. In fact, it is executed with with participation of stress testing to calculate utilization of resources. Normally, external instrument will be used to disclose conditions of system failures.

Recovery testing.
Recover testing is executed to test how system is fault tolerant. It means, the system will be examined to see if the system can recover from fault and resume its functionality. In this testing, testers force the system to fail and verify its recovery performance.

Security testing, installation testing, load testing, sanity testing and so on are other system testings which will not be discussed in this report.

2.4.7 Debugging

Testing process is always coupled with debugging, As it mentioned, tests are being done to reveal errors. These errors should be fixed. Repairing the errors are consider as debugging process. In the debugging process, testers try to find the source of the error by analysing the results and then fix it.

Chapter 3

Approach

This chapter will guide you through the testing systems, data collection methods and data analysis procedures used in this thesis. The following lines include an overview of the testing strategy, white box test and black box test environments which will be called WBTest and BBTest respectively from now on. The chapter will be followed by the explanation of data collection methods used in this thesis such as quantitative, qualitative and their combination. It will be ended by introducing a data analysis method for multi criteria decision making that is going to be used in the analysis chapter.

3.1 Testbed

In this section, the testing strategy used in the company will be described briefly. It contains all testing activities but since possible overlaps may occur on sub-system level, the focus will be on this level. This section will be followed by the WBTest and BBTest environments that are used in sub-system level testing.

3.1.1 Testing Strategy

This part contains a hardware generic and standard independent strategy for radio software development in the company based on internal documents in the company as well as the international software testing qualification board (ISTQB). [40, 41] Furthermore, test activities and levels within the radio organization will be described.

The main aim in software testing as mentioned in the background chapter, is to find faults as early as possible to ensure efficiency and fair cost of testing. Testing strategy provides a common view of the quality assurance and outline the quality assurance requirements set on the radio base station (RBS) features (feature is smallest saleable unit, i.e. software and/or hardware that adds customer value.) and radio software. It also supplies the daily work in the cross functional teams (XFT teams) which are responsible for radio software development and for testing properly.

General principle that should be noticed in this strategy are:

- The quality level should be as expected by end customers.
- Defects should be found and corrected as early as possible (to shorten feedback loops).
- Test-driven development recommended on all levels.
- Test automation should be the first choice when writing test cases.
- Requirements should be possible to test
- The test analysis should cover scope, costs, needed resources and risks.

The test strategy for radio software is specified in terms of goals and requirements for the software development. It includes all responsibilities supporting the XFT teams to develop high quality software. The strategy covers design level test of radio software, radio sub-system level test and node level test on multi-standard radio base station (RBS). [Figure3.1]

Design level test includes static, component test (CT), component integration test (CIT).[Figure3.1] In this level, testing activities should be done

from specification perspective which refers to white box testing. It means that design specification (DS) is the base of software development, therefore design test should also be based on DS. The test activities in the "static" are test review and static analysis. Test review represents reviewing the targets(design input specification, radio unit (RU) application and test code). Static analysis refers to compilation of all targets and additional static analysis by aid of coverity analysis tools. The important point in this level is that compilation and static analysis must be passed without error and warnings. After "static" testing activities, component test (CT) also known as unit test will be applied.

It is a test with the intent of finding bulk of implementation faults. It focuses on high code coverage: if 100% statement coverage happened, the test is passed and If less than 100% coverage occurred, the test is failed. As stated earlier, in the general principle, test driven development is recommended.The next activity in the design level is component integration test. In this activity all dependencies and impact of component should be extracted and tests should be designed to find faults in the interaction between components.

As can be seen in the figure3.1, design level test will be followed by sub-system level test which contains sub-system integration (SSI) and sub-system verification (SSV) testings. Overlaps and conflicts have higher likelihood to happen in this level, since this level is the junction of white box and black box testing methods. Based on testing strategy, sub-system integration and sub-system verification should be done in WBTest environment and BBTest environment respectively and in some cases these goals would be mixed because of the existing potential of overlap in the sub-system level test.

In sub-system integration testing of radio unit software, a simulator which simulates different radio units with different standards, can be used for pre delivery debugging and test case implementations before running them on real radio unit (RU) for execution of tests. Then, tests will be run in the lab which includes a wide selection of HW to cover all platforms, families , driver sets, frequency bands and so forth. In addition, test objects should mainly be mapped on RU functionality according to functional and nonfunctional (performance, robustness and etc.) aspects.

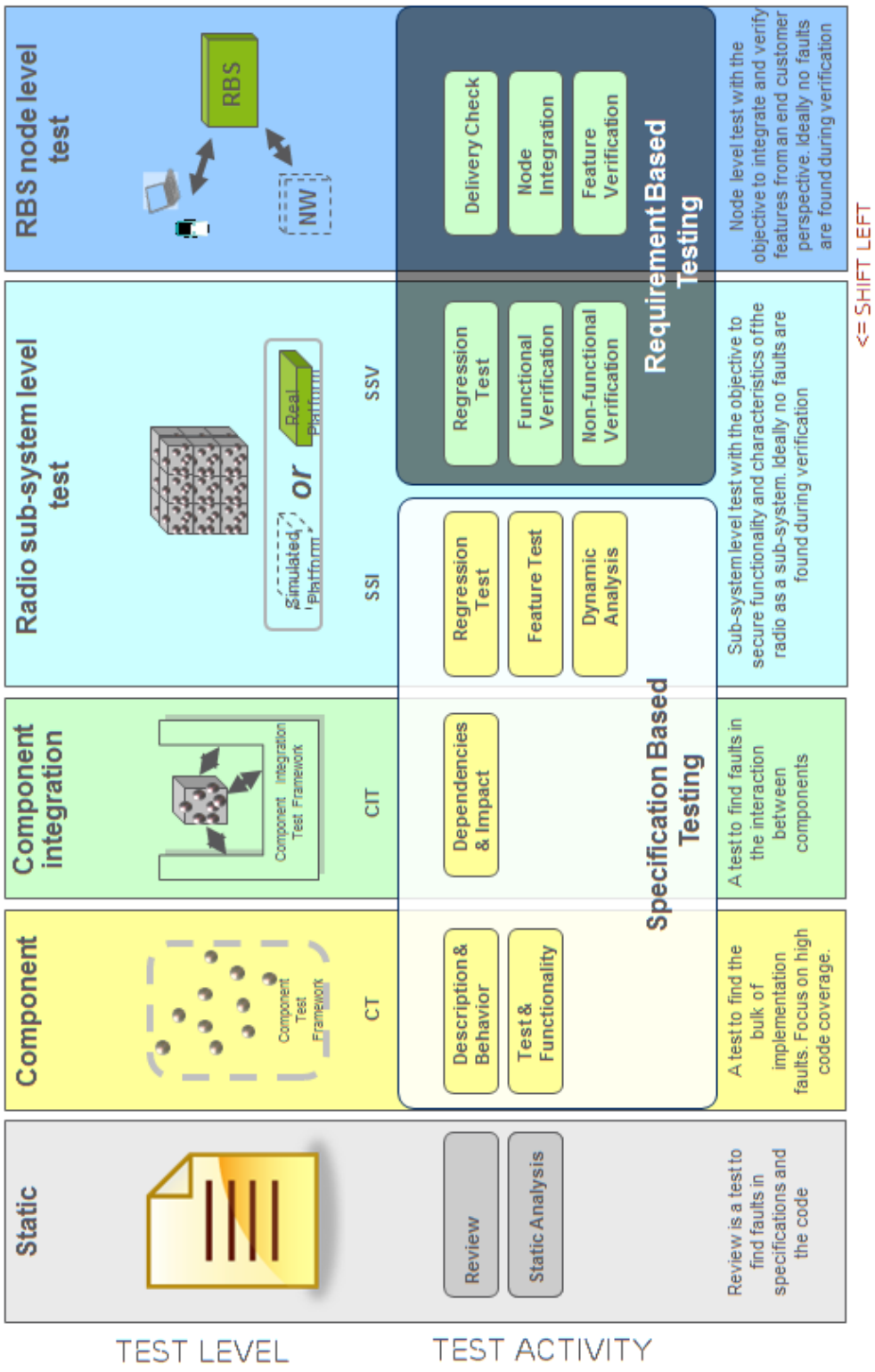

Figure 3.1: Test activities for radio software development printed with permission from company's internal document.[42]

Integration test cases shall be prioritized between each other. Prio 1: Test cases that cover functionality that have impact on traffic and/or is a significant problem for further testing or customers. Prio 2: Test cases that cover everything but prio 1 above. There shall be flexible automated test suites along with pre-defined suites "smoke" and "regression" with contents decided by each test object [Figure3.2] :

- Smoke test: 5% of all test cases, only prio 1, speed is of the essence here.
- Regression test: 30% of all test cases with the rule of thumb to achieve 90% function coverage.

In the sub-system verification, sub-system functional and nonfunctional requirements will be verified based on predefined specification. As mentioned, testing for sub-system verification should be done from a black box perspective. In addition, the tests are performed on a radio unit using a simulated digital unit and an antenna near the unit. All radio sub-system functional and nonfunctional requirements should be covered by test cases (100% requirements coverage).

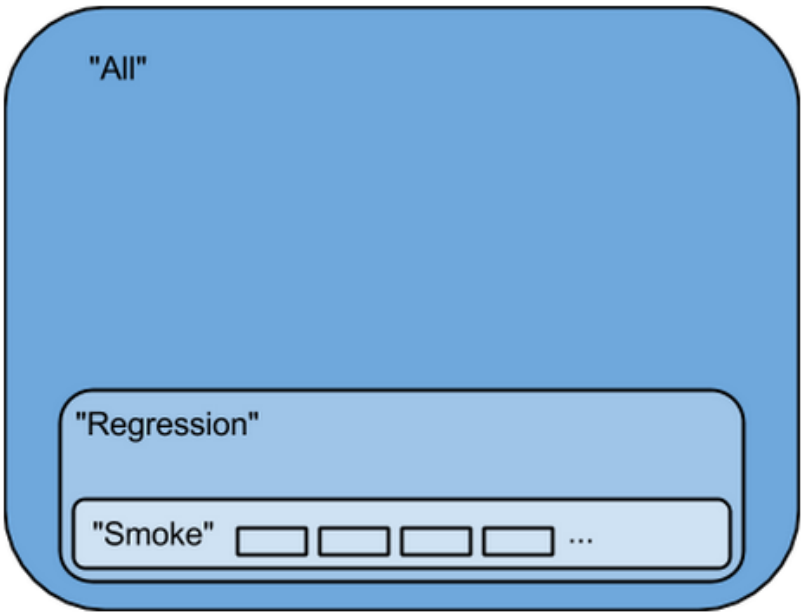

Figure 3.2: Pre-defined test suites schema.

The next level of testing activity is called the node level test. Major

emphasis is put on finding errors that are difficult to predict in addition to faults that allow the system to be perceived as unstable, e.g. real-time effects and interference between different parts of the system which may cause hanging resources, degradation of services, intermittent alarm, and system downtime. The work is based on operational procedures, including:

- Planning, preparation and deployment of the test configuration.
- Execution of test cases and monitoring of the multi standard radio base stations (RBS) as a black box.
- Using built in observability mechanisms.

The node level test covers the following scopes: delivery check, node integration, feature verification. Delivery check aims to verify software before delivery for release verification by executing a small set of automated single mode and mixed mode tests within the legacy scope on radio base station node level. Node integration intention is to provide test configuration ready for feature verification and also integration of radio unit and digital unit softwares before delivery. Feature verification has to be done in order to secure customer quality expectations for new features including support for new or modified hardware.

3.1.2 WBTest Environment

The WBTest environment is based on Ruby[43] and a behaviour driven development (BDD) tool called RSpec[44]. It offers the possibility to do test driven development (TDD) and specification-based testing for the radio unit (RU) functional parts. With the WBTest environment, it is encouraged to perform test-driven development. It means to write the test cases before the implementation, run them regularly and see them go from fail to pass. The WBTest environment architecture, how it connects to simulated digital unit(SDU) through the gateway and then radio unit in order to test radio unit software, can be seen in the figure 3.3. CPRI[45]refers to common public radio interface. It is the industry cooperation defining the publicly available specification for the key internal interface of radio base stations

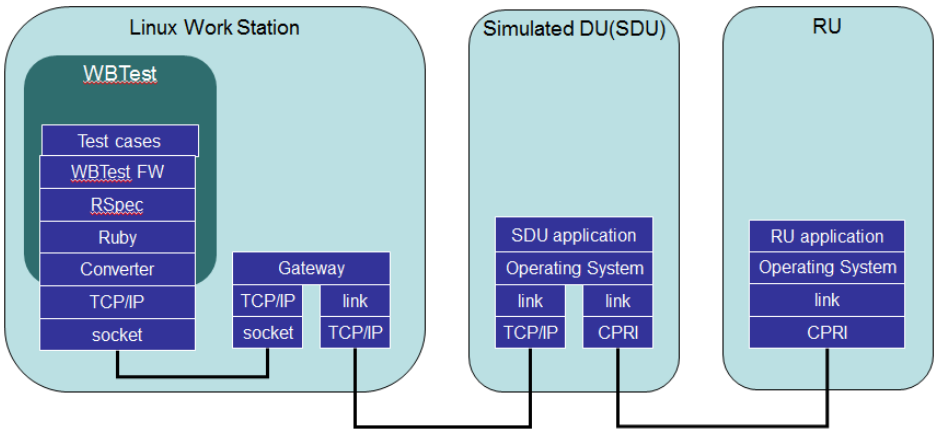

Figure 3.3: WBTest environment architecture.

between the Radio Equipment Control (REC) and the Radio Equipment (RE).

Test cases are grouped in so called test objects where each test object has the scope of covering one area of radio unit functionality, often specified by a design specification (DS). Test cases and test objects can be traced back to the design input thus being specification-based. WBTest is intended for integration testing on target. The combination of unit test on host, WBTest and functional test on target at the functional integration departments makes a good solution for RU functional quality assurance. The test strategy should strive to coordinate "what covered where", to make efficient testing. Each test case can be tagged for various purposes e.g. test objectives such as smoke, regression and so on. It is the responsibility of each test object to select a test objective for each test case.Test execution can be configured so that the sum of all test cases being executed, i.e. test suite, is the sum of all test cases matching tag(s). Thus it will be possible to select test cases to create test suites in a very flexible way. A test object should cover all requirements from design input specifications that are currently possible to test in the integration test environment.

WBTest tool has the following specifications:

- Facilitate integration and low level tests for radio software designers.
- High level of productivity for writing test cases.

- Quick turnaround time from test code changes testing to test execution (< 10s) to support agile workflow such as TDD.
- Easy to run in lab and support for running regression test in the lab.
- Easy to build abstractions.
- Easy to do complex tests.
- Access to external and internal interfaces via radio unit operating system signaling and shell commands.
- Tests are based on DS's or lower level documents.

Although WBTest tool focuses on white box perspective by existing supports testing of internal functionality (white box testing), it support testing external interfaces for black box aspect. The WBTest tool has methods to send and receive signals, methods for radio unit (RU) console commands and matching of received and expected signal data including default values, ranges and wildcards.

3.1.3 BBTest Environment

BBTest core is written completely in Perl[46], has been used since 2007 and runs on on Solaris and Linux machines. It uses an XML database and has a GUI to prepare and run test scripts and sequences. The BBTest can be run automatically (nightly tests, continuous verification). In addition, it runs test scripts written in the Perl language and one script equals one test case defined in the verification specification. Moreover, it generates log files for each test script. The BBTest environment performs functional verification in sub-system level. It means the BBTest environment verifies the release software on real hardware in addition to verification of hardware and software together on black box level. In the other hand, it is an examination aiming to determine whether the requirements have been fulfilled or not.

Figure 3.4 represents how BBTest core is connected to digital unit (DU) or Simulated digital unit (SDU) and also to other test equipments which includes signal generator, spectrum analyser and other measurement devices. The main focus on this environment is testing with black box perspective and mainly on radio unit software interfaces. Simply,

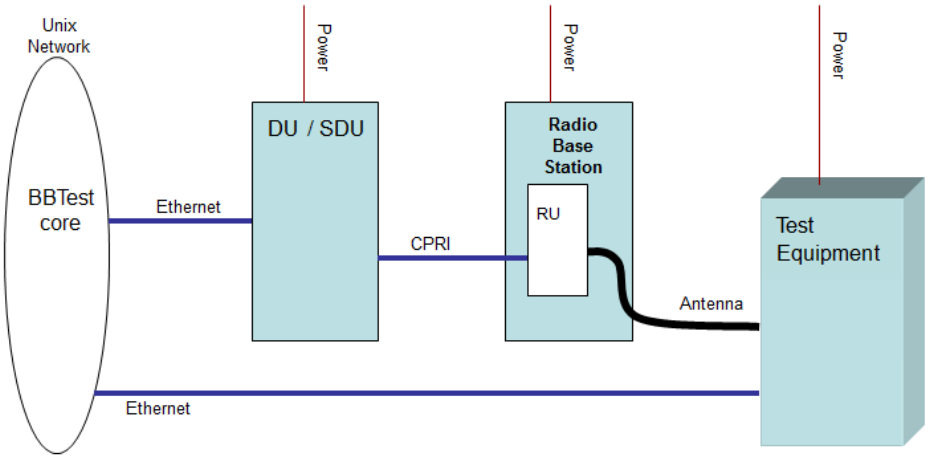

Figure 3.4: WBTest environment architecture.

from black box point of view the radio is a "black box" and the BBTest environment is not aware of internal functions and tests radio unit software on its interfaces. It also uses test equipment to measure input and output values on the radio unit interfaces.

This environment has been designed to fulfill requirements which are defined in functional specification (FS). For example, a signal as input will be generated on an interface and expected output value or range on the interface have to be obtained. This process is written in functional specification that the test case is designed based on. The values and parameters will be compared and checked against the radio unit (RU) software XML database (it is filled based on functional and verification specification documents). These comparison results either pass or fail test objects. Furthermore, in case of needing real measurements, BBTest would use test equipments.

3.2 Data Collection Methods

This section goes through the research methods used to investigate and compare the two testing environments (called WBTest and BBTest) which are explained briefly in the previous section. There are three different research methods: Quantitative, Qualitative and Mixed methods.

Choosing proper research methods and designing the research depends on the nature of the problem which is going to be investigated (Research problem), experience level of the researcher (Personal experience) and finally whom is interested in ongoing study(the audience).[47]

Regarding the research problem, the quantitative is the proper choice, if the problem needs to address identifying the parameters that affects the result or for understanding the best result prediction. Quantitative approach can be used to test a theory or explanation. Conversely, in case of a problem or research field that little investigation has been done on so far, qualitative research can be applied. The qualitative method is more exploratory method to find important variables that affect the topic. This approach is useful for researchers who face new issues.[47, 48] If either the quantitative approach or the qualitative approach is insufficient to understand the research problem, the mixed approach could be a suitable aid. In such cases, the researcher can first investigate which parameters or variables should be studied (qualitative) and then expand the research on larger number of samples (quantitative) or vice versa, the first inquiry on large number of samples and then continue it with a few participants.

The personal experience is another criteria that can affect choosing the proper research approach. The experiences regarding these three approaches may lead to choosing either the quantitative, qualitative or the mixed approach. Those who are experienced in statistics and computer statistic programs or scientific writing prefer quantitative method for their study. On the other hand, those with experiences in interviews and writing in literary method would select qualitative method. And finally, those who have enough time and resources in addition to being familiar with both approaches would choose mixed mode. Obviously, choosing mixed mode need extra work and effort, since both quantitative and qualitative data should be collected and analysed. On the other hand, the inquirer can profit from both quantitative and qualitative specifications, even though it is harder to be carried out. At last, the audience of the research can shape which method should be chosen. The researcher should know for whom the survey is going to be presented then he/she chooses proper approach upon it. [47]

Distinction between these two can be shaped in terms of applying numbers (quantitative) rather than words (qualitative) or applying open-ended questions (qualitative) rather than close-ended questions (quantitative).

[47] In addition, quantitative and qualitative aspects are not like polar opposites. [49]

A researcher may gather data by a set of predefined questions or interviews. She/He might conduct interviews without a set of predefined question and may let interviewee talk about the topic freely. Selecting the method depends on the intent of researchers, whether they would rather to specify the type of information that is going to be gathered in advance or they prefer to let participants inject their opinions into the research. On the other hand, type of data going to be analysed is also effective in this choice. Either numeric information by conducting quantitative research or text data by performing qualitative method can be collected. After collecting data, inquirer may search to find patterns out of emerged data from participants or carry out statistical analysis over numeric data. In some cases, both quantitative and qualitative methods are being used to gather, interpret and analyse. In mixed method, researcher will use both quantitative and qualitative databases which contain correspondent data. Table 3.1 represents qualitative, quantitative and mixed methods approaches and their practices.[47]

As it can be seen, mixed mode which is located between quantitative and qualitative approaches in the table 3.1, is a mixture of these two approaches and contain most of their specification and practices. Since the mixed approach is chosen for data collection in this thesis, it will be explained more in the upcoming lines before illustrating the overall research plan. In fact, the nature of the problem that will be faced in this thesis is the main reason for choosing the mixed approach. The comparison between two environments (WBTest and BBTest) which are going to be under assessment for an external researcher needs internal investigation. In fact, those who are involved in their daily job with these environments know more about pros and cons. They can provide better information for researcher. The mixture of quantitative and qualitative has been used in order to use data from one to complement another one and reach to a fair judgment on these two environments as well as check their integration feasibility.

	Qualitative	Mixed	Quantitative
Methods	Open-ended questions Emerging methods Interview data Observation data Document data, Audio-visual data Test and image analysis Pattern interpretation	Both open- and close-ended questions Both predetermined and emerging methods Multiple form of data drawing all possibilities Statistical and text analysis Across database interpretation	Close-ended questions Predetermined methods Performance data Numeric data Census data Statistical analysis Statistical interpretation
Practices	Collect participant meanings Focuses on a single concept Bring personal values into the study Validate the accuracy of findings Makes interpretation of the data Create agenda for change or reform Collaborate with the participants Focuses on a single concept	Collect both quantitative and qualitative data Develop a rationale for mixing Integrate the data at different stages of inquiry Present visual picture of the procedure in the study e Employ practices of both qualitative and quantitative research	Test theories or explanations Identifies variables to study Relates variables in questions or hypotheses Use standards of validity and reliability Observe and measure information numerically Use unbiased approaches Employ statistical procedure

Table 3.1: Qualitative, Quantitative and mixed methods approaches. Adapted from [47].

There are three main strategies called sequential mixed methods, concurrent mixed methods and transformative mixed methods. The sequential mixed methods refer to methods which researcher expand the results of one method (quantitative or qualitative) with the another one. This strategy may begin with quantitative survey in which a theory or concept is tested and is carried on by qualitative interviews which provide more detail information about the case under study. Alternatively, It can begin with some qualitative interviews and then ended up with quantitative questionnaire to generalize the result. The second main strategy is concurrent mixed methods which indicate collecting, merging and analysing both quantitative and qualitative data concurrently to achieve a comprehensive analysis. The both form of data will be collected at the same time and then merged for interpretation of result. In this from, researcher may embed one form of the result to another one in order to answer to raised questions. Finally, third mixed methods are called transformative mixed methods that represent the procedure which researcher uses an overall perspective to investigate the research problem. In this case, data collection can be either sequential or concurrent. This perspective can also contain outcomes expected from the study.[47]

One of the considerable points in the mixed method is "mixing the data". Mixing the quantitative and qualitative data is tiresome, since type of collected data is different (numeral for quantitative and text or images for qualitative). This mixing may happen in the different phases of research (Data collection, data analysis and data interpretation) or maybe at all of these three phases. In addition, circumstance of mixing divides it to the three types: connected, integrated and embedded. Connected type of mixing means that data gathered in quantitative and qualitative methods is connected to each other between analysis of the first phase(quantitative or qualitative) and collection of second phase. In fact the outcome of first phase is used to distinguish parameters of second phase of research. Second type of study is called integrated. In this scenario, quantitative and qualitative data may be collected concurrently and then researcher integrates them by comparing data.Transforming the data type should happen before comparing them. It mean, they should be in comparable format. The embedded mixing happens when researchers have initial intent to gather either quantitative or qualitative form of data and then provide another form of data to support first one. [47]

3.3 Data Analysis Method

Analytical Hierarchy Process (AHP) which is created by Thomas L.Saaty in [50] has been chosen to in order to combine data that is gathered with both quantitative and qualitative in addition to result analysis of selected research method for this thesis. Analytical Hierarchy Process is a method for multi criteria decision making (MCDM). It put multiple criteria in hierarchy structure, use pairwise comparison by relative importance of them and rank the alternative based on these comparison in the hierarchy structure.[51, 52] The Analytical Hierarchy Process has been applied for supporting the decision that made in the information system management [53]as well as many other different fields that can be found in [54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67]. Because of its pleasant mathematical specification, the analytical hierarchy process method become more popular in many research fields and among engineers and scientist. The AHP can be great assist in order to capture both objective and subjective measures. It has a useful mechanism for checking the consistency of measurements over criteria and alternatives that this checking mitigate the bias in decision making in organizations.[52]

The AHP is based on sets of pairwise comparisons to obtain relative weights of individual criterion and rating alternatives with respect to each criterion. Since it may reverse the final ranking in case of similar alternatives, the AHP model was stated unstable. Bolton and Gear introduced the Ideal Mode AHP in order to solve this problem.[68] This new optimized model was accepted by Saaty in the [69] and considered as most reputable Multi Criteria Decision Making (MCDM) model. It should be mentioned that the ideal mode AHP will be used at the analysis chapter of the thesis.

T. L. Saaty divided the Analytical Hierarchy Process in following four steps in order to apply an organized way for making proper decision:

1. "Define the problem and determine the kind of knowledge sought.
2. Structure the decision hierarchy from the top with the goal of the decision, then the objectives from a broad perspective, through the intermediate levels (criteria on which subsequent elements depend) to the lowest level (which usually is a set of the alternatives).

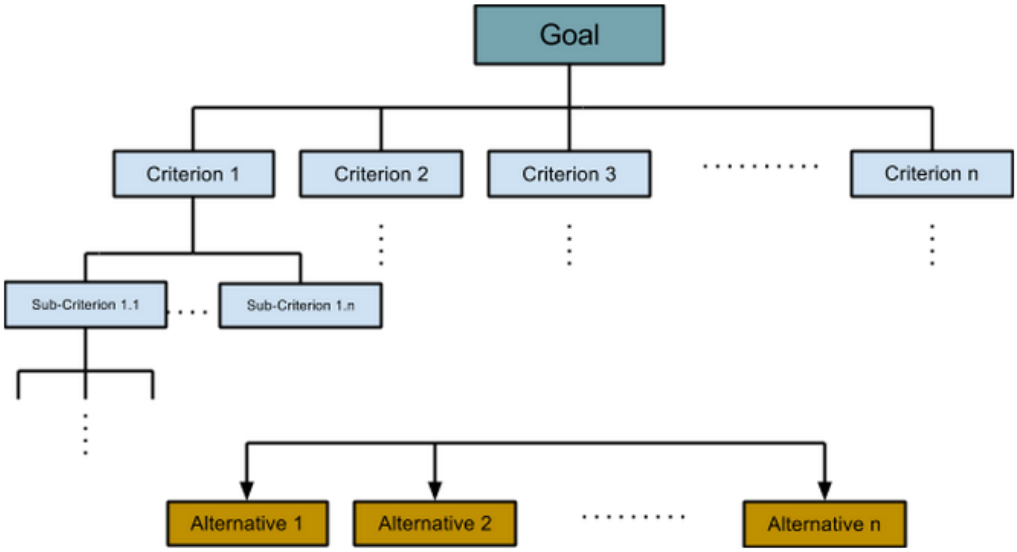

Figure 3.5: Structure of analytical hierarchy process.

3. Construct a set of pairwise comparison matrices. Each element in an upper level is used to compare the elements in the level immediately below with respect to it.

4. Use the priorities obtained from the comparisons to weigh the priorities in the level immediately below. Do this for every element. Then for each element in the level below add its weighted values and obtain its overall or global priority. Continue this process of weighing and adding until the final priorities of the alternatives in the bottom most level are obtained."[5]

Figure 3.5 shows the structure of the Analytical Hierarchy Process and how problem is decomposed in a hierarchy of "criteria" in the middle of structure and "alternatives" at leafage of the structure. This structure refers to the second step in Saaty's four steps.

After creating such hierarchy, pairwise comparisons as mentioned in Saaty's third step will be performed. The result of these comparisons with respect to the main goal in this decision making is based on table 3.2 which is called Saaty's pairwise comparison table. Table 3.2 represents which value should be inserted into the corresponding matrix.

Value	Definition	Meaning
1	Equal	Two activities contribute equally to the objective
2	Weak or slight preference	
3	Moderate preference	Experience and judgement slightly favour one activity over another
4	Moderate plus preference	
5	Strong or essential preference	Experience and judgement strongly favour one activity over another
6	Strong plus preference	
7	Demonstrated preference	An activity is favoured very strongly over another; its dominance demonstrated in practice
8	Very, very strong preference	
9	Absolute and extreme preference	The evidence favouring one activity over another is of the highest possible order of affirmation
1/i	Reciprocals of above values	If criteria i has one of the above non zero numbers assigned to it when compared with activity j, then j has reciprocal value when compared with i.
1.1-1.9	If the activities are very close	It is hard to choose best value. But in some cases the size of small values has considerable effect

Table 3.2: Saaty's pairwise comparison table.[5]

In order to understand Saaty's third step, assume an example with three different criteria (C1, C2 and C3) and three alternatives or options (A1, A2 and A3). A matrix for criteria will be created which its elements reflect the Saaty's pairwise comparison table (Table 3.2). This matrix which known as judgment matrix or criterion matrix is defined as follow:

$r_{ij} = \{9 \geq r_{ij} \geq 1/9 \mid i, j = 1, 2...n\}$ that $r_{ji} = 1/r_{ij}$ when $i = j$ then $r_{ij} = 1$

$$\text{Pairwise Comparison Matrix of Criteria} = \begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \end{array} \begin{array}{c} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} \\ \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \end{array}$$

As it can be seen in the criteria matrix which is built for pairwise comparison of three different criteria in this example, element $r12$ indicates the rating of C1 compared to C2 and is the answer to the question: "How important is the C1 in comparison to the C2?", The response can be extracted from the Saaty's pairwise comparison table. [Table 3.2]

Similar matrix should be created for alternative, but with respect to each criterion. For the mentioned example, Three matrices each contains pairwise comparison of three alternatives based on specific criterion should be created. The following matrix is a sample of such matrices:

$$\text{Alternatives matrix against criterion } C_1 = \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \end{array} \begin{array}{c} \begin{array}{ccc} A_1 & A_2 & A_3 \end{array} \\ \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \end{array}$$

The elements of this matrix should also be extracted from Saaty's pairwise comparison. In this matrix, all alternatives(A1,A2 and A3) will be compared against criterion C1. Similar matrices should be created for criterion C2 and criterion C3 as well.

Before starting the process (Saaty's fourth step) which lead to decision matrix by calculating priorities and weights, consistency ratio (CR) can be assessed for each of mentioned matrices (Four matrices for the mentioned example, one for the criteria and three for alternatives with respect to each criterion.) Consistency ratio calculation is one of the advantages in the AHP. As it stated in the[70] by Saaty, if CR is less than 0.1 or in the other word less than 10%, it is considered as adequate and the matrix is acceptable. Otherwise it should be recreate by fixing its rating issues. In order to calculate Consistency ratio (CR), the Consistency Index (CI) should be computed by the following formula:

$$CI = (\lambda_{max} - n)/(n-1)$$

which n refers to number of alternatives or criteria being compared and approximation of maximum eigenvalue is denoted by λ_{max}. After calculating Consistency Index (CI), using following formula will end to the Consistency Ratio: $CR = CI/RCI$ where RCI can be obtained from the Random Consistency Index (RCI) table that is given by Saaty [5] for different values of n as it can be seen in the table 3.3.

n	1	2	3	4	5	6	7	8	9
RCI	0	0	0.58	0.90	1.12	1.24	1.32	1.41	1.45

Table 3.3: Saaty's Random Consistency Index (RCI) table.[5]

As it declared in the fourth step, the process will be followed by calculating the relative weighting (priority) as known as weighting or priority vector of matrices to obtain original AHP decision matrix. For calculating priority vector, geometric mean of each matrix should be calculated. Following vector represents geometric mean that can be obtained from each matrix:

$$M = (m_1, m_2, ..., m_n)^T = \left\{ \sqrt[n]{\prod_{j=1}^{n} r_{1j}} \, , \, \sqrt[n]{\prod_{j=1}^{n} r_{2j}} \, , \, ... \, , \, \sqrt[n]{\prod_{j=1}^{n} r_{nj}} \right\}$$

The relative weighting or priority will be calculated by normalizing

geometric mean vector. Normalization will be achieved with dividing elements of this vector by sum of the geometric mean represented with following standardized priority vector: [71]

$$P = (p_{m1}, p_{m2}, ..., p_{mn})^T = \left\{ \frac{\sqrt[n]{\prod_{j=1}^{n} r_{1j}}}{\sum_{i=1}^{n} \sqrt[n]{\prod_{j=1}^{n} r_{ij}}} , \frac{\sqrt[n]{\prod_{j=1}^{n} r_{2j}}}{\sum_{i=1}^{n} \sqrt[n]{\prod_{j=1}^{n} r_{ij}}} , ... , \frac{\sqrt[n]{\prod_{j=1}^{n} r_{nj}}}{\sum_{i=1}^{n} \sqrt[n]{\prod_{j=1}^{n} r_{ij}}} \right\}$$

This priority vector should be calculated for all mentioned matrices. The next matrices are the summary of the mentioned calculations for the criteria (C1, C2 and C3) in the example:

$$\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \end{array} \begin{array}{c} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} \\ \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \end{array} \Rightarrow \begin{array}{c} M_{criterion} \\ \begin{bmatrix} m_{c1} \\ m_{c2} \\ m_{c3} \end{bmatrix} \end{array} \Rightarrow \begin{array}{c} P_{criterion} \\ \begin{bmatrix} p_{c1} \\ p_{c2} \\ p_{c3} \end{bmatrix} \end{array}$$

And three other matrices for alternatives follow the upcoming schema which X refers to the criterion which alternatives comparison carried out against (In the example: C1, C2 and C3).

$$\begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \end{array} \begin{array}{c} \begin{array}{ccc} A_1 & A_2 & A_3 \end{array} \\ \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \end{array} \Rightarrow \begin{array}{c} M_{alternatives} \\ \begin{bmatrix} m_{c_X a1} \\ m_{c_X a2} \\ m_{c_X a3} \end{bmatrix} \end{array} \Rightarrow \begin{array}{c} P_{alternatives} \\ \begin{bmatrix} p_{c_X a1} \\ p_{c_X a2} \\ p_{c_X a3} \end{bmatrix} \end{array}$$

The calculation of all priority matrices lead to the original analytical process

decision matrix that includes all computed priorities for each alternative in addition to priority vector of criteria.This outcome can be simplified in two matrices as follow:

$$M_{PofA} = \begin{array}{c} \\ A_1 \\ A_2 \\ A_3 \end{array} \begin{array}{c} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} \\ \begin{bmatrix} p_{c_1a_1} & p_{c_2a_1} & p_{c_3a_1} \\ p_{c_1a_2} & p_{c_2a_2} & p_{c_3a_2} \\ p_{c_1a_3} & p_{c_2a_3} & p_{c_3a_3} \end{bmatrix} \end{array} \quad M_{PofC} = \begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \end{array} \begin{bmatrix} p_{c1} \\ p_{c2} \\ p_{c3} \end{bmatrix}$$

The ideal mode analytical process (AHP) is concluded from this original AHP matrix by dividing each element of M_{PofA} by maximum element of its column as it shown in the following formula for element P_{c1a2} in the M_{PofA} for the mentioned example:

$$I_{c_1a_2} = P_{c_1a_1} / Maximum(P_{c_1a_1}, P_{c_1a_2}, P_{c_1a_3})$$

And finally ideal mode AHP relative priority will be concluded by multiplying the idealized matrix of priority for alternatives to the priority vector of criterion as it shown below for the mentioned example:

$$\begin{bmatrix} I_{c_1a_1} & I_{c_2a_1} & I_{c_3a_1} \\ I_{c_1a_2} & I_{c_2a_2} & I_{c_3a_2} \\ I_{c_1a_3} & I_{c_2a_3} & I_{c_3a_3} \end{bmatrix} \times \begin{bmatrix} p_{c1} \\ p_{c2} \\ p_{c3} \end{bmatrix} = \begin{bmatrix} p_{A1} \\ p_{A2} \\ p_{A3} \end{bmatrix}$$

As conclusion of AHP decision making method, the following formula generalizes the final priority vector which is shown in the previous

multiplication of matrices. The element with the highest value is most prefered and is the best choice among alternatives and the other alternatives should be sorted descendingly.

$$P_{Alternatives} = (P_{A_1}, P_{A_2}, ..., P_{A_n})^T = (\sum_{i=1}^{n}(I_{c_i a_1} \times P_{c_i}), \sum_{i=1}^{n}(I_{c_i a_2} \times P_{c_i}), ..., \sum_{i=1}^{n}(I_{c_i a_n} \times P_{c_i}))$$

3.4 Research Plan

Since current thesis is going to be done in a company where all properties related to the study are unknown for external thesis worker, it is decomposed into three stages. The first stage contains exploring the internal documents in order to gather basic information, plot questionnaire with respect to basic information as well as couple of unstructured interviews and then second complement questionnaire. The second stage includes interviews with those who are expert in the environments under study as qualitative part of survey in addition to further investigation about these two testing environments(The WBTest and BBTest environments). Third stage represents the summarization and analysis of two prior stages and using analytical hierarchy process (AHP) with intent to provide the best decision.

As it depicted in the figure 3.6, the procedure in this thesis includes three different stages with mixture of qualitative and quantitative methods in order to catch as much information as possible. Shapes are coded by colours which blue refers to quantitative parts of survey, green to the qualitative units and pink to the analysis parts. Three first units in the first stage are aimed to provide best questions for the comparison questionnaire. The data coming out of this inquiry and the rest of first stage leads to interviews with proper questions and directed data gathering from the documents and environments. The third stage uses the information collected from two prior stages for preparing inputs of analytical hierarchy process (AHP) and summarization of all gathered information.

The upcoming lines in this chapter contain the questions asked in the first

pilot questionnaire and the reason for this questionnaire. The chapter will be followed by questions of second questionnaire and ended up by overall questions in the structured interviews.

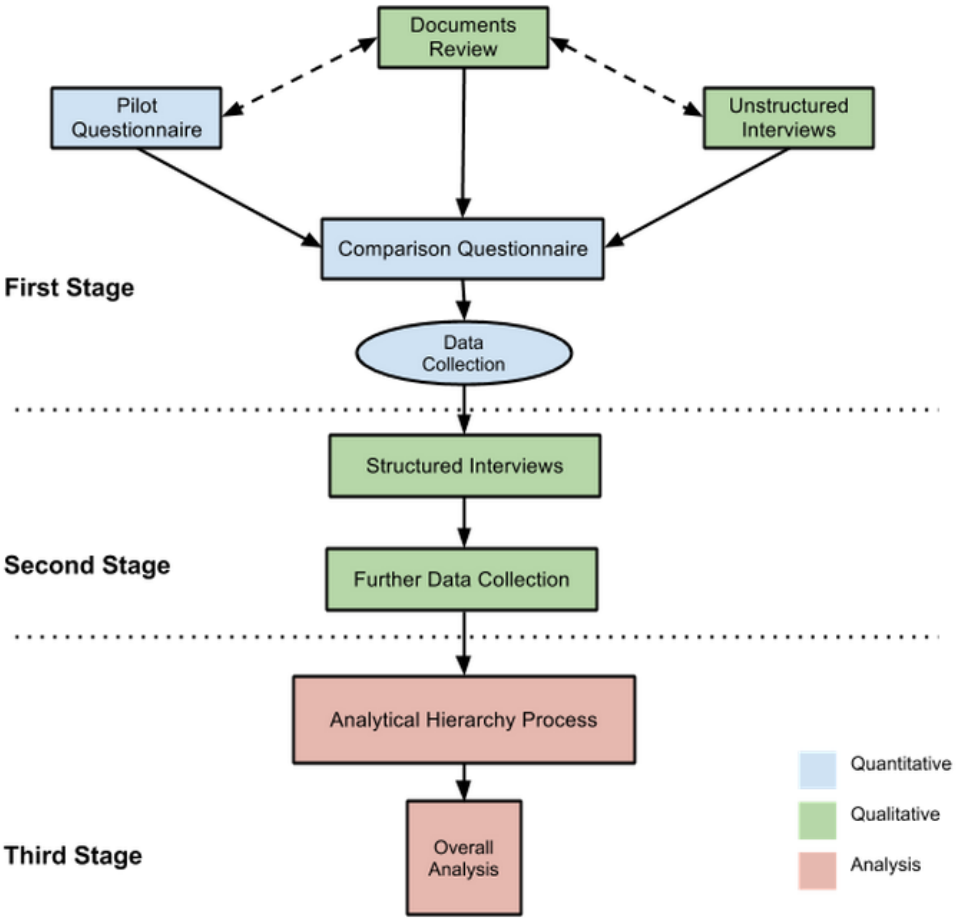

Figure 3.6: Thesis plan and structure.

3.4.1 Pilot Questionnaire

Since these two environment are being used by more than hundred developers and testers in their daily work for developing radio unit (RU) software, they are selected to contribute in the first questionnaire with the intent of obtain their valuable opinions about the WBTest and BBTest environments. Furthermore, these developers and testers are newly reorganized from different departments (design, development and test) and also from different levels of radio unit software development into cross functional teams (XFTs). In these new cross functional teams,

they are responsible for all radio unit software development process. Hence, categorizing them based on their prior experiences in radio unit software testing is another goal which is tried to be covered with the first questionnaire. It should be added that both questionnaires are anonymous and are not done electronically in order to gather more responses that could be not provided with online methods due to their heavy duties and daily works.

The first questionnaire includes following questions:
"Which testing tool were you using in your previous team in order to perform radio sub-system level tests before reorganization(establishing cross functional teams)?"

- BBTest.
- WBTest.
- Both of above options.
- Did not test in radio sub-system level.

The first question is asked in order to categorize the participants by their prior experience and then extracting their opinions in the next questions based on this categorization.

Second question in the first questionnaire is:
"Which development concept do you use in your new role in your cross functional team?"

- Test Driven Development(TDD)
- Behavioral Driven Development(BDD)
- Feature Driven Development(FDD)
- Never heard about them.
- No development has been performed yet.
- No explicit concept.
- More than one of the mentioned concepts.

The main reason for asking this question refers to the development methods which being used by the WBTest and BBTest. The WBTest environment uses behavioural and test driven method and the BBTest applies feature driven method. Since the cross functional teams are composed from different departments, it is decided to know which development method they know and apply in their daily work in sub-system level and other levels.

Third question:
"Based of your opinion as a cross functional team member, which of the following testing tools is more proper in order to perform successful radio sub-system level tests?"

- WBTest.
- BBTest.
- Both are needed, but integrated into one tool.
- Both are needed, but not integrated into one tool.
- None of them. New tool(tools) is required to be developed.
- Cannot make proper judgement.

As it mentioned previously, the participants of this questionnaire are engaged with both environments in their daily work therefore, they can be considered as those who know both environments properly and are aware of their pros and cons as well as the way these environments perform testing. This question collects their opinion about the proper testing environment.

In the following question the respondent are asked to provide their reasons for their previous response. Although this question will not be used in the statistic analysis of this questionnaire, it is good aid for the thesis worker to investigate and compare the results and reasons.

"Please provide short reason for your choice in the previous question."

The responses to this question are collected and read carefully in order to keep track of the participants opinion with respect to responses from previous question and used in overall analysis.

Fifth and sixth questions:
"Grade WBTest, how does it fulfill testing goals in radio sub-system level?"

- Very good coverage.
- Good coverage.
- Moderate coverage.
- Weak coverage.
- No or little experience in it.
- Other.

And:
"Grade BBTest, how does it fulfill testing goals in radio sub-system level?"

- Very good coverage.
- Good coverage.
- Moderate coverage.
- Weak coverage.
- No or little experience in it.
- Other.

The aim for these two questions is to use the participants experience in these two testing environments and investigate that how different categories (extracted from first question) grade testing environments that they are experienced in.

3.4.2 Comparison Questionnaire

The second questionnaire is designed based on the result of first questionnaire, document reviews and some unstructured interviews with individuals in the company. This questionnaire focuses on grading two testing environments based on users opinions and experiences in both environments. This questionnaire in addition to first questionnaire will prepare initial analysis and guideline for qualitative part of survey which contains interviews with members of technical team for these environments and later, inputs and structure of analytical hierarchy process (AHP).The second complement questionnaire includes following questions in order to grade both environments in the defined criteria:

First question:
"Stability? Refers to Up-Time in testing environment that includes lab devices and all attached instruments."

- Grading is from 1 means least as completely unstable and 10 means most as completely stable.

Stability represents that the software or environment can be up and running without any breakdown. Since changes, configurations and maintenance are continuous tasks in every environment, the new changes may lead to instability in that environment. This newly configured environment should provide proper response to the daily needs with keeping system in its high standard and quality.[72] This attribute can be considered as main attribute among the other attribute, while existence of other attribute depends on stability of software or environment.[73]

Second question:
"Usability? Indicates how easy to use testing environment. (Language, User interface, Tracing logs, reading reports and ..."

- Grading is from 1 means least as completely difficult and 10 means most as completely easy.

Usability indicates easy-to-use concept and is a quality attribute in soft-

ware design. In addition, it can be expressed by following quality components: Learnability, Efficiency, Memorability, Errors and Satisfaction.[74]

Third question:
"Coverage? Means how do test cases cover predefined specification (Design specification for WBTest environment and functional specification for BBTest environment.)"

- Grading is from 1 means least as completely coverage and 10 means most as completely no-coverage.

Specification coverage refers to that how much it can exercise the software under the test by a serie of execution. In fact, specification coverage is considered as an important measurement in the software quality and has a precious role in the software testing.[75, 76, 77, 78, 79]

Fourth question:
"Grading is from 1 means least as completely dissatisfied and 10 means most as completely satisfied."

- Grading is from 1 means least as completely coverage and 10 means most as completely no-coverage.

Documentation refers to all communicable materials which are applied in order to explain, describe, instruct procedures and systems. It can contain parts, installation, configuration, maintenance, and use of a specific system. In fact, documentation can provide shorter time to run, use and maintenance for a system as well as save time and resources. Proper documentation can mitigate the cost for further support as well.[80]

Fifth question:
"Technical support? Indicates useful and in time responses via email or phone and in mailing list and forum by support team in case of problems."

- Grading is from 1 means least as completely coverage and 10 means most as completely no-coverage.

Technical support indicates a set of assistances that support team provide to users of these two environments. In general, technical support team attempts to help the user solving their specific problems with testing environments, rather than providing training. Technical support can be delivered over the phone, by e-mail and mailing lists or a specific tool where users can log the problem.

Sixth question:
"Training? Expresses user conferences, training sessions for groups or individual in person or remotely."

- Grading is from 1 means least as completely coverage and 10 means most as completely no-coverage.

Training refers to how the responsible team provide routine training session in order to keep users updated in their skills and knowledge. This can increase users efficiency and decrease extra resource usage. Obviously, user conference can be interactive with intent of sharing users experiences with each other and learning from experiences of other attendances.

As a consequence, data gathered from these two questionnaire provides better understanding from these two environments and then more proper questions in the following interviews and analysis.

3.4.3 Interviews

After conducting two questionnaires and reviewing related documents, two structured interviews with members of WBTest and BBTest frameworks team are defined. They are asked to represent their ideas about these two environments and their specifications, five possible scenarios and feasibility of integration in addition to which one cost less to port to another one as well as they personal opinion about their most preferred scenario. They are asked to be fair in their judgments and are asked to respond the second questionnaire and grade these two environments. The responses as well as document reviews will be discussed and evaluated in analysis chapter.The responses as well as document reviews will be at the end of result and discussion chapter. These interviews as well as analytical hierarchy process

and document reviews assist the thesis worker in order to analyse the results of the questionnaire in the analysis chapter. The five scenarios will be explained in detail in the analysis chapter with respect to all gathered information from two first stages of the plan.

Chapter 4

Result and Discussion

This chapter contains the outcome of the questionnaires are done during this thesis. It will be started by results of the pilot questionnaire, followed by a second complement questionnaire. These results are going to be evaluated in the analysis chapter in addition to all gathered information by qualitative part of this research.

4.1 Pilot questionnaire results

The two environments (WBTest and BBTest) are being used by more than one hundred individuals (member of cross functional teams) in their daily work for developing the radio unit (RU) software. As mentioned in the approach chapter, they are selected to cooperate in the first questionnaire with the intent of obtaining their valuable opinions about the WBTest and BBTest environments. It should be noted that the number of respondents in the first questionnaire was 79 that are almost 76% of the population with 104 testers .

Result of the first question: **"Which testing tool were you using in your previous team in order to perform radio sub-system level tests before reorganization(establishing cross functional teams)?"** is depicted in the figure 4.1.

As can be seen in figure 4.1, 27% of the respondents did not test in the

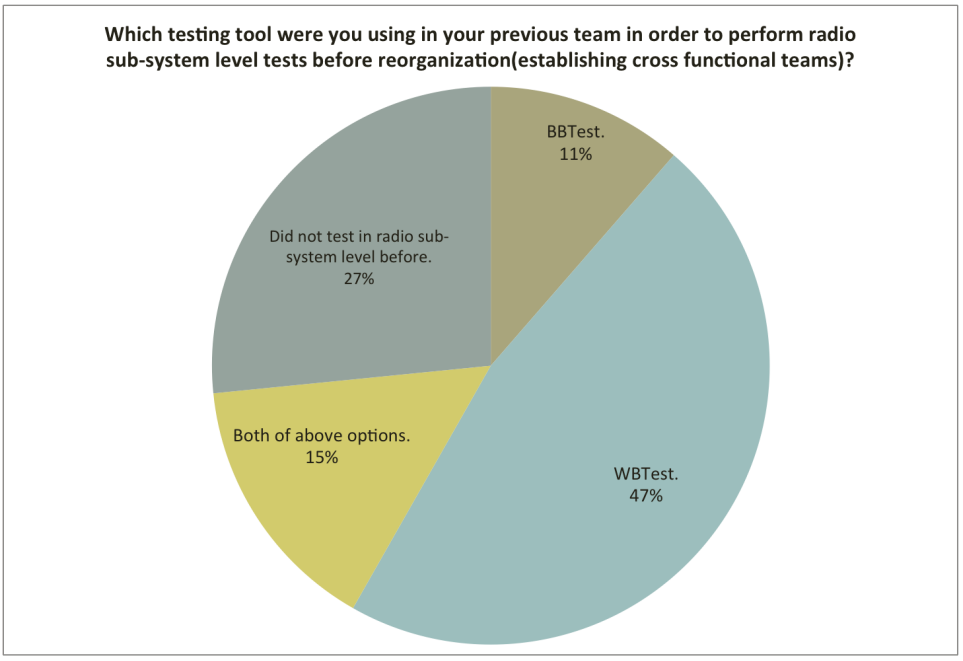

Figure 4.1: Result of the first question in the first questionnaire.

radio sub-system level which is the main working level for the WBTest and BBTest environments, before joining these new cross functional teams. It shows that 11% of participants are experienced in the BBTest environment previously and this amount for the WBTest environment is 47%. Those who are experienced in both WBTest and BBTest compose 15% of responders.

Figure 4.1 shows the result of the second question in which the testers are asked about their development method applied in their daily work: **"Which development concept do you use in your new role in your cross functional team?"**.

As depicted in figure 4.1, Test Driven Development (TDD), Bahavioral Driven Development (BDD) and Feature Driven Development (FDD) are used by 24%, 4% and 39% of respondents respectively. Feature Driven Development (FDD) that is mainly used in the BBTest environment gained 39% which is highest among other options and second place is allocated to Test Driven Development (TDD) by 24% of responders. As it shown, 8% of testers responded that they have never heard about these three possible options and 6% of participants have not performed any development until this questionnaire has been carried out. Only 3% of responders do not use any explicit concept in their daily work and finally 16% of them apply more

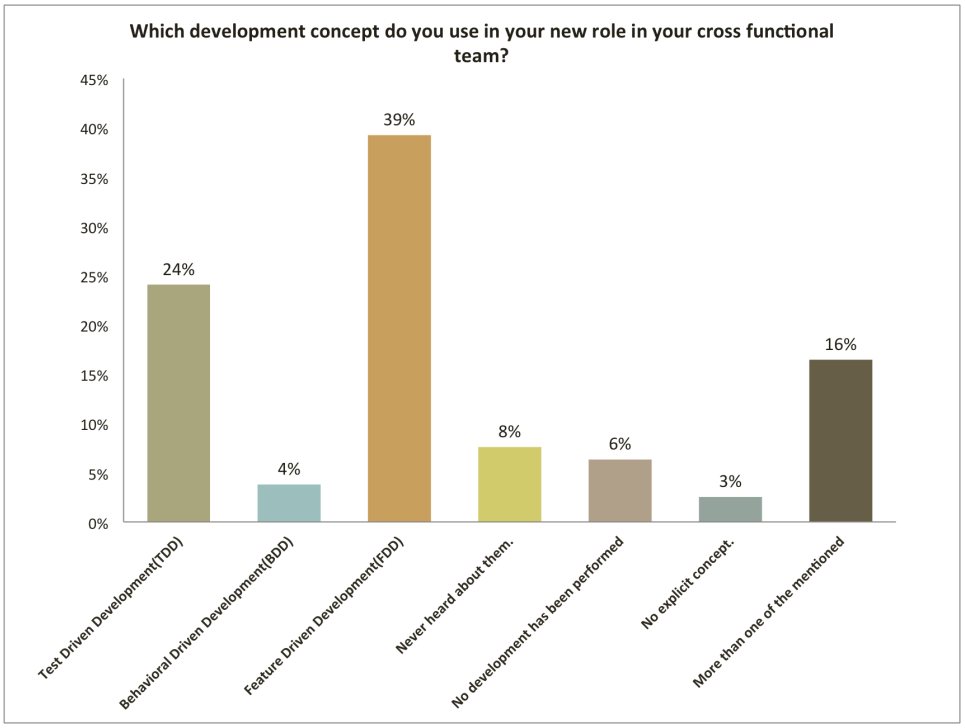

Figure 4.2: Result of the second question in the first questionnaire.

than one of these three development options (TDD,BDD and FDD).

The third question which can be considered as the most important question in the pilot questionnaire asks participants aspects about proper testing environment for performing sub-system level tests. The responses to the question: **"Based of your opinion as a cross functional team member, which of the following testing tools is more proper in order to perform successful radio sub-system level tests?"** are illustrated in figure 4.1.

Figure 4.3 represents ideas of all participants in the questionnaire about proper environment for sub-system level testing. It shows, 41% of them believe that these two environments are needed and should be integrated into one tool including advantages of both environments. In addition, 34% of respondents mentioned that both of these environments are needed, but without any changes such as integration into one environment. The WBTest and BBTest attracted 6 and 10 percent of participants respectively and only 4% of them believe that a new environment should be developed and none of them are suitable for the sub-system level testing goals. Finally, 5% of contributors could not make proper judgment and preferred to

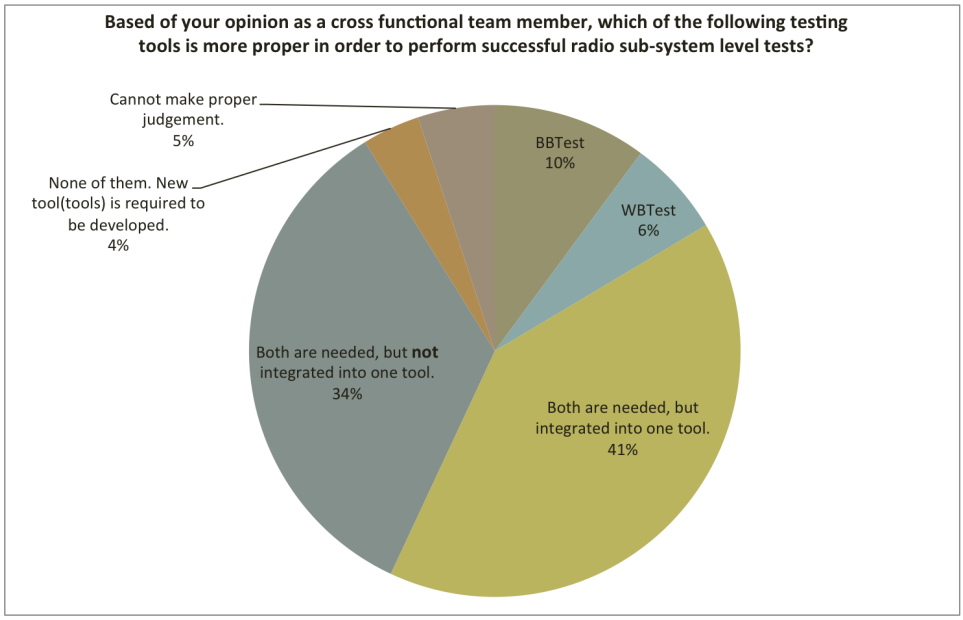

Figure 4.3: Result of the third question in the first questionnaire.

not choosing any stated options. Obviously, 75% of testers think both environment should exist but integration achieved more responses by 41% in comparison to 34% who believe there is no need for integration. Among BBTest and WBTest alternatives, BBTest received 4% more responses than WBTest option. Moreover, at the end of this ranking, developing new tool is located by only 4% that are even less than those who did not make any judgment between these options.

Four upcoming figures: 4.4, 4.5, 4.6, 4.7, indicate four different categories based on the participants prior experiences which is asked in the first question of pilot questionnaire. These categories assist readers to understand how different classes of testers stated their opinions about suitable environment in the sub-system level testing.

Figure 4.4, shows that options "Both are needed, but integrated into one tool." and "Both are needed, but not integrated into one tool" achieved equally 38% responses of the prior WBTest experienced users. Additionally, none of the them thinks a new test environment should be developed in order to fulfill the goals. The BBTest and WBTest environments are in the next places by 11 and 8 percent respectively and 5% of respondents could not make proper judgment on these options. The interesting point in this result is, the "WBTest" option gained less supporters than "BBTest" option

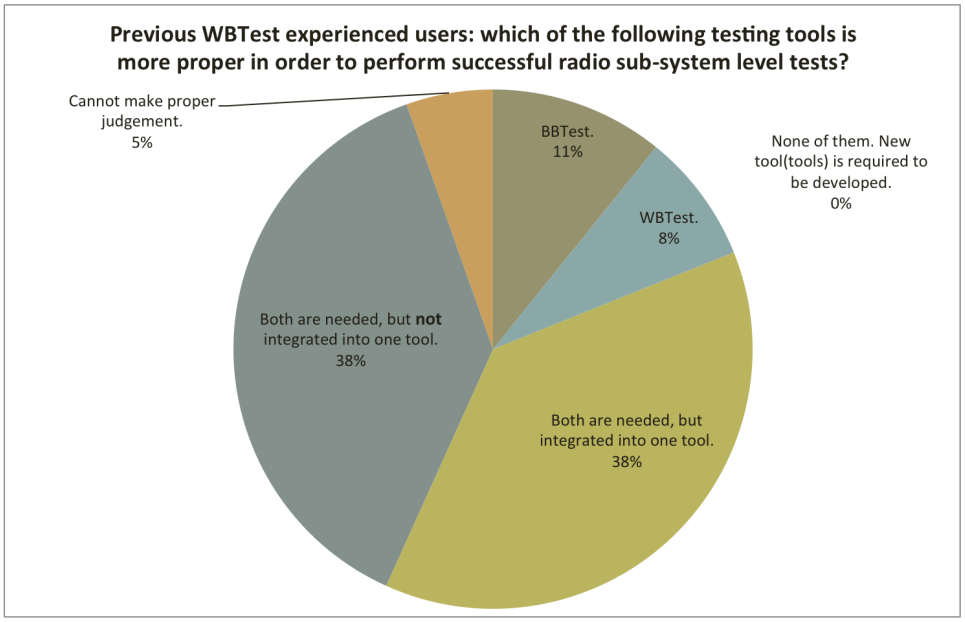

Figure 4.4: Result of the third question in the first questionnaire for the prior WBTest experienced users.

from prior experienced testers in the WBTest. Furthermore, although 76% of testers responded that both environments are needed, they are divided equally into integration and no integration alternatives.

Figure 4.5 expresses that 45% of the prior BBTest experienced responders prefer to have an integrated environment of them and none of them supports the WBTest environment, development of new tool as well as keeping them both without integration. In addition, 33% of them stated BBTest environment is suitable for sub-system level testing and 22% of these participants could not judge properly. As it can be seen in the result, for the prior BBTest experienced user, the WBTest is not proper test environment for fulfilling all sub-system level testing goals but 33% of them think the BBTest environment can fulfill predefined goals. Interestingly, option "BBTest" received 12% less respondents than those that think integration of these two environments is the most proper alternative which gained 45% of these participants.

As it illustrated in the figure 4.6, All contributors who are experienced in the both environments previously think that both WBTest and BBTest environments are needed, but 58% of them supports integration of these two into one and 42% believe that they should not be merged into

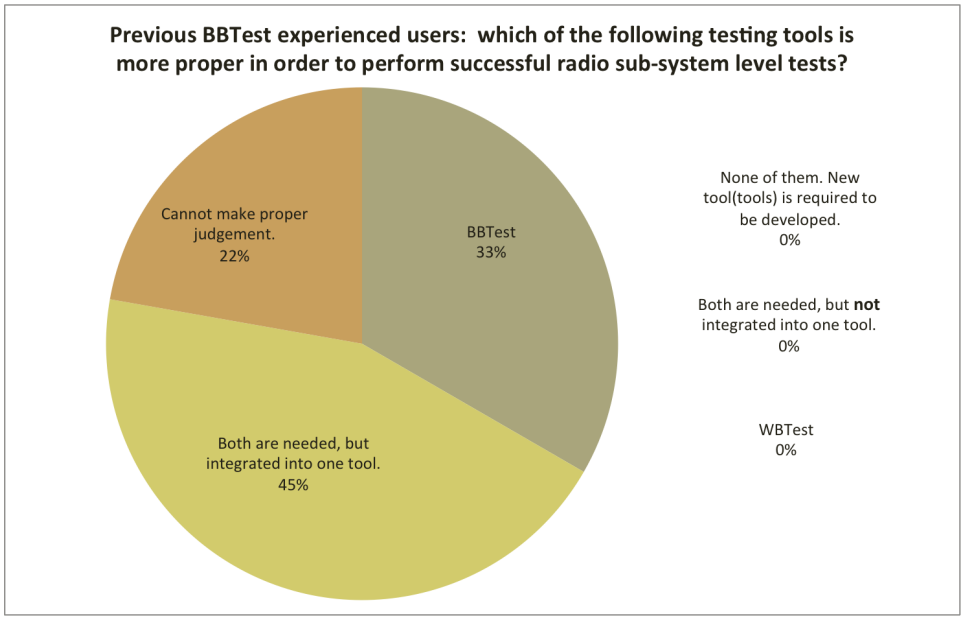

Figure 4.5: Result of the third question in the first questionnaire for the prior BBTest experienced users.

one environment. The rest of the options received no support among participants who worked with both environments. In the other word, those who were experienced in both environment for performing sub-system level tests did go for neither WBTest nor BBTest environments. Actually, all of them think, none of these two environments can fulfill sub-system level testing goals without having properties of the other one.

The responses of new testers in sub-system level testing are illustrated in the figure 4.7. Around 38% of the new sub-system level testers stated that both are needed, but it should not be integrated. In the next place is integration of these two into one tool with 33%. The development of a new tool, the WBTest and BBTest are in the next places with 14, 10 and 5 percent of new testers in sub-system level testing respectively. About 71% sub-system level testers responded that both environments are needed but keeping them without merging gained 5% more responses than merging them into one environment. In addition, 14% of them think a new environment should be developed which is more than responses to BBTest and WBTest options.

In the fourth question, as it stated in the approach chapter, the contributors are asked to provide short description about their opinions in the third

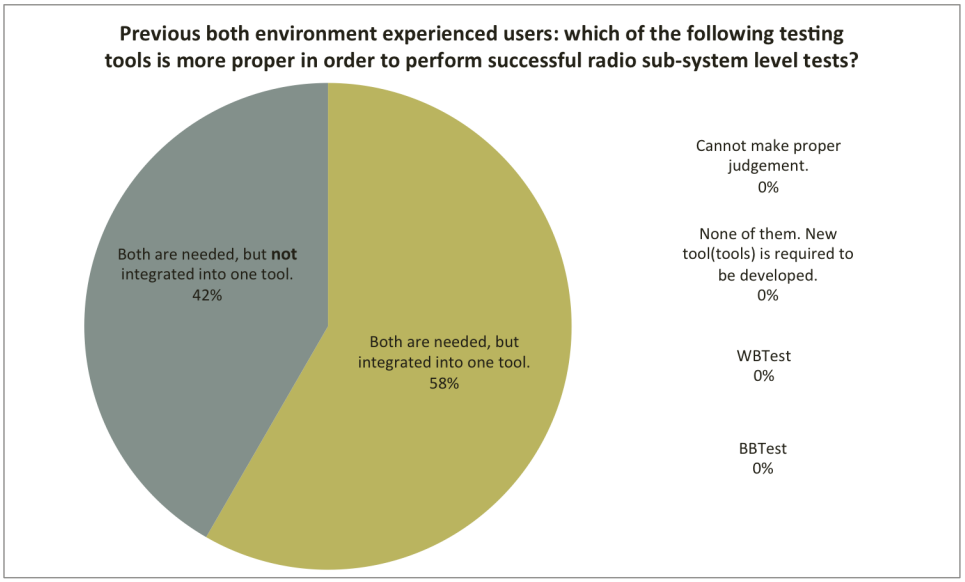

Figure 4.6: Result of the third question in the first questionnaire for the prior both environments experienced users.

question which are used to prepare the questions of structured interviews which will be explained in the analysis chapter. This response will not be provided in this chapter and since it was qualitative question, it is used for digging more into the proper documents and evaluating these two environments adequately.

In the fifth and sixth questions, the participants are asked to answer the following raw questions about coverage in the sub-system level for both environments. These questions are as follow: **"Grade WBTest, how does it fulfill testing goals in radio sub-system level?"** and **"Grade BBTest, how does it fulfill testing goals in radio sub-system level?"**.

Figure 4.8 shows that 44% of participants believe WBTest coverage in radio sub system level is moderate while 25% of them think the BBTest has moderate coverage. On the other hand, the responses of "Good coverage" for the BBTest and WBtest are 42 and 22 percent respectively. A similar trend can be seen for "Very good coverage" responses which are 10% for the BBTest and 4% for the WBTest. Around 14% of respondents believe that the WBTest coverage in sub-system level is weak while only 6% are with "Weak coverage" for the BBTest. By comparing coverage of two environments, 10% mentioned that the BBTest has very good coverage

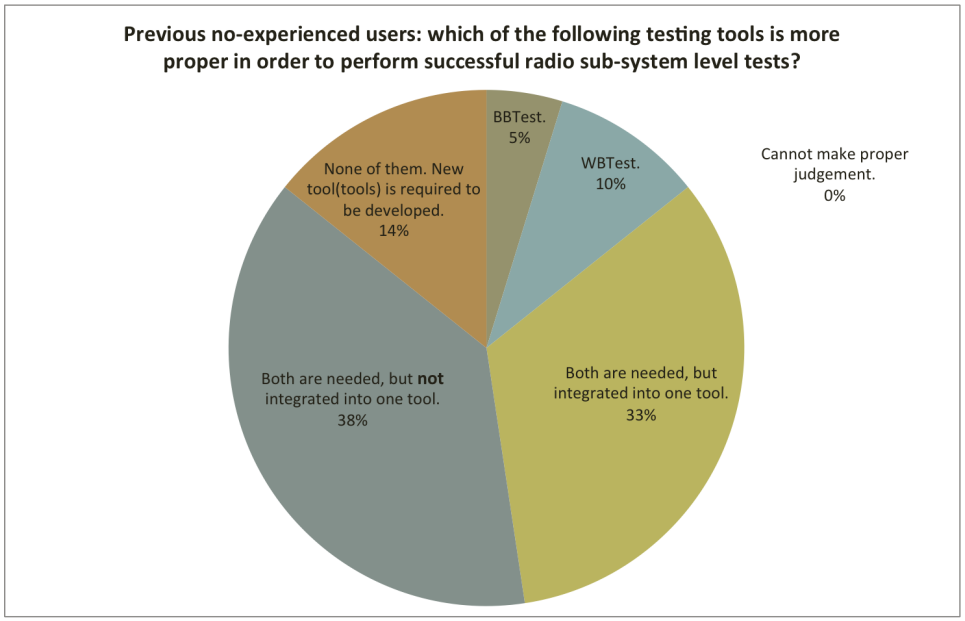

Figure 4.7: Result of the third question in the first questionnaire for the new tester in sub-system level testing.

while only 4% of respondents stated it for the WBTest. Additionally, 13% of respondents have no or little experience in either the WBTest or BBTest. It also represents, 52% of contributors stated that the BBTest has "Good coverage" and "Very good coverage", while this amount for the WBTest is half of it by 26 percent. General overview on this results shows that responses tend to the BBTest environment, when coverage level gets better. In other words, the WBTest received more responses in lower coverage and for higher coverage level the BBTest got more.

4.2 Results of the Complement Questionnaire

As stated in the approach chapter, the second questionnaire is complement questionnaire that investigates and grades these two testing environments based on selected criteria such as stability, usability, specification coverage, documentation, technical support and training which are effective in comparing software environments and reveal their drawbacks. They also assist researchers in analytical hierarchy process pairwise comparison which are going to be performed in the analysis chapter. It should be

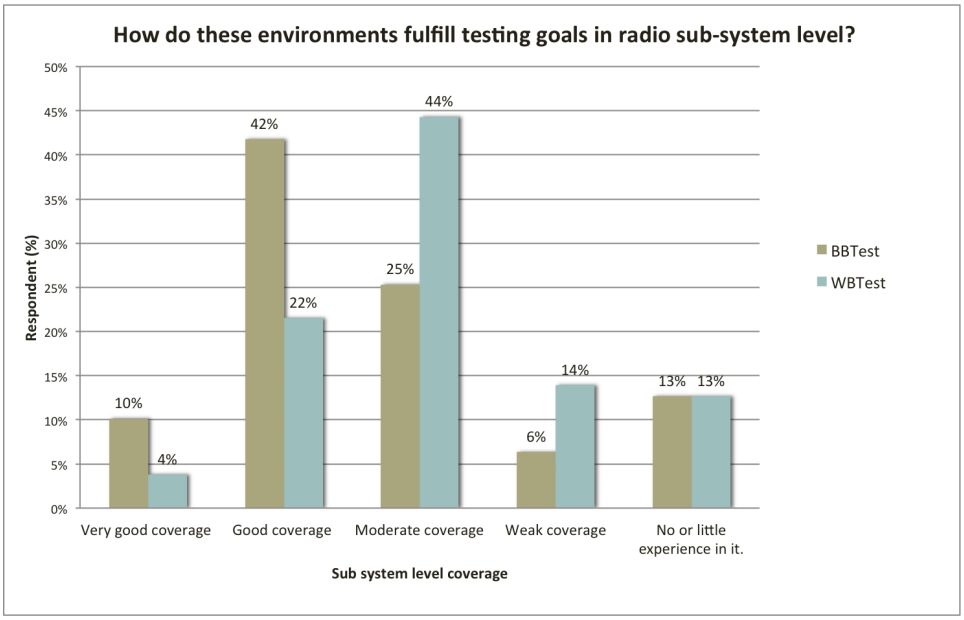

Figure 4.8: Result of the fifth and sixth questions in the first questionnaire.

noted that sample size is 72 and the population is the same as in the pilot questionnaire. In addition, the short description of each criterion which are going to be demonstrated here is provided in the approach chapter.

4.2.1 Stability

In the first question, they are asked to grade stability of the environments under study: **"Stability? Refers to Up-Time in testing environment that includes lab devices and all attached instruments."**. Figure 4.9 represents the result of this question.

As figure 4.9 shows, 10% of respondents found the WBTest environment(Located in the left side of the chart) "Absolutely stable" while none of them thinks so for the BBTest. Around 22% of sub-system level testers suppose the WBTest is "Moderately stable" and this is only 3% for the BBTest. About 20% of respondents mentioned that the BBTest is "Weakly stable" or even more unstable, but only 3% of them stated that WBTest is "So unstable". Almost 76% and 77% of participants found the WBTest and BBTest "Stable" and even higher stability respectively which shows that these two environments are almost similar from stability point of view

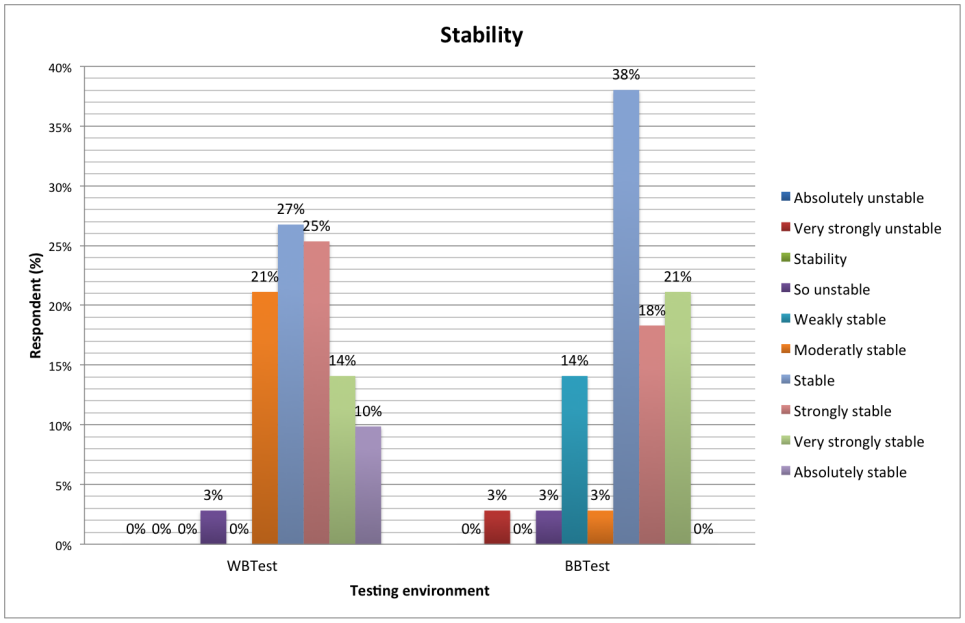

Figure 4.9: Result of the first question about stability in the complement questionnaire.

for sub-system level testers. The most responses are attracted by option "Stable" in the BBTest by 38% which is the most chosen option in the WBTest by 27% as well. Only the WBTest received responses for "Absolutely stable" which are 10% of respondents. Option "Very strongly stable" is chosen by 21% of respondents for the BBTest comparing to 14% for the WBTest. But option "Strongly stable" shows a reverse trend, 25% for the WBTest and 18% for the BBTest. Furthermore, there is huge difference for the "Moderately stable" between two environments, only 3% of responses for the BBTest and this amount is 22% for the WBTest. Interestingly, 14% of respondents did go for "Weakly stable" in BBTest in comparison to no responses for the WBTest. Moreover, it should be noticed that there are 3% of responses for "Very strongly unstable" in the BBTest. On the other hand, the lowest grade for the WBTest is for "So unstable" option with 3%.

Figure 4.10 indicates the average grades for the stability of these two environments. The WBTest gained 7.54 out of 10 and almost similarly, the BBTest got 7.07 out of 10 in average. Their standard deviation are 1.38 and 1.62 respectively. Notice that higher standard deviation of WBTest shows that responses for the BBTest are a bit more polarized than responses for the WBTest.

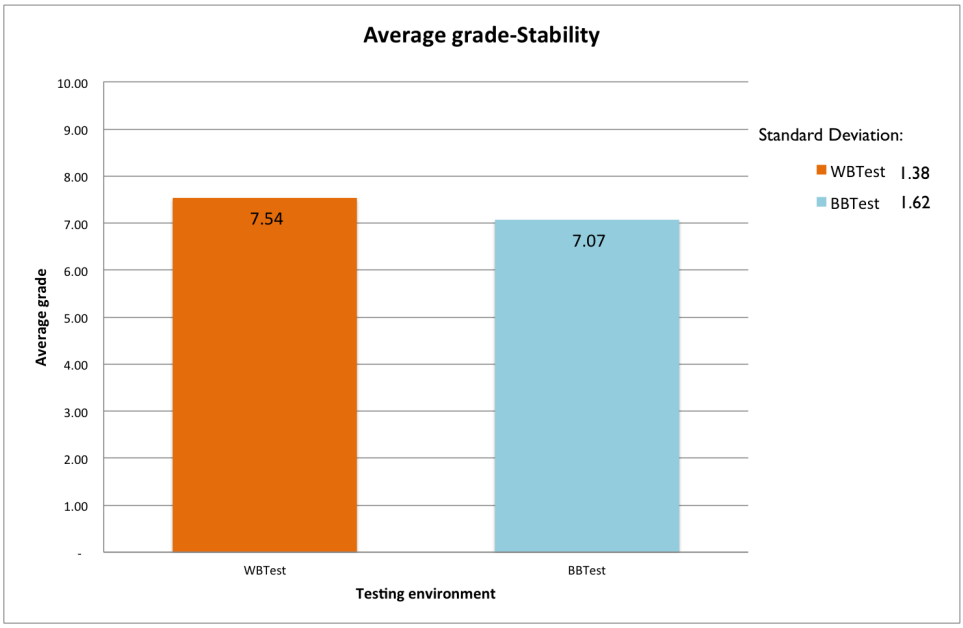

Figure 4.10: Average grade of stability for the WBTest and BBTest environments.

4.2.2 Usability

The second question in the complement questionnaire refers to usability of these two environments. The question is as follow: **"Usability? Indicates how easy to use testing environment. (Language, User interface, Tracing logs, reading reports and ..."**. The result of this question is depicted in the figure 4.11.

Figure 4.11 shows that none of the respondents found both environments "Absolutely easy" but 14% of them think the WBTest is "Very strongly easy" to use in comparison to only 3% for the BBTest environment. Although 62% of participants found the WBTest "Easy" and even more easy, 52% of them stated similar responses for the BBTest. Percentage of testers who think these two environment are weakly easy or less easy are almost equal, 26 for the BBTest and 24 for the WBTest. Furthermore, none of the sub-system level testers found these two environments "Absolutely difficult" or "Very strongly difficult" to use. In contrast, only 8% of them found the WBTest "Strongly difficult" comparing to just 4% for the BBTest environment. About 10% of respondents selected "Strongly easy" for BBTest comparing to 24% in the WBTest but 39% of them chose "Easy" in the BBTest comparing

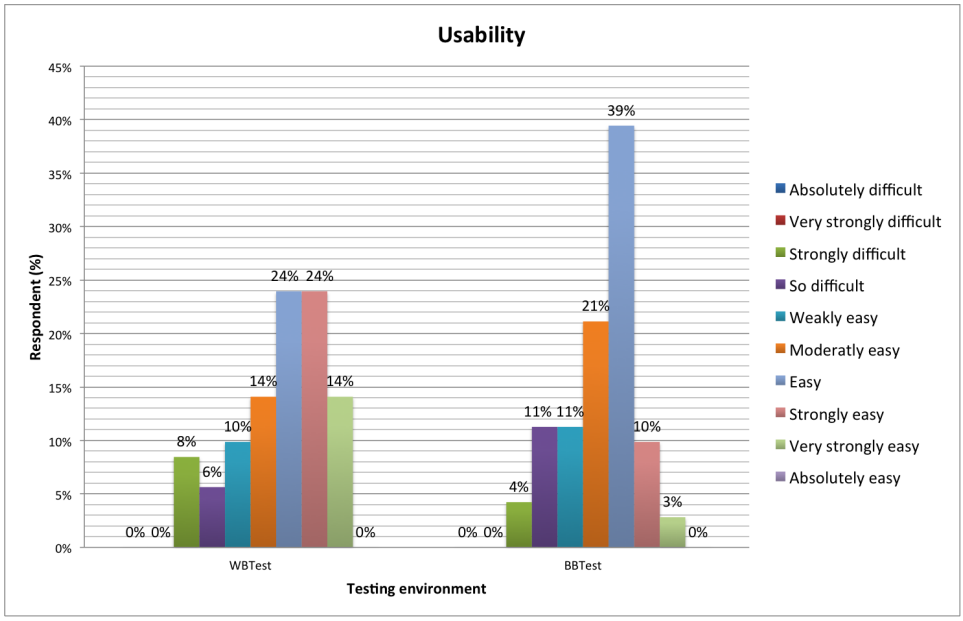

Figure 4.11: Result of the second question about usability in the complement questionnaire.

to 24% for the WBTest environment. Similarly, 21% stated that WBTest is "Easy" to use which on the other hand, 14% selected this option in the WBTest. Responses to "Weakly easy" are nearly similar with 11% and 10% for the BBTest and WBTest respectively. Option "So difficult" is chosen by 11% for the BBTest which is selected by 6% of them for the WBTest. Conversely, the "Strongly difficult" is selected by 8% of respondents for the WBTest in comparison to 4% for the BBTest. Two lowest grades are for "Very strongly difficult", "Absolutely difficult" as well as "Absolutely easy" that received no responses in both environments.

Figure 4.12 represents the average grades that usability of these two environments achieved. The average usability grade of the WBTest is 6.68 out of 10 and on the other hand, the BBTest gained 6.21 out of 10 in average. The standard deviation of the responses for the WBTest is more than the BBTest with 1.77 comparing to 1.40. They show more polarized responses for WBTest with higher standard deviation than the BBTest.

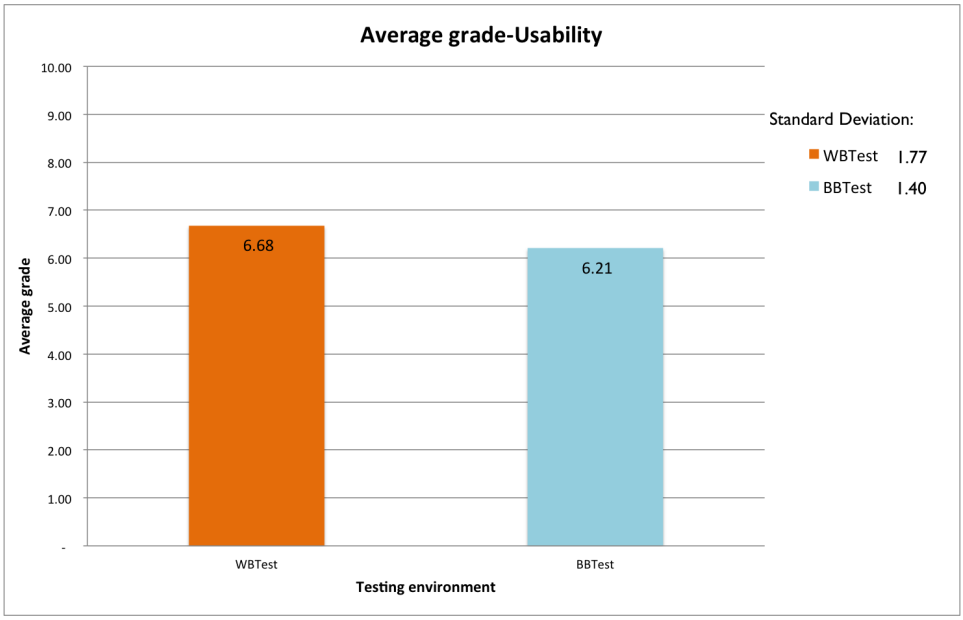

Figure 4.12: Average grade of usability for the WBTest and BBTest environments.

4.2.3 Specification Coverage

Specification coverages of these two testing environments are asked in the third question of complement questionnaire. The responses to the question about coverage are illustrated in figure 4.13. The third question is as follow: **"Coverage? Means how do test cases cover predefined specification (Design specification for WBTest environment and functional specification for BBTest environment.)"**.

As it depicted in figure 4.13, "Coverage" with 31% , "Strongly coverage" with 37%, "Very strongly coverage" with 18% and "Absolutely coverage" with 7% attracted 93% of responses in the BBTest which show high specification coverage in it comparing to only 7% "Coverage" for the WBTest environment. The higher coverage alternatives are not chosen at all in the WBTest. While option "Coverage" got only 7% of respondents for the WBTest, it gained 31% of responses in the BBTest environment. Similarly, 93% of respondents found that the WBTest specification coverage is moderate and even worst while it is only 7% for the BBTest environment. Most of the responses in the WBTest received by "Weakly coverage" with 37% and this was "Strongly coverage" in the BBTest environment with

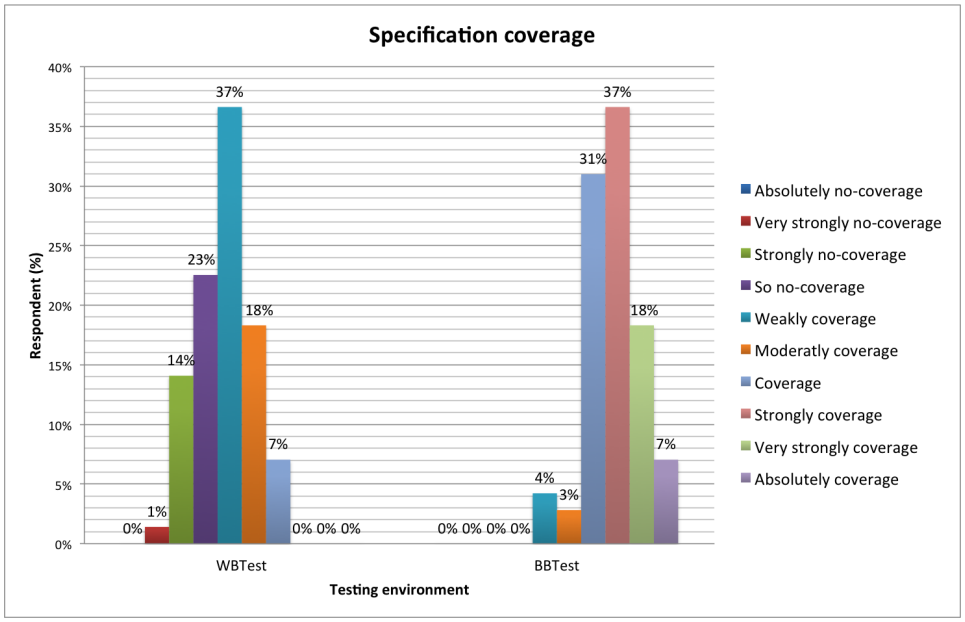

Figure 4.13: Result of the third question about specifications coverage in the complement questionnaire.

37% of responses. The second highest rate of responses in the BBTest is for option "Coverage" with 31% comparing to only 7% in the WBTest for the same alternative. Moreover, option "Weakly coverage" is selected by only 4% of contributors in comparison to 37% of respondents in the WBTest. Additionally, none of the alternatives "So no-coverage", "Strongly no-coverage", "Very strongly no-coverage" and "Absolutely no-coverage" are selected by respondents in the BBTest, but these options received 23, 14 and 1% of responses and just "Absolutely no-coverage" got no respondents in the WBTest.

Figure 4.14 shows the average grades of these two environments in specification coverage. As it can be noticed in the figure14, the WBTest got only 4.77 out of 10 and the BBTest got 7.83 out of 10 in average. Obviously there is big difference between these two environment in case of specification coverage. Responses show that BBTest has higher specification coverage comparing to the WBTest environment. Their standard deviation are 1.16 and 1.12 respectively that are almost equal. In fact, they do not show high polarization in the responses for both environments under study.

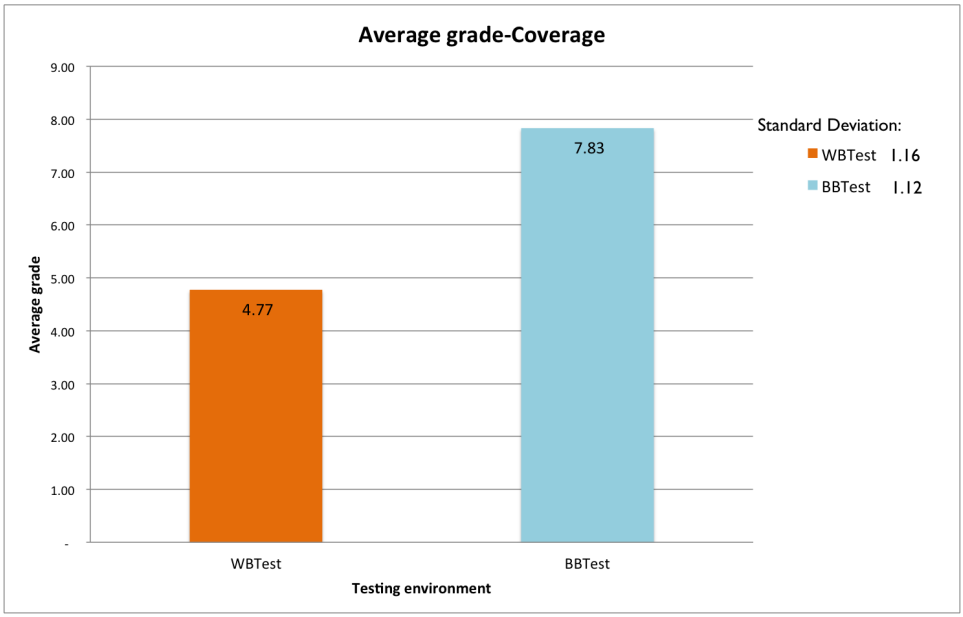

Figure 4.14: Average grade of specifications coverage for the WBTest and BBTest environments.

4.2.4 Documentation

In the fourth question, they are asked about documentation quality and availability in these two environments: **"Documentation? Represents clear and useful documents, handbooks and ... in the wiki or web pages of environment being asked."**. Figure 4.15 represents the result of the fourth question related to the documentation.

As it shown in figure 4.15, rate of satisfaction for the WBTest by 73% of responses with respect to options "Satisfied", "Strongly satisfied" and "Very strongly satisfied" (with 27%,31% and 15% respectively) is higher than its rate by 32% in the BBTest with 20%, 6% and 6% for similar alternatives. The contributors in the survey found the BBTest "Weakly satisfied" and "So dissatisfied" with 18% and 11%. On the other hand, "Weakly satisfied" and "So dissatisfied" options in the WBTest gained only 2% (one percent each) of responses. In addition, there are 6% (3% for "Strongly dissatisfied" and 3% for "Very strongly dissatisfied") of respondents who found strongly and very strongly dissatisfaction with WBTest while only 1% of them found "Very strongly dissatisfied" with the BBTest environment. The highest rate is for "Moderately satisfied" with 38% in the BBTest and in the other side,

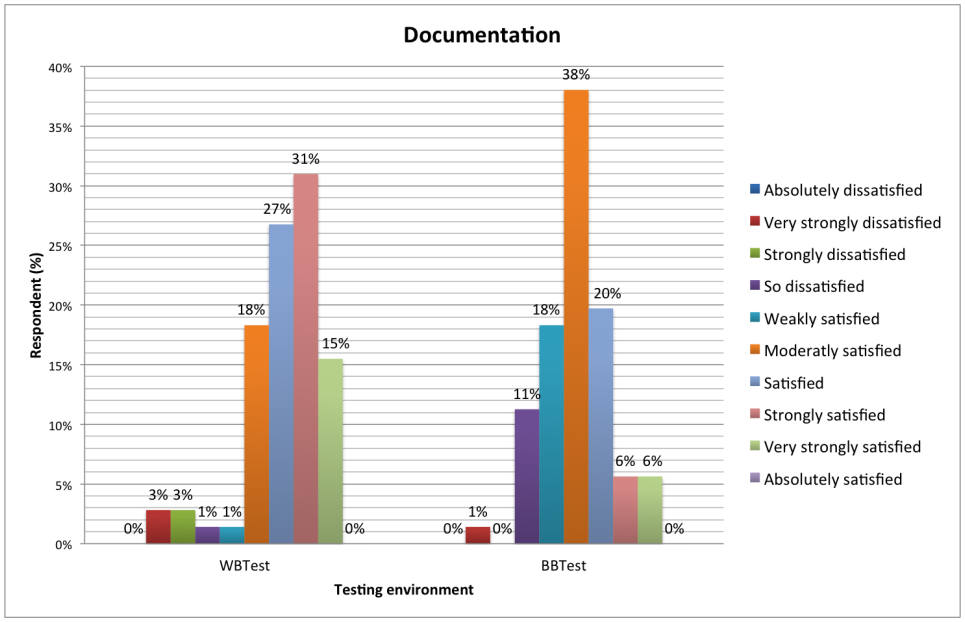

Figure 4.15: Result of the fourth question about documentation in the complement questionnaire.

it gained 18% of responses for the WBTest. On the other hand, "Strongly satisfied" received 31% of responses which is the most among other alternatives in the WBTest and it should be noticed that it only attracted 6% of responses in the BBTest environment. Similar trend can be seen for the "Very strongly satisfied" with 15% for the WBTest in comparison to 6% in the BBTest. Moreover, none of these environments got absolutely satisfaction and this is the same for the absolutely dissatisfaction.

Figure 4.16 represents the average satisfaction grades that documentation of these two environments achieved. As it can be seen, the average satisfaction in documents of the WBTest is 7.11 out of 10 and in the other hand, the BBTest got 6.01 out of 10 in average. The standard deviation of the responses for the WBTest is more than its value in the BBTest with 1.57 comparing to 1.35 respectively. This shows more fluctuation of opinions in the WBTest than the BBTest responses for documentation.

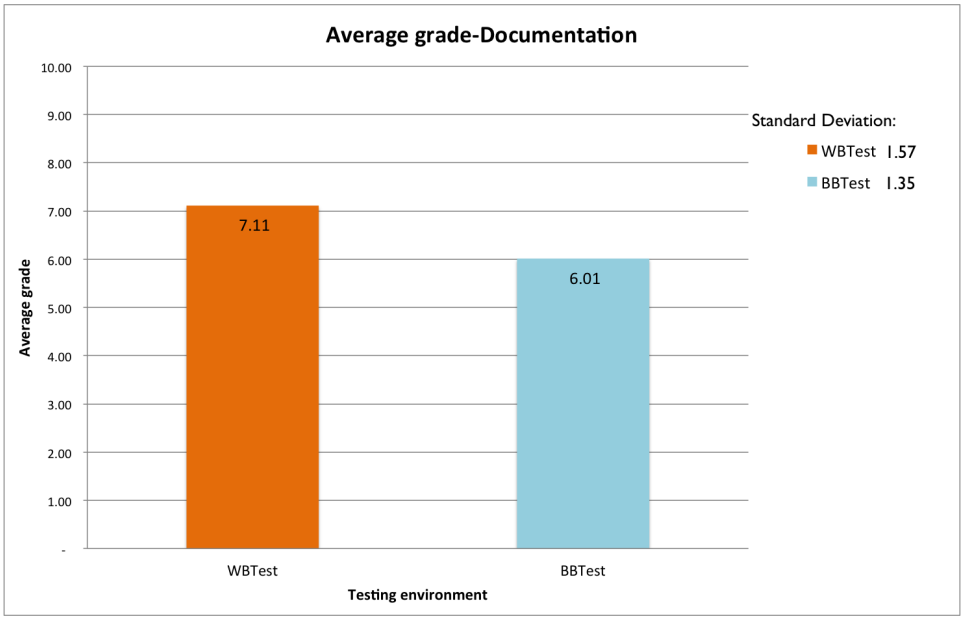

Figure 4.16: Average grade of documentation for the WBTest and BBTest environments.

4.2.5 Technical support

The technical support satisfaction rates of these two testing environments are asked in the fifth question of complement questionnaire. The responses to the following question about technical support are illustrated in figure 4.17. The question is as follow: **"Technical support? Indicates useful and in time responses via email or phone and in mailing list and forum by support team in case of problems."**.

Figure 4.17 demonstrates that 95% of respondents are "Satisfied" and even more satisfied with the technical support in the WBTest environment with 30% for "Satisfied", 41% for "Strongly satisfied", 10% for "Very strongly satisfied" and 14% for "Absolutely satisfied" in comparison to 53% for the BBTest environment with 32% for "Satisfied",7% for "Strongly satisfied", 11% for "Very strongly satisfied" and only 3% for "Absolutely satisfied". In addition, 35% of contributors stated that they are "Weakly satisfied" with the BBTest comparing to only 4% for the WBTest environment in the sub-system level testing. The highest rate of responses in both WBTest and BBTest is for "Strongly satisfied" with 41% for the WBTest environment. Conversely, "Weakly satisfied" got 35% of responses in the BBTest which

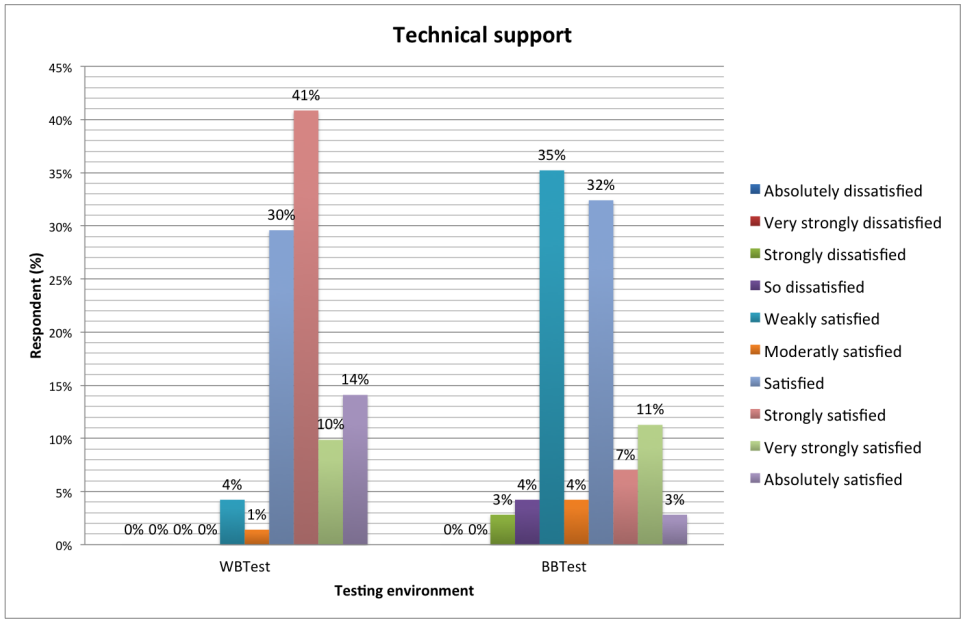

Figure 4.17: Result of the fourth question about technical support in the complement questionnaire.

is the highest rate among other grades in the BBTest. There are 3% of participants who stated they are "Absolutely satisfied" with the BBTest, this alternative achieved 14% in the WBTest. But option "Very strongly satisfied" got almost equal respondents in both environments with 11% for the BBTest and 10% for the WBTest. This trend can also be seen for "Satisfied" with 32 and 30 for the BBTest and WBTest respectively. Interestingly, four worst options got no responses in the WBTest, but for the BBTest this happened in two worst alternatives.

Figure 4.18 represents the average grades that technical support of these two environments achieved. As it is depicted, the average grade of technical support in the WBTest environment is 7.93 out of 10 and in the other hand, the BBTest gained 6.39 out of 10 in average. The standard deviation of the responses for the WBTest environment with 1.19 is less than its value in the BBTest environment with 1.66. They show a bit more polarized responses for BBTest with higher standard deviation but not so considerable.

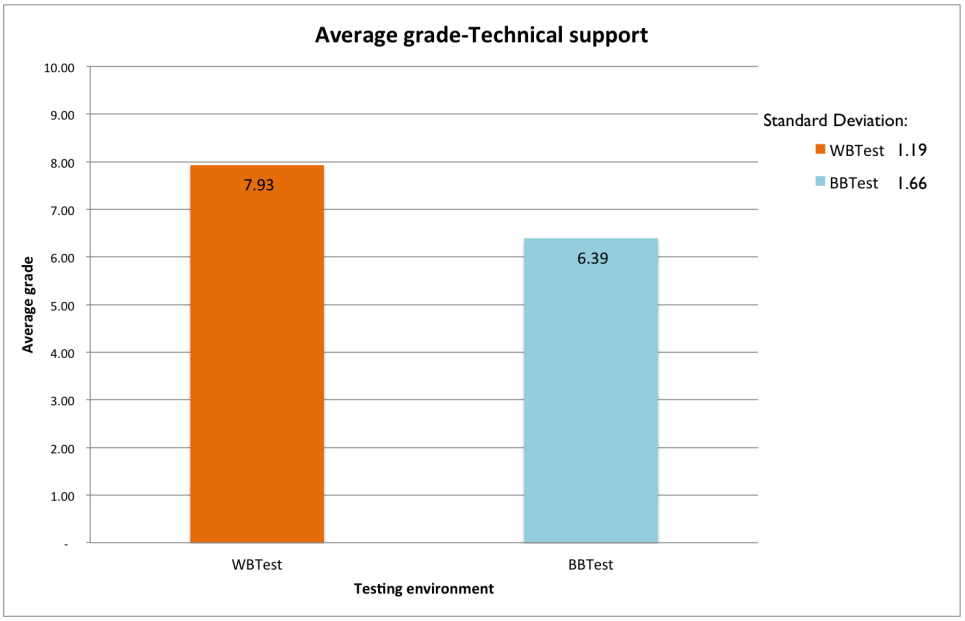

Figure 4.18: Average grade of technical support for the WBTest and BBTest environments.

4.2.6 Training

The final question of the complement questionnaire asked respondents about satisfaction grade of training in these two environments. The question is as follow: **"Training? Expresses user conferences, training sessions for groups or individual in person or remotely."**. The result of this question is depicted in figure 4.19.

As illustrated in figure 4.19, around 76 percent of respondents are "Satisfied" and even more with training that provided in the WBTest environment by 34% for "Satisfied", 34% for "Strongly satisfied", 6% for "Very strong satisfied" as well as 4% for "Absolutely satisfied". In contrast, this trend is 3% for only "Satisfied" option and three better options attracted no respondents in the BBTest poll box. Furthermore, 48% of respondents chose "Weakly satisfied" for the BBTest and options "So dissatisfied", "Strongly dissatisfied", "Very strongly dissatisfied" and "Absolutely dissatisfied" gained 20%, 11%, 6% and 3% for the BBTest environment respectively.

On the other hand, the WBTest responses show only 7% of participants for

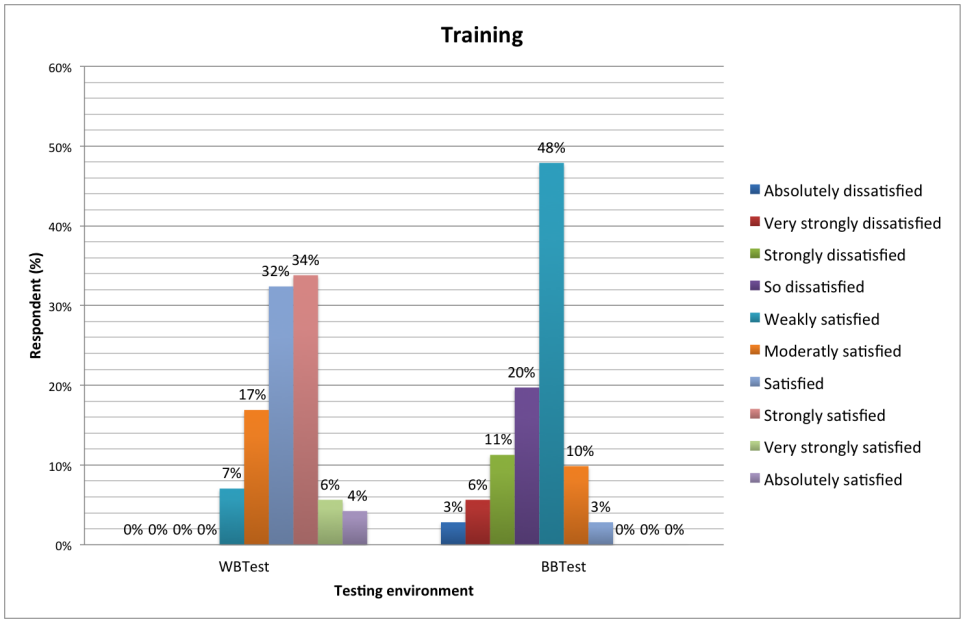

Figure 4.19: Result of the fourth question about training in the complement questionnaire.

the "Weakly satisfied" option and less satisfaction achieved no respondents in this environment. The highest rate is for the "Weakly satisfied" with 48% of responses in the BBTest and on the other side 34% and 32% for the "Strongly satisfied" and "Satisfied" for the WBTest environment. As mentioned "Weakly satisfied" got 48% in the BBTest, but this option achieved only 7% in the WBTest environment. Four worst alternatives in scaling gained no respondents in the WBTest, while 40% of responses in the BBTest are attracted by these four options and if "Weakly satisfied" would be added to this responses, this rate will increase to 88% of respondents.

Figure 4.20 expresses average grades for training of these two environments. The WBTest got 7.27 out of 10 and the BBTest got only 4.45 out of 10. This large difference in averages shows that the WBTest environment satisfied participants pretty better than the BBTest in case of Training. Their standard deviation are 1.16 and 1.24 respectively which are not so high and also almost similar to each other. Therefore, there is no big fluctuation of responses in both environments.

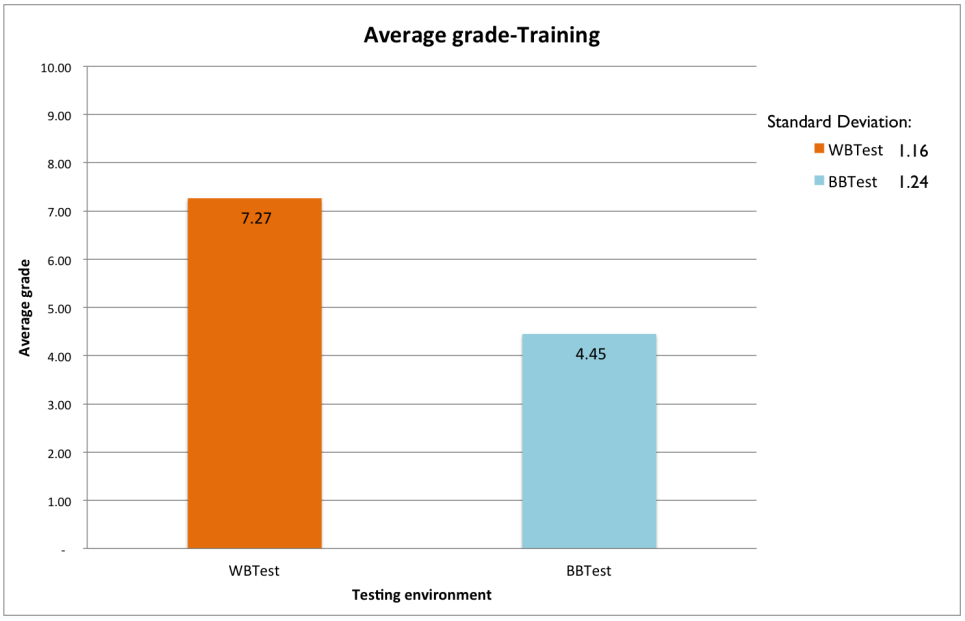

Figure 4.20: Average grade of training for the WBTest and BBTest environments.

In the next chapter which is analysis chapter, the result presented in this chapter will be sum up with all other information gathered in interviews and documents in addition to using Analytical Hierarchy Process (AHP) for analysis of the WBTest and BBTest environments.

Chapter 5

Analysis

In the analysis chapter comprehensive assessment of the results which is illustrated in a couple of charts in the previous chapter will take place. The results of two questionnaires as well as interviews and document reviews will be addressed. In addition, the Analytical Hierarchy Process (AHP) will be applied in order to compare these two testing environments.

5.1 Overall evaluation of result

The result of the questionnaires can categorize the respondents into four groups based on their prior experiences in sub-system level testing environments. Due to recent reorganization in the company, these testers are mixed up from different teams which may be involved in different levels of testing in the company. Hence categorizing them is useful with the intent of classification of their opinions. The following four groups are categorized in the first questionnaire:

ClassWB : Testers who have previous experience in the WBTest environment which uses black box testing perspective.(47%)

ClassBB: Testers who have previous experience in the BBTest environment which uses white box testing perspective.(11%)

ClassBoth : Testers who have previous experience in both WBTest and BBTest environments.(15%)

ClassNew: Testers who did not execute any test in the WBTest and BBTest environments and are new in sub-system level testing.(27%)

Since they were using different development methods in their previous teams (Before composing new cross functional teams which their members are responsible for developing and testing radio unit software), it was decided to ask them about development methods they use in their new roles in the cross functional teams. The result obtained from testers shows that Feature Driven Development(FDD) which is used by the BBTest environment with 39% gained the first place among other options and Test Driven Development (TDD) achieved second place by 24% which is the recommended method for testing in sub-system level but only 24% of participants are using this development method. It should be added that 16% of testers use more than three possible options and Behavioral Driven Development(BDD) is the least known method for sub-system level testers by only 4% of respondents.

By referring to the opinion of users about most suitable testing environment to achieve best outcome in the sub-system level, it is tried to provide them five possible scenarios, ask them about these scenarios and then categorize the result based on four defined groups of respondents. These five scenarios are as follow:

Scenario 1, "KeepWB": Keeping the current WBTest environment and halting test activities of the BBTest environment. In this scenario, all the testing responsibilities of sub-system level testing will be on the shoulder of the current WBTest environment and no more testing will be executed in the current BBTest environment.

Scenario 2, "KeepBB": Keeping the current BBTest environment and halting test activities of the WBTest environment. In this scenario which is opposite of first scenario, all the testing responsibilities of sub-system level testing will be on the shoulder of the current BBTest environment and no more testing will be executed in the current WBTest environment.

Scenario 3, "Merged": Merging and integrating these two environment into an environment with advantages of both WBTest and BBTest environments. This scenario covers all benefits in these two environments. Porting from one to another should happen but the matter is

porting from which environment to another one. Two sub-scenario is predictable in this case. Scenario 3-1: First one is porting the BBTest environment properties like test cases into the WBTest environment. In this case, the WBTest environment is the foundation and the BBTest will be ported into it. Scenario 3-2: Second scenario is vice versa. Porting WBTest properties into the BBTest environment occurs. In this case the BBTest environment is considered as basis of the scenario.

Scenario 4, "NotMerged" Keeping both two testing environment with their current testing activities. It means that there is no change needed in the current environment. This scenario tries to keep current testing environment in the sub-system level testing. Obviously no merging takes place and these environments will carry on their current testing activities.

Scenario 5: "NewTool" Developing a new comprehensive tool which covers activities of both testing environment and removing their disadvantages. In this scenario, all efforts will be on creating a new testing environment including all recognized requirements in the radio unit software testing.

Table 5.1 represents how four different classes of testers (which are classified earlier) found the proper scenario in order to perform suitable testing in the sub-system level in addition to opinions of all respondents without classification.

	ClassWB	ClassBB	ClassBoth	ClassNew	All
Scenario1: KeepWB	8%	0%	0%	10%	6%
Scenario2: KeepBB	11%	33%	0%	5%	10%
Scenario3: Merge	38%	45%	58%	33%	41%
Scenario4: KeepBoth	38%	0%	42%	38%	34%
Scenario5: NewTool	0%	0%	0%	14%	4%
No Judgment	5%	22%	0%	0%	5%

Table 5.1: How different classes of testers found the proper testing environment in sub-system level testing.

As it can be seen in the table1, Scenarios 3 and 4 which are explained earlier in this chapter have more supporters among other scenarios for

all contributors with 41% and 34% of respondents. This means that 75% of all respondents believe that these two environments cannot fulfill sub-system level testing individually and both are needed, but 41% of them think that integration of these two can make sub-system level more simpler and more suitable while 34% of testers are with leaving them same as the current conditions without integrating them two into one tool. Among prior WBTest experienced respondents, both scenarios 3 and 4 gained equally 38% of participants. This means they believe both environments are needed (76%) but none of the scenarios 3 and 4 has superiority over another one. The next class contains prior BBTest experienced testers, intriguingly, none of them supports scenarios 4 and 45% of them chose scenario 3 which is merging these two environments into one. In addition, scenario 2 achieved 33% of responses that means 33% of prior BBTest experienced testers distinguished the BBTest environment as the most suitable environment to fulfill sub-system level testing goals. Responses for both environments experienced users are distributed between scenario 3 and scenario 4 with 58% and 42% respectively. It declares that they think both environments are needed but most of them (58%) believe that these environments should not be integrated into one and on the other hand, 42% of them responded to integration of two environment into one. Finally, new testers who did not test in the sub-system level are mostly with scenarios 3 and 4 with 33% and 38% of responders respectively. In the other word, 71% of contributors think both of these environments are needed but those who support integration are 5% more than non-integration supporters. The interesting point that can be found in responses of this class is that only 14% of them think that new environment should be developed and current environments are not suitable for sub-system level testing. It should be mentioned that except this class of respondents, none of other participants supports scenario 5.

In table 5.2, opinions of respondents about fulfilling sub-system level goals can be seen with respect to their prior experiences before joining cross functional teams.

Table 5.2 represents that only prior WBTest experienced testers (9% of them) distinguished fulfillment of the sub-system level testing goals in the WBTest environment is "Very good" which is only one twentieth of responses of all participants. Interesting point is, about twice of all contributors think that the BBTest has "Good coverage" than for the

	Class1	Class2	Class3	Class4	All
WBTest Very good coverage	9%	0%	0%	0%	4%
BBTest Very good coverage	9%	44%	17%	0%	10%
WBTest Good coverage	21%	11%	33%	19%	22%
BBTest Good coverage	35%	33%	58%	43%	42%
WBTest Moderate coverage	53%	22%	42%	43%	44%
BBTest Moderate coverage	32%	22%	8%	29%	25%
WBTest Weak coverage	15%	11%	17%	14%	14%
BBTest Weak coverage	6%	0%	8%	10%	6%
WBTest Little experience	3%	56%	8%	24%	13%
BBTest Little experience	18%	0%	8%	19%	13%

Table 5.2: How these two environments fulfill sub-system level testing goals.

WBTest. This trend is reverse for option "Moderate coverage", the BBTest environment response rate is roughly half for the WBTest among all participants without classifications. But, this trend can be seen also among four classes of testers. In the other word, similar trends for options "Good coverage" and "Moderate coverage" are distinguished. It is also considerable that almost half of the prior BBTest experienced users are stated that the BBTest fulfills sub-system level testing goals "Very good" and for both environments experienced respondents, this value is amost one fifth of responses. For the "Weak coverage" option, responses tend to WBTest environment comparing to the BBTest for all participants. Similar responses can be seen for these four classes of testers.

The complement result of evaluation, based on different effective criteria from second questionnaire is depicted in figure 5.1 . In addition, table 5.3 indicates standard deviation of the grades for each criteria.

As it depicted in figure 5.1, grades of stability and usability for these two environments are almost similar, but the WBTest achieved slightly higher grades with 7.54 for stability and 6.68 for its usability comparing to the BBTest environment with 7.07 and 6.21 for its stability and usability respectively. Although the difference is not so much but we can see that stability and usability in the WBTest environment is slightly better than in

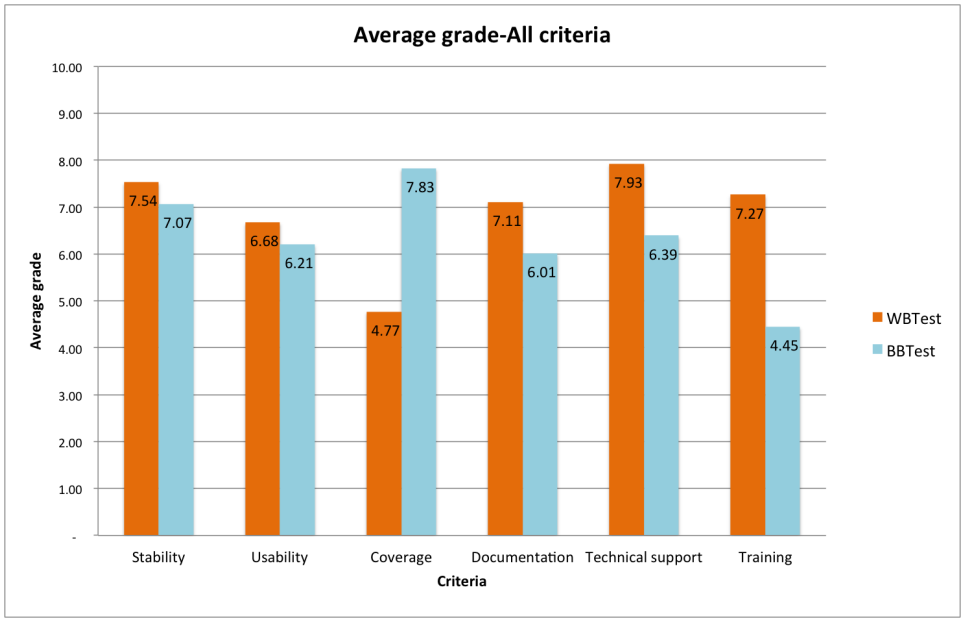

Figure 5.1: Average grade of different criteria for the WBTest and BBTest environments.

the BBTest environment. This trend is completely reverse in specification coverage criterion. In this criterion, the BBTest environment gained almost twice respondents than the WBTest environment. The BBTest with 7.83 covers more predefined specification in comparison to only 4.77 for the WBTest environment. Furthermore, the results show that in three other criteria (Documentation, Technical support and Training), the WBTest environment is quite better comparing to the BBTest environment. In documentation, the WBTest gained average grade of 7.11 and the BBTest got 6.1. In addition, in technical support, the WBTest environment received 7.93 (which is highest grade for the WBTest environment among other criteria) in comparison to 6.93 for the BBTest environment. The trend is similar for training but the difference is more than documentation and technical support (Training average grades: 7.27 for the WBTest and 4.45 for the BBTest). Furthermore, it should be noticed that the least grade for the BBTest is in training with only 4.45 and the least grade for the WBTest is in specification coverage with only 4.77. As it explained, the BBTest can be considered better only in the specification coverage and in the other five criteria the WBTest seems to be better and the most noticeable difference is in specification coverage with 7.83 for the BBTest and 4.77 for the WBTest environment and second most considerable difference is in the training which the WBTest has larger average grade than the BBTest. The standard

deviations of responses which are stated in the table3 indicate that there is no large polarization in the results and largest standard deviations are in usability for the WBTest with 1.77 and in technical support with 1.66 for the BBTest. The second largest standard deviation in WBTest is for documentation with 1.57 and in the BBTest, the second largest one is for stability with 1.66. These standard deviations with less than 1.5 for most of the criteria in these two environments show that grades of these criteria do not have considerable fluctuations. In addition, standard deviations in four criteria of these two environments is more than 1.5 (and less than 1.8). This shows that in all evaluated criteria, the polarization of grades are not so large and most of the respondents have almost similar assessments on these environments for these six selected criteria.

	Stability	Usability	Coverage	Document	Tech.Support	Training
WBTest	1.38	1.77	1.16	1.57	1.19	1.16
BBTest	1.62	1.40	1.12	1.35	1.66	1.24

Table 5.3: Standard deviation of grade in different criteria for the WBTest and BBTest environments.

5.2 Analytical Hierarchy Process (AHP)

In order to compare these two environments based on the six mentioned criteria, it is decided to use the analytical hierarchy process method. This section includes details calculations and ends up with rankings of these two environments. As it described in the approach chapter, the AHP contains four steps as follow:

1. Decomposing.
2. Weighting.
3. Evaluating.
4. Selecting.

At first it is needed to perform decomposition which constructs the hierarchical structure for this multi criteria decision making problem. Three

levels of hierarchy is created as depicted in figure 5.2.

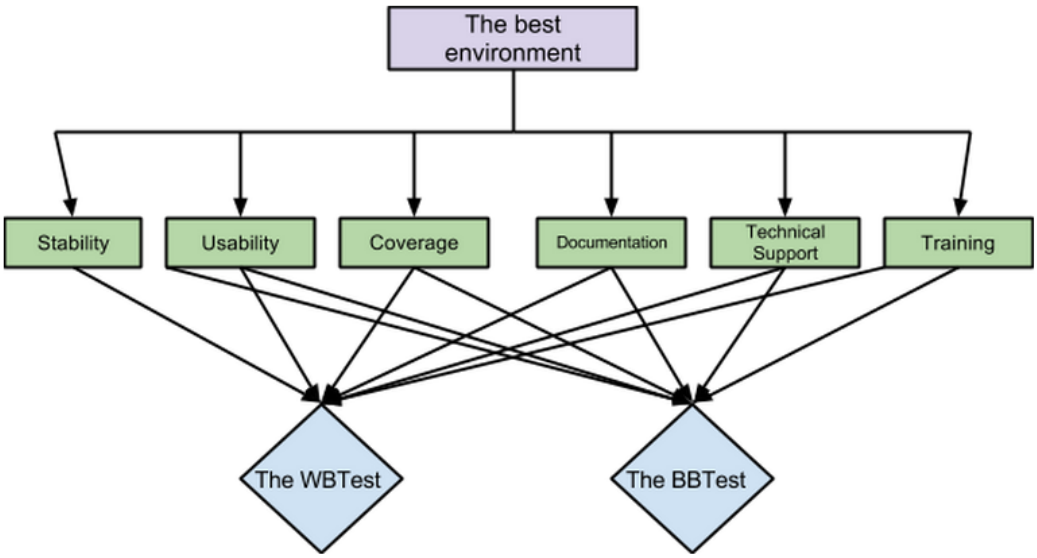

Figure 5.2: Analytical hierarchy structure in the AHP method for testing environments.

In figure 5.2, the first level is the goal of the AHP which is choosing the most appropriate environment among these two alternatives (the WBTest and BBTest). The alternatives are located at the third level of hierarchy and the middle level in the structure contains six effective criteria (Stability, Usability, Coverage, Documentation, Technical support and Training) for comparison.

The second phase in the AHP is weighting. Saaty's table that is introduced in the approach chapter will be used to create matrices. It means, all values in these matrices are concluded from Saaty's table. Seven matrices are needed for comparing these two environments, one matrix for the criteria, six matrices for alternatives based on these six criteria. The values of these matrices are chosen from Saaty's pairwise comparison table based on the structured interviews and result of the second questionnaire. The process will be started by filling the matrices with the scales and followed by the corresponding calculations which is explained in detail in the approach chapter of this thesis. The criteria matrix is as follow:

$$
\begin{array}{c}
 \\
C_1 = Stability \\
C_2 = Usability \\
C_3 = Coverage \\
C_4 = Documentation \\
C_5 = Technicalsupport \\
C_6 = Training
\end{array}
\begin{array}{c}
\begin{array}{cccccc} C_1 & C_2 & C_3 & C_4 & C_5 & C_6 \end{array} \\
\left[\begin{array}{cccccc}
1 & 5 & 2 & 8 & 7 & 7 \\
1/5 & 1 & 1/4 & 4 & 3 & 4 \\
1/2 & 4 & 1 & 7 & 6 & 7 \\
1/8 & 1/4 & 1/7 & 1 & 1/4 & 1/4 \\
1/7 & 1/3 & 1/6 & 4 & 1 & 3 \\
1/7 & 1/4 & 1/7 & 4 & 1/3 & 1
\end{array}\right]
\end{array}
$$

The geometric mean for the criteria matrix will be calculated:

$$GM_{Criteria} = (m_{C_1}, m_{C_2}, ..., m_{C_6})^T = \left\{ \sqrt[6]{\prod_{j=1}^{6} r_{1j}} , \sqrt[6]{\prod_{j=1}^{6} r_{2j}} , ... , \sqrt[6]{\prod_{j=1}^{6} r_{6j}} \right\}$$

$$(m_{C_1}, m_{C_2}, ..., m_{C_6})^T = (3.9708, 1.1571, 2.8944, 0.2556, 0.6701, 0.4347)^T$$

After normalization, the priority vector will results. The normalized vector can be computed by dividing the elements of the geometric mean vector by the sum of the element in the geometric mean vector:

$$P_{criteria} = (p_{m1}, p_{m2}, ..., p_{m6})^T = \left\{ \frac{\sqrt[6]{\prod_{j=1}^{6} r_{1j}}}{\sum_{i=1}^{6} \sqrt[6]{\prod_{j=1}^{6} r_{ij}}} , \frac{\sqrt[6]{\prod_{j=1}^{6} r_{2j}}}{\sum_{i=1}^{6} \sqrt[6]{\prod_{j=1}^{6} r_{ij}}} , ... , \frac{\sqrt[6]{\prod_{j=1}^{6} r_{6j}}}{\sum_{i=1}^{6} \sqrt[6]{\prod_{j=1}^{6} r_{ij}}} \right\}$$

$$(p_{m1}, p_{m2}, ..., p_{m6})^T = (0.4232, 0.1233, 0.3084, 0.0272, 0.0714, 0.0463)^T$$

After the calculation of weightings (priorities) vector for the criteria matrix, the other six matrices should be built with respect to each of these criteria. Moreover, their priority values will be calculated in the similar manner as it was for criteria matrix. These six matrices as well as their calculated geometric means and normalized priorities vector are as follow:

$$M_{Stability} = \begin{matrix} \\ A_1 = WBTest \\ A_2 = BBTest \end{matrix} \begin{matrix} \begin{matrix} A_1 & A_3 \end{matrix} \\ \begin{bmatrix} 1 & 2 \\ 1/2 & 1 \end{bmatrix} \end{matrix} \quad GM_{Stability} = \begin{bmatrix} 1.4142 \\ 0.7071 \end{bmatrix} \quad P_{Stability} = \begin{bmatrix} 0.6666 \\ 0.3333 \end{bmatrix}$$

$$M_{usability} = \begin{matrix} \\ A_1 = WBTest \\ A_2 = BBTest \end{matrix} \begin{matrix} \begin{matrix} A_1 & A_3 \end{matrix} \\ \begin{bmatrix} 1 & 3 \\ 1/3 & 1 \end{bmatrix} \end{matrix} \quad GM_{usability} = \begin{bmatrix} 1.7320 \\ 0.5744 \end{bmatrix} \quad P_{usability} = \begin{bmatrix} 0.7509 \\ 0.2490 \end{bmatrix}$$

$$M_{coverage} = \begin{matrix} \\ A_1 = WBTest \\ A_2 = BBTest \end{matrix} \begin{matrix} \begin{matrix} A_1 & A_3 \end{matrix} \\ \begin{bmatrix} 1 & 1/6 \\ 6 & 1 \end{bmatrix} \end{matrix} \quad GM_{coverage} = \begin{bmatrix} 0.4082 \\ 2.4494 \end{bmatrix} \quad P_{coverage} = \begin{bmatrix} 0.1428 \\ 0.8571 \end{bmatrix}$$

$$M_{document} = \begin{matrix} \\ A_1 = WBTest \\ A_2 = BBTest \end{matrix} \begin{matrix} \begin{matrix} A_1 & A_3 \end{matrix} \\ \begin{bmatrix} 1 & 2 \\ 1/2 & 1 \end{bmatrix} \end{matrix} \quad GM_{doc.} = \begin{bmatrix} 1.4142 \\ 0.7071 \end{bmatrix} \quad P_{doc.} = \begin{bmatrix} 0.6666 \\ 0.3333 \end{bmatrix}$$

$$M_{tech.sup.} = \begin{matrix} \\ A_1 = WBTest \\ A_2 = BBTest \end{matrix} \begin{matrix} \begin{matrix} A_1 & A_3 \end{matrix} \\ \begin{bmatrix} 1 & 3 \\ 1/3 & 1 \end{bmatrix} \end{matrix} \quad GM_{tech.sup.} = \begin{bmatrix} 1.7320 \\ 0.5744 \end{bmatrix} \quad P_{tech.sup.} = \begin{bmatrix} 0.7509 \\ 0.2490 \end{bmatrix}$$

$$M_{training} = \begin{array}{c} \\ A_1 = WBTest \\ A_2 = BBTest \end{array} \begin{array}{c} \begin{array}{cc} A_1 & A_3 \end{array} \\ \begin{bmatrix} 1 & 4 \\ 1/4 & 1 \end{bmatrix} \end{array} \quad \begin{array}{c} GM_{training} \\ \begin{bmatrix} 2 \\ 0.5 \end{bmatrix} \end{array} \quad \begin{array}{c} P_{training} \\ \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix} \end{array}$$

As described in the approach chapter, the priority vectors will compose a matrix which is called the original AHP matrix and should be converted to the ideal AHP matrix by dividing each element of the original AHP matrix by the maximum value of the containing column. The two upcoming matrices are original and ideal AHP matrices:

Original AHP matrix:

$$\begin{array}{c} \\ A_1 \\ A_2 \end{array} \begin{array}{c} \begin{array}{cccccc} C_1 & C_2 & C_3 & C_4 & C_5 & C_6 \end{array} \\ \begin{bmatrix} 0.6666 & 0.7509 & 0.1178 & 0.6666 & 0.7509 & 0.8 \\ 0.3333 & 0.2490 & 0.8821 & 0.3333 & 0.2490 & 0.2 \end{bmatrix} \end{array}$$

Ideal AHP matrix:

$$\begin{array}{c} \\ A_1 \\ A_2 \end{array} \begin{array}{c} \begin{array}{cccccc} C_1 & C_2 & C_3 & C_4 & C_5 & C_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0.1335 & 1 & 1 & 1 \\ 0.5 & 0.3316 & 1 & 0.5 & 0.3316 & 0.25 \end{bmatrix} \end{array}$$

The final decision priorities will be concluded by multiplying the ideal AHP matrix to the criteria priorities vector which is calculated earlier.

$$
\begin{array}{c} \\ A_1 \\ A_2 \end{array}
\begin{array}{c}
\begin{array}{cccccc} C_1 & C_2 & C_3 & C_4 & C_5 & C_6 \end{array} \\
\left[\begin{array}{cccccc} 1 & 1 & 0.1335 & 1 & 1 & 1 \\ 0.5 & 0.3316 & 1 & 0.5 & 0.3316 & 0.25 \end{array}\right]
\end{array}
\times
\begin{array}{c} P_{criteria} \\ \left[\begin{array}{c} 0.4232 \\ 0.1233 \\ 0.3084 \\ 0.0272 \\ 0.0714 \\ 0.0463 \end{array}\right] \end{array}
\Rightarrow
$$

$$
\begin{array}{l} \\ \text{The WBTest environment:} \\ \\ \text{The BBTest environment:} \end{array}
\begin{array}{c}
\text{Relative priorities} \\
\left[\begin{array}{c} \\ 0.7325 \\ \\ 0.6097 \end{array}\right]
\end{array}
$$

By applying the AHP method with respect to six defined criteria and scaling them using Saaty's table with assistance of quantitative and qualitative results, the WBTest environment is preferred to the BBTest environment, if there is no other alternative to choose. The relative priority for the WBTest environment is 0.7325 in comparison to 0.6097 for the BBTest environment. This result shows that although the coverage of the WBTest environment is less than the BBTest environment, other factors such as stability, usability, documentation, technical support and training could cover this weakness. The scales for the pairwise comparisons were chosen with the cooperation of the interviewees which were responsible for framework of these two environments. Obviously, different scaling can result the different priorities. For instance, if the coverage scale is extremely higher in comparison to other criteria, it may reverse the final relative priorities of these two environments.

5.3 Evaluation of Scenarios

In this section, implementation feasibility of defined scenarios will be assessed. This evaluation is based on the documents and qualitative interviews. In addition, this analysis can be considered as the main assessment about these two testing environments in this thesis. It is tried to provide a fair overall outcome that shows the feasibility of each scenario. At first, the third scenario (It is attracted most of respondents in the first questionnaire. table 5.1) with its two sub-scenarios will be discussed. Then, the fourth scenario which got second place among other four scenarios will be evaluated. This part will be followed by describing two first scenarios in case of requiring to choose either the WBTest or the BBTest. It should be added that fifth scenario will not be described because of obvious huge cost of its development and implementation, since all process of development have to be done from scratch.

From technical point of view, it is possible to implement the scenario 3 which is integration of these two environments into one. In fact, from framework perspective the cost is not that high. That means, it is possible to turn the WBTest into the BBTest and vice versa. The noticeable cost is porting all properties like test cases from one to another one and usage of them into place. If it is demanded for something like a merged environment which is an environment that can fulfill both test activities, it should be started all over to port all test cases because of different languages and properties in these two environments. Moreover, the coverage in the new integrated environment should also be increased. Although these environments can be integrated into one comprehensive environment, their assigned test activities should be performed and cannot be skipped. There is a good lesson from history of replacing old WBTest environment with current WBTest environment. Since the new WBTest environment is introduced which was two years ago, all test cases have not been ported to the new WBTest environment from old one yet and there are still some important tests which being done in the old one. The huge cost in the integration is porting already written test cases from one environment to another one by considering different implemented languages as well as different databases. Furthermore, if we assume the WBTest as foundation of new environment (Scenario 3-1) and adding all functional verification using all external instruments which the BBTest already has, it would add a new abstraction layer for controlling these instruments which would be

huge job to be defined and proper test cases be written to use these external measurement instrument. It could have same language and be started in the same way, but it would still have to have separate test objects because they designed for aiming very different purposes. Let say it is rather to use new merged environment for sub-system level integration which is testing against design specification, then it will be same way as the current WBTest environment fashion testing and also if tester want to use new tool for sub-system level verification, then it will be the same method of the BBTest testing which has heavy abstraction layer with all instruments and equipment in the BBTest environment. In the simple word, for opening a door to a car, maybe as integration test activity, you would like to find out whether not only the lamp inside is lighted up but also something that happened inside the software, while as you do it for verification, you open the door and light goes on and that is it and you do not care about what is going on inside the software. As it can be seen, they are in two different scopes. But from technical aspect, it is possible to develop and implement drivers and other properties for instrument and measurement layers. This can be explained more with a complement example as follow: Just imagine to use a big electric screwdriver to unscrew a little screw on your watch. Obviously, you should use proper tool when needed. Is an electric screwdriver needed as the same time as small hand driven screwdriver? The answer is yes. Because they have completely different scopes. The electric screwdriver is also needed whenever it is applicable.

On the other hand, for integrating the WBTest environment into the BBTest which is stated as scenario 3-2, the similar process should be followed, the test cases should be rewritten in the BBTest environment with respect to different implementation languages and different level of abstraction. In fact, there is another matter which is how high is the abstraction level. In the WBTest the level of abstraction is so higher than it is in the BBTest. In the BBTest, most of the test cases are following the action-result format, but in the WBTest it is higher level API. On the same hand, the WBTest has lower level concept, because it has interfaces to all boards to read registers and sensors which is only can be tested by the WBTest at the moment.

It should be emphasized, that the cost is much more higher to port test cases in the BBTest environment into the WBTest than opposite. There are five to six hundred test cases with respect to design specifications by now and on the other hand, around 13000 test case scripts are already existed in the BBTest environment with respect to functional specifications

document. Since the BBTest environment is running for couple of years, so more test cases with respect to requirements have been developed during these years.

The scenario that indicates to keep these two environments without integration is called scenario 3. It indicates to keep both environments but make them look and feel the same and adding benefits of one to another one. As an example, the BBTest can be run into the well designed WBTest lab, a test device can be booked and results can be presented in the same way as the WBTest in which it is so comfortable to follow results, reports and logs. But when it comes to write test case, different purposes of both integration and verification testings should be noticed. It seems that huge missed point is a proper test plan for test activities in these two environments. The testers should know what should be tested in which environment. They should consider which environment is suitable for testing against which specification. This scenario can be considered as least costly scenario but needed proper efforts. It is needed to have an adequate test plan to inform testers that which testing environment should be selected to perform a specific test. As it seen in the documents, there is a test strategy which is in high level and shows a bigger picture of testing. A test plan for specific part of testing (i.e. sub-system level) has to be implemented and deployed. There is a step which is called continuous analysis which contains some sort of test plan but it is not so visible. In the whole concept, there are some reports called overall test analysis report (OTAR) and detail test analysis report (DTAR). Up until lately, in these reports, the node level testing is considered. Obviously, all test activities should be inserted into OTAR and DTAR. It should be noticed that this kind of plan can be broken down in the agile way of development which is used in the radio software development section. Despite all difficulties, a proper analysis have to be performed before each step in the radio software development department and specially before sub-system level testing. In fact, cross functional teams are very confused. They cannot see relations between steps and way forward as well as how to avoid duplicated tests and overlaps. This should be done in the early analysis. It means when testers start execution over certain feature or specification, they should know where to put their efforts in different phases, environments and tools.

Keeping one environment for sub-system level testing, either the WBTest environment or the BBTest environment are stated as scenarios 1 and 2. If only one test environment is going to be kept, the framework of the BBTest

known as the BBTest core is more capable of doing most of the sub-system level testing activities, since the WBTest environment is mainly suited for with heavy adaptation toward measurement equipments, instruments and drivers. The BBTest has more built in keys to answer sub-system level testing. The BBTest core knows what the radio unit should reply when it is asked about capabilities by checking the XML database which is only available in the BBTest environment and it is needed to do formal verification. On the other hand, the WBTest is more lightweight and easier to use for integration testing which does not care about thing that has to be so formal. If there was only one choice at the moment without any technical modification in the platforms, the choice has to be the BBTest environment because the WBTest environment does not support formal verification which is absolutely important in the sub-system level testing. Moreover, the WBTest environment is not that much mature as the BBTest is and there is still room for progress in the WBTest environment since it is newer than the BBTest environment in the sub-system level. Finally, there is a big risk in removing test activities and further problem will be raised which will cost more to solve later.

At the first glance, there are some test cases which are overlapped in these two testing environments, but further investigations show that although they may be seen overlapped and duplicated, they are designed for two different aims. Figure 5.3 represents a simple example for such occasions. The radio unit is imagined as it depicted in figure 5.3. It has two components including their registers as well as two interfaces, one for input and another one for output. As it can be seen, from a black box point of view which sees the radio unit as black box and does not care about inside components and processes, there is an input signal which is inserted by a test case (in the BBTest environment) in order to test if the radio unit software produces expected output signal or not. This can be checked by the predefined functional specification and its database. If the output signal is as expected, this black box test is passed otherwise it will fail. This process can be performed for all different test cases in the BBTest environment for different requirements. In this case, we can assume that if all tests would be passed, the radio unit software contains no errors and completely passes predefined requirements of radio unit software. In addition, we can suppose there is no need for extra white box tests, since the radio unit behaves as expected based on the requirements and functional specifications. On the other hand, from white box point of view, if the input signal would need to set a register or flag when it passes

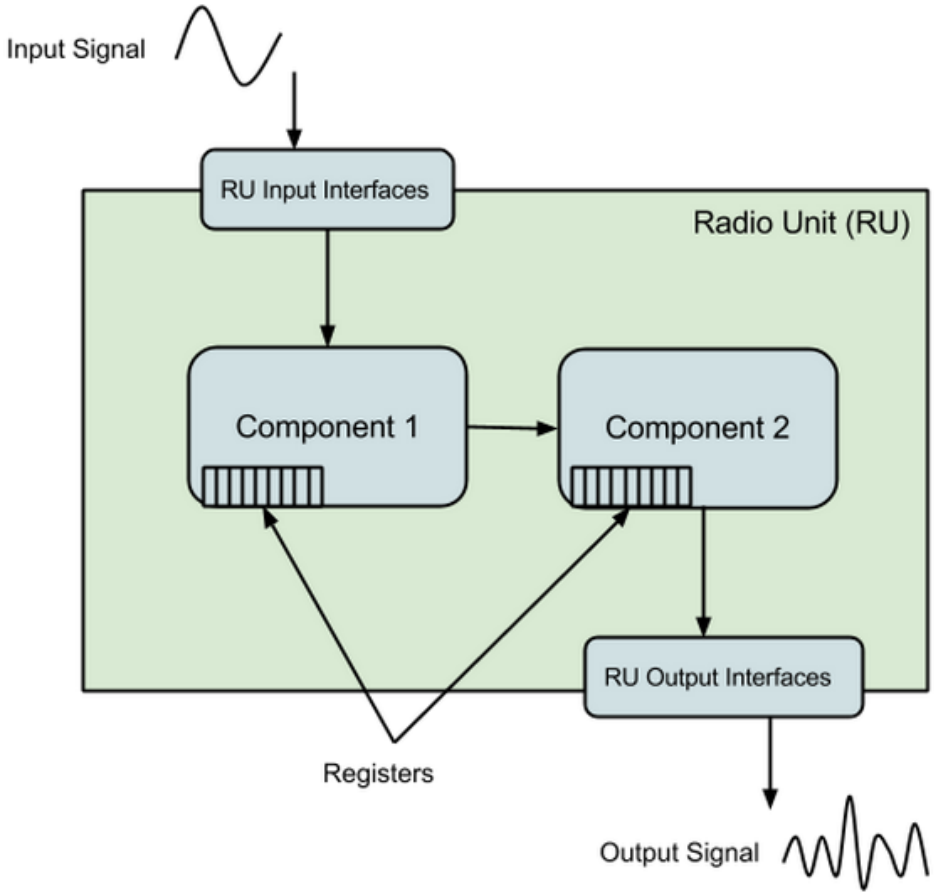

Figure 5.3: Radio Unit schema with two simple components in addition to their registers

component 1, how it can be checked? Obviously, all black box tests are passed, even though this register is not set, but this specific register did not make any problem for passing signal and the output is as expected. The problem may occur later on. An hour or a day later it may cause some crucial problems for this radio unit. It may lead to signal failure, fire and other problems. This issue can be solved with white box testing aspect. The WBTest environment has facilities to test these internal interfaces. Actually, a test case will be executed to test if such register is set or not. The white box aspect complement the radio unit software testing. It will assure that the internal properties are also working as they expected. As consequence, both aspects are needed for testing radio unit software properly.

Chapter 6

Conclusion

The problem statement for this thesis was to compare two radio unit software testing environments which use black- and white-box testing approaches. During this thesis, it is tried to answer the questions which are stated in the problem statements. Since these two testing environments, the WBTest and BBTest environments were unknown to the author of this, it was decided to perform quantitative survey based on responses from the testers that were dealing with these two environment and were aware of pros and cons in the company in addition to the qualitative part which included interviews and internal documents. In the upcoming list, answers to the corresponding questions made in problem statement section 1.2 are supplied.

- As explained in the analysis chapter, if only one test environment is going to be kept, the framework of the BBTest, the BBTest core, is more capable of doing most of the sub-system level testing activities, since the WBTest environment is mainly suited for heavy adaptation toward measurement equipments, instruments and drivers. The BBTest has more built in keys to answer sub-system level testing. The BBTest core knows what the radio unit should reply when it is asked about capabilities by checking the XML database which is only available in the BBTest environment and it is needed to do formal verification. On the other hand, the WBTest is more lightweight and easier to use for integration testing, for which the environment does not need to de as formal. If there was only one choice at the moment without any technical modification in the platforms, the choice has to

be the BBTest environment because the WBTest environment does not support formal verification which is crucial in the sub-system level testing. It should be noticed that relying on one of these environment can lead to missing some important test activities and by relying on the WBTest environment this loss can be huge in comparison to the BBTest environment.

- By applying the analytical hierarchy process (AHP) method with respect to six defined criteria and scaling them using Saaty's table with assistance of quantitative and qualitative results, it was possible to grade these two environments. The AHP is based on sets of pairwise comparisons to obtain relative weights of individual criterion and rating alternatives with respect to each criterion. Additionally, it has a great advantage, the consistency ratio (CR), with the intent of evaluating the adequacy of the initial input and as the consequence the final priorities. After the corresponding calculations, the WBTest achieved higher priority than the BBTest, mainly because of the higher scales in the pairwise comparisons which the WBTest obtained in five criteria except in the coverage.

- This question about the merging the of the BBTest and WBTest, was defined as one of the scenarios. From a technical point of view, it seems to be possible to implement an integrated test environment. In fact, from a framework perspective the cost is not that high. That means, it is possible to turn the WBTest into the BBTest and vice versa. The noticeable cost is porting all properties like test cases from one to another one. If it is demanded for something like a merged environment which is an environment that can fulfill both test activities, that it should be started all over to port all test cases because of different languages and properties in these two environments. In the case of integration, two possible scenarios would happen, that either the WBTest or the BBTest environment could be the basis of integration. Since porting test cases was assumed main cost of integration and the BBTest environment has already almost twenty times more written test cases than the WBTest environment, obviously considering the BBTest as basis is cheaper and needs less resources. It should also be noted, that the other technical properties of each of them that is chosen, should be moved to another one as well.

- At the first sight, there are some redundancies in these two testing

environments, but the investigations up until finishing this report shows that although they may be seen to overlap, they are designed for two different aspects. In other words, they may test the same thing at the first glance, but their scopes are different and if testers remove one of the redundant test cases, it may cause further issues, although it could pass the other test case which is in another approach (either black- or white box approaches).

- Scientific and systematic testing methodology used in this part of the IT industry could obviously readily be translated and applied to systems normally controlled by network and system administration. It is needed to spend more time and also extend the ideas and experiences which are gathered during this thesis into the network and system administration which requires further attention to testing in order to improve the quality of services and products.

For the future work, it is recommended to apply what are learned from this thesis to the systems such as cloud computing, intrusion detection systems and more, in order to fulfill SLAs which are often neglected by the network and system administrators. Additionally, further investigations might be needed for the company running these two testing systems to uncover more detailed overlaps in the test activities of these two testing environments.

Bibliography

[1] R. Agarwal and D. A. Umphress, "A flexible model for simulation of software development process," in *ACM Southeast Regional Conference*, p. 40, 2010.

[2] D. Bell, I. Money, and J. Pugh, *Software Engineering: A Programming Approach*. Prentice-Hall, 1987.

[3] C. V. Ramamoorthy, A. Prakash, W. Tsai, and Y. Usuda, "Software engineering:problems and perspectives," *Computer*, vol. 17, no. 10, pp. 191–209, 1984.

[4] W. Chantatub, *The Integration of Software Specification,Verification, and Testing Techniques with Software Requirements and Design Processes*. PhD thesis, University of Sheffield, March 1995.

[5] T. Saaty, "Decision making with the analytic hierarchy process," *Int. J. Services Sciences*, vol. 1, no. 1, pp. 83–98, 2008.

[6] T. Parveen and S. Tilley, "When to migrate software testing to the cloud?," *in Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, ICSTW '10, (Washington, DC, USA)*, vol. IEEE Computer Society, pp. 424–427, 2010.

[7] J. Gao, X. Bai, and W. Tsai, "Cloud testing- issues, challenges, needs and practices," *Software Engineering: An International Journal*, vol. 1, pp. 9–23, 2011.

[8] L. M. Riungu, O. Taipale, and K. Smolander, "Research issues for software testing in the cloud," *in Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, CLOUDCOM '10, (Washington, DC, USA)*, vol. IEEE Computer Society, pp. 557–564, 2010.

[9] L. van der Aalst, "Software testing as a service (staas)," *Sogeti Whitepaper*.

[10] W. Chan, S. Cheung, and K. Leung, "A metamorphic testing approach for online testing of service-oriented software applications," *Int. J. Web Service Res.*, vol. 4, no. 2, pp. 61–81, 2007.

[11] Priyanka, I. Chana, and A. Rana, "Empirical evaluation of cloud-based testing techniques: A systematic review," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 3, 2012.

[12] P. Ralph and Y. Wand, "A proposal for a formal definition of the design concept," *Design Requirements Engineering: A Ten-Year Perspective: Springer-Verlag*, pp. 103–136, 2009.

[13] A. Sharma and D. S. Kushwaha, "A metric suite for early estimation of software testing effort using requirement engineering document and its validation," *International Conference on Computer and Communication Technology (ICCCT)*, 2011.

[14] B. W. Boehm, "Software engineering," *IEEE Transactions on Computers*, vol. C-25, pp. 35–50, December 1976.

[15] I. Shemer, "Systems analysis: A systemic analysis of a conceptual model," *Computing Practices*, vol. 30, pp. 506–512, June 1987.

[16] R. Hamlet, "Randomtesting," *Encyclopedia of Software Engineering New York:Wiley*, pp. 970–978, 1994.

[17] B. Cheng-Gang, J. Chang-Hai, and C. Kai-Yuan, "A reliability improvement predictive approach to software testing with bayesian method," *Proceedings of the 29th Chinese Control Conference*, July 2010.

[18] *Software Testing Overview - Tutorials Point*.

[19] G. Myers, *The Art of Software Testing*. Wiley, 1979.

[20] R. S. Pressman, *Software Engineering, A Practitioner's approach*. McGraw-Hill, 5th ed.

[21] A. Davis, *201 Principles of Software Development*. McGraw-Hill, 1995.

[22] C. Kaner, J. Falk, and H. Nguyen, *Testing Computer Software*. Van Nostrand-Reinhold, 1993.

[23] T. McCabe, "A complexity measure," *IEEE transaction on software engineering*, vol. se-2, December 1976.

[24] B. Beizer, *Software Testing Techniques*. Van Nostrand-Reinhold, 2nd ed., 1990.

[25] K. Tai, "What to do beyond branch testing," *ACM Software Engineering Notes*, vol. 14, pp. 58–61, April 1989.

[26] M. Deutsch, *Verification and Validation in Software Engineering*. Prentice-Hall, 1979.

[27] L. White and E. Cohen, "A domain strategy for program testing," *IEEETrans. Software Engineering*, vol. SE-6, no. 5, pp. 247–257, 1980.

[28] W. Howden, "Weak mutation testing and the completeness of test cases," *IEEE Trans. Software Engineering*, vol. SE-8, pp. 371–379, July 1982.

[29] B. Beizer, *Black-Box Testing*. Wiley, 1995.

[30] M. Phadke, "Planning efficient software tests," *Crosstalk*, vol. 10, no. 10, pp. 11–15, 1997.

[31] B. Boehm, *Software Engineering Economics*. Prentice-Hall, 1981.

[32] D. Wallace and R. Fujii, "Software verification and validation: An overview," *IEEE Software*, pp. 10–17, 1989.

[33] E. Miller, "The philosophy of testing in program testing techniques," *IEEE Computer Society Press*, pp. 1–3, 1977.

[34] J. Musa and A. Ackerman, "Quantifying software validation: When to stop testing?," *IEEE Software*, pp. 19–27, 1989.

[35] S. Campodonico, "A bayesian analysis of the logarithmic-poisson execution time model based on expert opinion and failure data," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 20, September 1994.

[36] R. Savenkov, "How to become a software tester," *Roman Savenkov Consulting*, p. 386, 2008.

[37] T. L. Graves, M. J. Harrold, J. min Kim, A. Porter, and G. Rothermel, "An empirical study of regression test selection techniques," *ACM Transactions on Software Engineering and Methodology*, vol. 10, no. 2, pp. 184–208, 2001.

[38] J. McCarthy, *Dynamics of Software Development*. Microsoft Press, 1995.

[39] B. Beizer, *Software System Testing and Quality Assurance*. Van Nostrand-Reinhold, 1984.

[40] ISTQB, *Standard glossary of terms used in software testing*. ISTQB, http://www.istqb.org/downloads/finish/20/14.html, version 2.1 ed., 2010.

[41] ISTQB, *Certified Tester, Advanced level Syllabus*. ISTQB, http://www.istqb.org/downloads/finish/3/2.html, 2007 ed., 2007.

[42] CDM, "Company's internal document web page.,"

[43] http://www.ruby lang.org/en/.

[44] http://www.rspec.info/.

[45] http://www.cpri.info/.

[46] http://www.perl.org/.

[47] J. W. Creswell, *Research Design: Quantitative, Qualitative, Mixed Methods Approaches*. Thousand Oaks, CA: Sage Publication, 3rd ed., 2008.

[48] J. Morse, "Approaches to qualitative and quantitative methodological: Triangulation. qualitative research," no. 40, pp. 120–123, 1991.

[49] I. Newman and C. R. Benz, "Qualitative-quantitative research methodology: exploring the interactive continuum," *Carbondale and Edwardsville: Southern Illinois University press*, 1998.

[50] T. L. Saaty, "A scaling method for priorities in hierarchical structures," *Journal of Mathematical Psychology*, no. 15, pp. 57 – 68, 1977.

[51] D. Power, "Decision support systems glossary," *http://www.dssresources.com/glossary/dssglossary1999.html*.

[52] R. AL-QUTAISH, M. MUHAIRAT, and B.M.AL-KASASBEH, "The analytical hierarchy process as a tool to select open source software," *Proceedings of the 8th WSEAS Int. Conference on SOFTWARE ENGINEERING, PARALLEL and DISTRIBUTED SYSTEMS*.

[53] E. Huizingh and H. Vrolijk, "Decision support for information systems management: Applying analytic hierarchy process," *Universiteitsbibliotheek Groningen*, vol. Research Report 95B26, 1995.

[54] L. Santillo, "Early fp estimation and the analytic hierarchy process," *in Proceedings of the ESCOMSCOPE Conference, Munich, Germany*, pp. 249–257, 2000.

[55] J. Soininen, S. Boumard, T. Salminen, and H. Heusala, "Application of decision-making method for architecture selection of adsl modem," *in Proceedings of the Euromicro Symposium onDigital Systems Design, Warsaw, Poland*, pp. 21–28, 2001.

[56] C. S. Grewal, "A multi criteria logistics- outsourcing decision making using the analytic hierarchy process," *International Journal of Services Technology and Management*, vol. 9, no. 1, pp. 1–13, 2008.

[57] J. Lewe, B. Ahn, D. A. Delaurentis, D. N.Mavris, and D. P. Schrage, "An integrated decision-making method to identify design requirements through agent-based simulation for personal air vehicle system," *in the AIAA Aircraft Technology ,Integration, and Operation (ATIO) Technical Forum, Los Angeles, CA*, 2002.

[58] L. Mikhailov and P. Tsvetinov, "Evaluation of services using a fuzzy analytic hierarchy process," *Applied Soft Computing*, vol. 5, no. 1, pp. 23–33, 2004.

[59] R. Febriamansyah, "the use of ahp (the analytic hierarchy process) method for irrigation water allocation in a small river basin (case study in tampo river basin in west sumatra, indonesia)," *11th Conference of the International Association for the Study of Common Property, Bali, Indonesia*, 2006.

[60] J. Noh and K. M. Lee, "Application of multiattribute decision-making methods for the determination of relative significance factor of impact categories," *Environmental Management*, vol. 31, no. 5, pp. 633–641, 2003.

[61] C. Alves and A. Finkelstein, "Challenges in cots decision-making: A goal-driven requirements engineering perspective," *n Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy*, pp. 789–794, 2002.

[62] J. D. Kendrick and D. Saaty, "Use analytic hierarchy process for project selection," *Six Sigma Forum Magazine*, vol. 6, no. 8, pp. 22–29, 2007.

[63] K. Eldrandaly, "Gis software selection: A multicriteria decision making approach," *Applied GIS*, vol. 3, no. 5, pp. 1–17, 2007.

[64] A. Koscianski and J. C. B. Costa, "Combining analytical hierarchical analysis with iso/iec 9126 for a complete quality evaluation framework," *in Proceedings of the 4th IEEE International Symposium and Forum on Software Engineering Standards, Curitiba, Brazil*, pp. 218–226, 1999.

[65] I. Dikmen and M. T. Birgonul, "An analytic hierarchy process based model for risk and opportunity assessment of international construction projects," *Canadian Journal of Civil Engineering*, vol. 33, no. 1, pp. 58–68, 2006.

[66] G. E. Pavlikakis and V. A. Tsihrintzis, "Evaluation of three multi-criteria decision-making methods in ecosystem management," *in Proceedings of the 8th International Conference on Environmental Science and Technology, Lemnos Island, Greece*, pp. 667–674, 2003.

[67] S. Nataraj, "Analytic hierarchy process as a decision-support system in the petroleum pipeline industry," *Issues in Information Systems*, vol. 4, no. 2, pp. 16–21, 2005.

[68] V. Belton and T. Gear, "On a short-coming of saaty's method of analytic hierarchies," *Omega*, pp. 228 – 230, 1983.

[69] T. L. Saaty, "Fundamentals of decision-making and priority theory with the ahp," *WS Publications, Pittsburgh, PA, USA.*, 1994.

[70] T. Saaty, "The analytic hierarchy process," *McGraw-Hill International , New York, NY, USA*, 1980.

[71] J. M. X. Biyang, "A qualitative and quantitative assessment method for software process model," *IEEE*, 2010.

[72] C. Chiang, "Software stability in software reengineering in information reuse and integration," *IEEE International Conference, IEEE Xplore: Arkansas University.*, 2007.

[73] S. Yau and J. Collofello, "Some stability measures for software maintenance," *IEEE Transactions On Software Engineering*, pp. 545–552, 1980.

[74] J. Nielsen, "Usability 101: Introduction to usability," *http://www.nngroup.com/articles/usability-101-introduction-to-usability/*, 2012.

[75] M. Harder, B. Morse, and M. Ernst, "Specification coverage as a measure of test suite quality," *MIT lab for computer science*, 2001.

[76] J. Goodenough and S. L. Gerhart, "Correction to "toward a theory of test data selection"," *IEEE Transactions on Software Engineering*, vol. 1, no. 4, 1975.

[77] D. Richardson, O. O'Malley, and C. Tittle, "Approaches to specification-based testing," *Proceedings of the ACM SIGSOFT '89 Third Symposium on Testing, Analysis, and Verification (TAV3)*, pp. 86–96, 1989.

[78] J. Chang, D. Richardson, and S. Sankar, "Structural specification-based testing with adl," *ISSTA*, pp. 62–70, 1996.

[79] A. Offutt and S. Liu, "Generating test data from sofl specifications," *The Journal of Systems and Software*, vol. 49, no. 1, pp. 49–62, 1999.

[80] LINFO, "Linux information project," *http://www.linfo.org/documentation.html*.