DEVELOPMENT OF AN AUTOMATED COMPILATION TEST SYSTEM FOR EMBEDDED SYSTEM TESTING

OOI JUN HWAN

UNIVERSITI SAINS MALAYSIA

2016

DEVELOPMENT OF AN AUTOMATED COMPILATION TEST SYSTEM FOR EMBEDDED SYSTEM TESTING

By

OOI JUN HWAN

A Dissertation submitted for partial fulfilment of the requirement for the degree of Master of Science (Electronic Systems Design Engineering)

August 2016

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Dr Mohamad Khairi Ishak for his guidance and advice throughout the whole research. He has provided me a lot of useful information which I can use to complete my dissertation. I really appreciate his patience, suggestion, and enthusiasm during several meetings that we had. His fruitful and valuable advices have been helped a lot in developing ideas for my research.

Besides, I would like to thank my direct manager, Mr. Koay Kai Chong for his continuous support and understanding which encourage me to fully utilize my time for both working task and research development. I have gained a lots from his personal input on the development of this test methodology as well.

On the other hand, I would like to thank my mentor, Mr. Ong Kein Wei for his advice and comment on helping me to produce a quality result in my research. His willingness to share his experience on this research has contributed in enhancing the overall process of test methodology development.

On personal note, I would like to thank my family members especially my parent who are always supporting me and keep me motivated on completing my research. Lastly, I would like to thank all of my classmates for their help and encouragement during this research development.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

WPF	Windows Presentation Foundation
RTOS	Real-Time Operating System
HAL	Hardware Adaptation Layer
IDE	Integrated Development Environment
DUT	Device Under Test
ISC	In-System Configuration
PLDs	Programmable Logic Devices
TAP	Test Access Port
TFCL	Test Flow Control Language
HBD	Hardware Block Diagram
HSBD	Hardware Software Block Diagram
FIT	Fault Injection Target
DPPS	Digital Plant Protection System
VP	Verification Pattern
UML	Unified Modeling Language
MDE	Model-Driven Engineering
PIM	Platform Independent Model
PSM	Platform Specific Model
UAV	Unmanned Aerial Vehicle
SSD	Solid State Drive
PDB	Program Database File
WOW64	Windows on Windows 64

PEMBANGUNAN SISTEM KOMPILASI SECARA AUTOMASI UNTUK PENGUJIAN SISTEM TERBENAM

ABSTRAK

Dalam pengujian sistem terbenam yang melibatkan ujian integrasi di antara perisian dan perkakasan, penilaian fungsi bagi setiap modul menjadi semakin sukar untuk dijalankan dalam masa yang singkat disebabkan oleh peningkatan jumlah ujian yang diperlukan. Kajian ini mencadangkan satu sistem kompilasi secara automasi melalui struktur reka bentuk untuk meningkatkan kualiti ujian dan mengoptimumkan masa ujian dengan bantuan automasi. Dalam erti kata lain, liputan ujian boleh dimaksimumkan dengan kerja manual dan masa ujian minima yang diperlukan. Sistem pengujian ini telah digunakan untuk mengautomasikan proces pembinaan kod ujian dan pelaksanaan ujian untuk modul perkakasan yang dihasilkan. Hasil analisis keputusan yang diperolehi dalam pelaksanaan sistem pengujian ini, terbukti bahawa sebanyak 56.42% masa pengujian telah dijimatkan di samping memastikan kualiti pengujian sistem ini. Kesimpulannya, kajian ini telah mencadangkan satu sistem kompilasi secara automasi yang membantu dalam penjimatan masa dan jaminan qualiti untuk pengujian sistem terbenam.

DEVELOPMENT OF AN AUTOMATED COMPILATION TEST SYSTEM FOR EMBEDDED SYSTEM TESTING

ABSTRACT

In the embedded system testing which involved the integration testing between both software and hardware, it becomes increasingly difficult to evaluate the functionality of each functional modules in a short time due to the increasing number of testing required. This research proposed an automated compilation test system through a design structure to improve the quality of testing and optimize the testing time using automation. In other words, the testing coverage can be maximized with minimum manual work and testing time required. This test system has been used to automate the test code compilation and execution for different hardware module testing. The result analysis from the implementation of proposed test system proved that there is a significant testing time saving for around 56.42% besides ensuring the performance and quality of the result. In summary, this research has proposed an automated compilation test system which contributes in time saving and quality assurance for test code compilation and execution process.

CHAPTER 1

Introduction

1.1 Background

This section provides a background overview for the whole research scope. Embedded system is an electronically controlled system with the integration of hardware and software. It consists of software application layers that utilize services provided by underlying system service and hardware support layers. A typical embedded system application consists of multiple layers and user tasks which having different functionality respectively. Hence, in the design of embedded system, the interactions among these different layers are playing very important roles. There are two important classes of embedded system, which are safety critical embedded system and technical scientific algorithm based embedded system [1]. Besides, host based embedded devices and target based embedded devices are also another sub classifications for embedded systems [1].

On the other hand, testing is an important process which is executed to identify any possible defects that are available in the system using different debugging

methodology. Testing needed to be run on embedded system to determine the functionality of the system and ensure the quality within the system development process [2]. In test plan, sole characteristics of embedded systems must be reflected as they are application specific systems that give embedded systems a testing exclusive and distinct flavor.

In this research, a testing methodology which consists of both black box and white box testing will be presented to help the testing of embedded systems. It can be used to automate and improve the quality of testing by reducing testing time but maximize the overall coverage. Furthermore, it can then help to prioritize the test case based on the test coverage and defect area to prepare the regression test.

1.2 Problem Statements

Error may occurs between the communication of both hardware and software as embedded system are consist of both hardware and software. Defects that are detected in the later stage of system development may affect the product quality, besides lead to higher cost of resolution. Therefore, it is important to develop a testing technique to detect any possible interaction faults that may occur at a minimum time.

On the other hand, software testing may include some physical process which requires user input to interact with the system during the testing. It will make the testing process more time consuming as these tests have to be executed manually with many test steps. By automating some of manual tests, it will help to reduce the time needed to execute the tests.

There are many different platforms for test code compilation in embedded system which need different types of test methodology [11]. Hence, development of a

generic test methodology that applicable to different platforms is proposed and developed to reduce the development cost with better maintenance.

1.3 Research Objectives

The objectives of this research project are as the following:

1. To develop a methodology on embedded system testing to improve the quality of testing and optimize the testing time using automation.
2. To implement the proposed methodology on different platforms of embedded system in order to utilize the generic characteristic of the method.
3. To investigate the performance of the quality testing from both software and hardware perspective.

1.4 Research Scope

This research scope will focus on the design structure of the proposed testing methodology that apply to different platforms of embedded system. Moreover, test methodology that can automate the test code compilation and execution to minimize the testing time is tested on actual hardware. Lastly, the result analysis is performed to discuss the contribution of the proposed methodology.

The limitation of this research is the test code development on the hardware and software driver are not included. Instead, with the pre-developed software driver for hardware module which include the test code development, the proposed test methodology is implemented and the overall improvement is analyzed.

1.5 Research Contribution

This research project contributes on improving the test code compilation time and execution time for test code developed on hardware driver using a software that run on embedded system. Besides, the test case prioritization is performed based on number of defects and execution time to help on test sequences review for regression test to improve the quality of the hardware driver.

The proposed methodology has been utilized in the example code testing for Keysight Technologies PXIe module software driver. After the implementation, the time period for Software Quality (SQ) testing has been successfully reduced and provide a better quality result.

1.6 Thesis Outline

In Chapter 2, the outcome of literature reviews is carried out and some background of embedded system testing has been described in detail. Besides, the analysis is performed on different research studies related to different types of embedded system testing methodology and their respective challenges.

In Chapter 3, the proposed methodology is described and discussed sequentially. Every stages involved in the methodology are explained in details by showing the example of implementation in actual cases. All of the functional modules that have been developed for this test methodology are further break down to provide a clearer description for each modules.

Chapter 4 presents the result obtained through experiment and application on actual hardware using the proposed test methodology. Next, the results are analyzed using table or graph analysis to obtain a better overview of the analysis. Besides, the

result analysis also helps to provide a comparison between existing method and the proposed method to show the contribution of this research.

Chapter 5 presents a conclusion by summarizing the overall structure of the proposed test methodology in embedded system testing. Next, the future work for this research is further described follow by the achievement of this research.

CHAPTER 2

Literature Review

2.1 Introduction

This chapter provides research study about embedded system and different types of testing methodology to understand this research and its related work. Section 2.2 discusses the background on embedded systems design, including both hardware and software interface. Section 2.3 describes about different testing methodology that can be applied in software testing such as black box testing and white box testing. Lastly, section 2.4 discusses on background of different testing techniques that have been developed and implemented on validating embedded systems.

2.2 Embedded System Design

Embedded system is typically designed with an assortment of software and hardware to perform specific tasks in particular computational environments. It is usually constructed with least powerful computers that can meet the functional and performance requirements. Embedded systems generally use microprocessors that combining multiple functions of a computer on a single device. Figure 2.1 shows the typical structure of embedded system in terms of four layers where the first three layers consisting of software and the fourth consists of one layer of hardware.

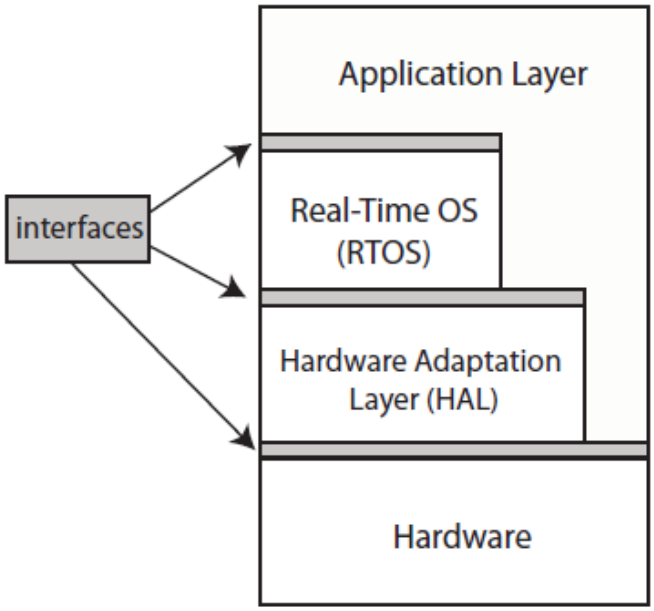

Figure 2.1: Structure of An Embedded System [3]

Two popular operating systems are being used which are Linux and Windows Embedded to enable the implementation of the embedded systems [12]. Most but not all of the embedded systems are real – time system in which the correctness of a operation not only depends on its accuracy of functionality but also depends on the timing of the produced result.

Embedded platforms are designed based on complex integrated system which involves different multi-tasking environments. Each component can perform several different tasks at the same time for a real-time embedded system. By doing so, the interaction across components are eventually increased in a dramatic way [13]. During the integration of different layers, if the tests running on the interactions between components are not fully tested, unexpected results may occur and lead to more critical problem.

Moreover, the temporal behavior is another important functional behavior in the real-time systems [37]. For real-time embedded system, all requests that are involved must be handled and completed in the allocated period of time after the event is triggered. From a common experience, there are some general characteristics of the embedded system [18]:

1. Platform dependent

Most of the embedded software development do not have the same runtime environment. Besides, due to its hardware dependent characteristics, embedded software may have different test result when the test is run on host or target environment. Therefore, the embedded system testing must be run on both host and target environment as well to enhance the test coverage. In this case, it will increase the test development cost and consume time on test distribution to both host and target environment.

2. Accuracy of response

In order to determine the accuracy of embedded system on meeting the pre-requirement, performance of the system can be used as an indicator. Besides, response

time taken for the request to complete a single transaction can be another suitable indicator to obtain the accuracy of embedded system.

3. Limited resources (memory usage)

Memory is an important element to take into consideration during the embedded system design due to limited memory allocated for different tasks.

2.2.1. Hardware Interface

The hardware interface is a runtime system that handles the device drivers and provides interfaces to help the higher level of software systems such as applications and Real-Time Operating System (RTOS) to obtain services. One of the important hardware interface is the Hardware Adaptation Layer (HAL). In general, a typical HAL implementation includes some collection of standard libraries and vendor specific interfaces [4]. With the implementation of HAL, applications are more independent and can be developed easily to execute directly on the hardware without the support of operating system.

2.2.2. Software Interface

A software interface consists of code developed for different usages of specific application. Applications are developed using different programming language and the services are utilized from different layers like RTOS and HAL.

In the development of RTOS, there are many components involved such as task management, inter-task communication, context-switching, interrupt handling, and memory management facilities. With the present of these components, different embedded system applications can be designed to meet functional requirements and

deadlines using the provided libraries and Application Program Interface (API). For example, the Kernel is developed to support applications in devices with limited memory and small footprint. In general, Kernel is a program that constitutes the central core of a controller's operating system. In order to support real-time applications, the execution time of most of the Kernel functions must be deterministic to schedule the tasks based on priority assignment.

Generally software components can be developed using different programming languages and implemented on different operational platforms as well. Development of these software components may involve either internal developer, outsource to external third party company or the cooperation of both parties. The advantage of implementing this design structure is the quality of the product can be well maintained through the analysis of software components from different parties.

On the other hand, testing is also another important key component in the design structure to identify issues as earlier as possible. Without the full system testing, some of the significant issues or defects are unable to address due to independence characteristic of components. As a result, during the integration of related components, complex behaviors are observed and many possible defects may become more and more visible. Hence, full integration system testing is still required regardless of how many testing has been done on each of the component independently. Large amount of possible integration test have to be developed and executed to validate the functionality of the system [15].

The possibility of having problems in the overall design of embedded system is directly proportional with the size and complexity of software system. The key component which lead to success of software projects may highly depend on the way how the groups of components are designed, integrated, and structured. Any possible

problems that exists in the integration may lead to higher cost in the overall system development.

Poor controllability and maintenance of the embedded system can increase the difficulty on testing the overall system of embedded software. As most of the embedded software are communicating with the hardware by signal integration rather than user interface, the overall system is more difficult to control and monitor. Apart from this, the specifications of component-based software can also increase the complexity, which at the same time affects the observability and controllability.

2.3 Software Testing

Software testing is a process of developing and executing a test plan or written program with given inputs to ensure the program is running as designed and no error occurs. It is one of the important stage in system development as it represents the final stage of validation of design and specification. An ideal software testing is able to identify any possible errors with minimum execution time but maximum test coverage.

Software testing plays an important role in reducing any possible operating cost due to a defect injected in the requirements that is found in the final stage of a product life cycle. Insufficient of the validation done on the software system may lead to higher fixing cost. The condition could be worse if the defect is found in the field and requires to recall product from customer. Hence, it clearly shows that a successful and effective testing process manages to reduce the defect lifecycle as short as possible. This is the reason why most of the testing process is involved in the early stage of development process.

Software testing shall be planned carefully to develop an effective test strategy to avoid any major financial loss due to poor testing planning. During the software testing planning, test strategy shall be focused on balancing both testing costs and test coverage based on the criteria of maximum defects allowed.

In general, software testing can be divided into two different categories which are black-box testing and white-box testing [25] [26]. Both testing methodologies are having their pros and cons respectively depending on the testing requirement and scope of testing. Besides, there are some important principles in software testing that can affect the outcome and efficiency of the overall testing process such as:

1. Test case must be developed based on the requirement of the system.
2. A test case must contain a clear definition of the expected output or results.
3. Both invalid and unexpected input conditions that are valid and expected must be written in a test case.
4. A test case shall be able to examine a program to check if it is behaved as expected.
5. Test cases that are created should be repeatable to use in regression testing.
6. Every test result shall be inspected and analyzed thoroughly.
7. A test case must always be designed with assumption to find a defect.
8. The probability of identifying defects is proportional to the number of defects that have been found.

2.3.1. Type of Testing

In general, testing process are classified into 4 different levels to improve the quality of software testing and provide a more proper testing methodology which applicable for several projects. The classification of these levels help to organize the tasks more effectively and allow multiple tests to be performed simultaneously. The flow chart of different software testing levels is shown in Figure 2.2.

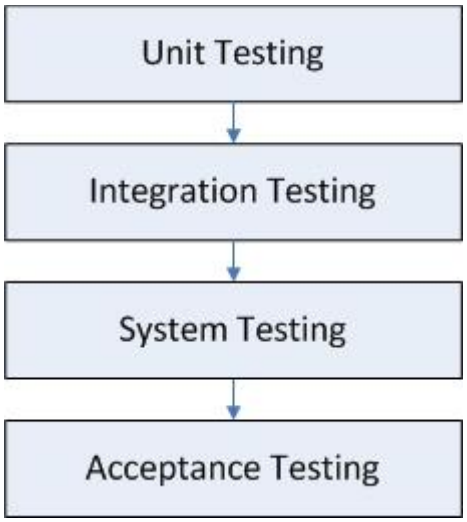

Figure 2.2: Level of Software Testing [18]

The levels of testing have a hierarchical structure which builds up in sequence where higher level is performed with assumption that lower level tests have been executed successfully without error. In other words, lower level testing like unit testing is always executed first before integration testing follows by system testing and acceptance testing. Unit and integration testing are normally executed in Integrated Development Environment (IDE) of the programmer while system testing is executed on a simulated environment or a system test machine.

Unit testing is a code based testing which is normally performed by developers during the code development as it requires knowledge and understanding of the overall

systems functional specification. It is the most basic testable function of software and mainly executed to test every individual units separately. Moreover, it also helps to determine the reliability of the overall system when testing is performed on the next hierarchical levels. For example, it typically refers to basic classes and interfaces in an object-oriented programming language.

In unit testing, most common testing method that is used is white box testing due to the simplicity of the test code. The purpose of unit testing is to discover any possible defects as early as possible because it is easier and faster to fix defects at this stage compared to later development stages. Besides, unit testing also minimizes any issue caused by code changes as the test codes are independent and can be validated instantly every time a change is made to ensure the functionality of the code.

Unit testing is normally performed within the IDE with the aid of some test case generator tool. Some good examples of a framework that allows automated unit testing is JUNIT [28]. It is not only a unit testing framework for both xUnit and java, but is also a more general framework which supports other languages like Python, C++, C# and ASP.Net. Apart from this, Andrews J. H et al. has come out with a framework namely RUTE-J to randomize unit testing for java to generated unit test cases [27]. It works by constructing appropriate test case arguments according to the optimal ranges that are set for test case parameters. In short, unit testing is considered as the most important level of testing as it is the root for the overall software structure.

Integration testing is the second level of software testing that combines several units of test codes to form a working subsystem. As unexpected error might occur even every unit has been gone through successful unit test, validation test needed to be performed to ensure the integrations work together properly. In this level, the testing is focus on the interaction between different test code units. This testing is normally

performed by software developer during the development of functional codes. Different testing approach like black box testing, white box testing or grey box testing can be used based on the type and function of the unit test code.

Nowadays, many complex systems are developed by combining several libraries of code [22]. These libraries may be developed by different developers or download from an open source project. As different developers tend to have different programming style, integration test developer need to develop a wrapper code to combine the use of different unit codes to obtain the desired functionality of the given library. Although all of the unit codes are tested thoroughly, it is still very important to test the interoperability between these units or libraries to produce stable functions.

Unified Modeling Language (UML) diagrams are very useful as test cases can be derived or developed automatically from both state diagrams and use-case diagrams to automate the integration testing. In this case, Siemens Corporate has performed a research on automating test case generation and execution using UML diagram [29]. It achieved a design based testing environment by generating and executing test case based on the detailed definition of the system's dynamic behavior and a good description of the test requirements with the help of UML state chart. Other than this, Substra has proposed another framework that generates integration tests referring to the sequence of method calls that obtained from an initial run in subsystem under test [30].

System testing is the next higher level of the software testing process after the completion of integrated system testing. On embedded systems, this testing is usually executed by testing team on the prototypes after all functionalities of the embedded software have been tested. It is usually performed on a system test machine or a configured environment to simulate the end user environment as realistically as

possible. The purpose of system testing is to identify defects in software that does not meet the requirements.

Test planning for system testing is usually performed in the early stage of project development lifecycle. The test plan depends on the high-level of design specification in the development process as it needs to translate the requirements specification and design specification. In summary, the objective of this testing is to ensure the overall system work properly in a production-like location so that the required business functions can be provided as specified in the high-level design.

In system testing, there are several forms of testing which are involved generally. One of the form is the stress testing where it is carried by loading multiple program under heavy volume to test the robustness of the system over a short period of time. The flow chart of the stress testing is shown in Figure 2.3.

One of the good example for stress testing is loading a large amount of database into an online application. Next, the maximum number of concurrent users that the application can handle is determined and investigated to observe how the systems handle the case when the number of concurrent users exceeds maximum amount.

Besides, another form of system testing namely performance testing can be performed to identify how efficient a system can work under certain workloads. This type of software reliability testing can help to validate the stability of the systems to maintain in good condition over a given uptime.

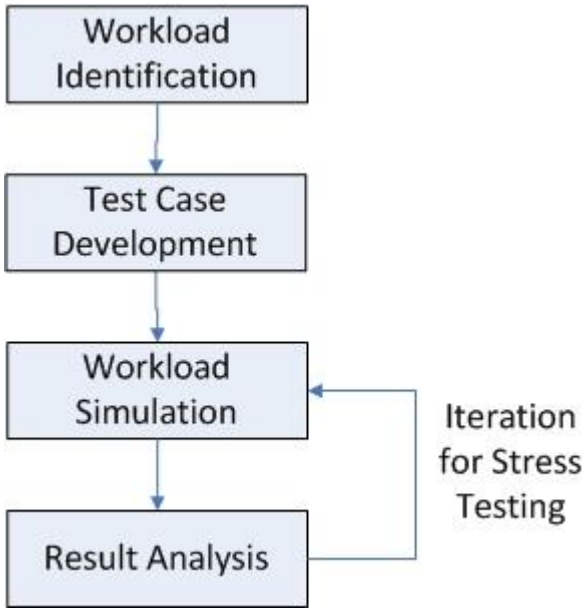

Figure 2.3: Stress Testing Flow Chart

In the final stage, acceptance testing is performed on the product to evaluate the design's compliance before release to customer. The purpose of this testing is to inspect whether the designed product fulfill its original requirements for the customer. Acceptance testing is usually the first to plan in a software testing cycle as it is completely based on user requirements specifications.

Acceptance testing is designed based on user requirement and normally conducted by testing team or sometimes customer. It can be carried out by either pilot testing or benchmark testing. Benchmark testing is run by simulating the required configuration settings to measure the system's performance in the end-user or customer environment.

On the other hand, pilot testing is carried out on a user site or a user simulated environment through Alpha testing and Beta testing. Alpha testing is developed and executed internally. Next, Beta testing is performed where the system is only released to a limited number of people for further testing in order to find out any possible defect

or issue. However, it is expected to have minimum number of defects discovered or preferably none as many testing have been carried on before reaching this phase.

In a corner case, compatibility testing may get involved in acceptance testing as well. It is used to determine any compatibility issue when the newly developed system is used together in old environment. All existing data are required to be validated to ensure no data loss when the new system is used to replace an older version of system.

Aside from four basic levels of system testing, there is another type of testing namely regression testing. It is a continuous testing of software that spans through all level of software testing whenever there is code changes made or defect fixes to ensure the reliability of each software release. Besides, this testing also ensures the stability of the software by ensuring no any new error is introduced into the system after the changes. Figure 2.4 shows how regression testing involved in the overall software testing process.

Regression testing can be run with two general types of modification which are adaptive maintenance and corrective maintenance. Adaptive maintenance involves modification of system specification where generation of new test cases are needed to suit the new specification while corrective maintenance is not modifying the system specification and only supports the reuse of test cases.

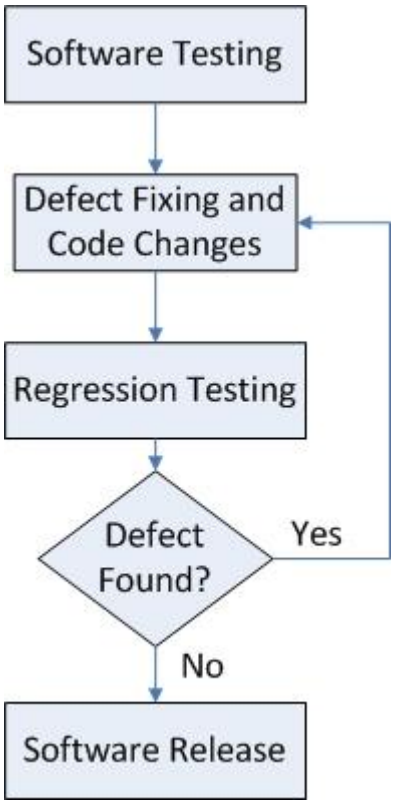

Figure 2.4: Regression Testing Flow Chart

In regression testing, there is always an important issue on whether to re-run all tests or only part of the test cases that related to the changes. Re-run all tests can provide a complete coverage for the testing but it is more time consuming and not necessary especially when the changes is minor. A more practical way is to perform selective strategy to identify which part of the existing test cases should be run to reduce the time allocated for regression testing. However, another question is raised on how to come out with the selective strategy.

Despite the complexity on determining the dependencies between the test cases and source code changes, Qing Gu and Chen have compared selective regression test technique to optimize the regression test coverage [31]. Zhihao Zhang also came out with regression test generation approach based on Tree-Structured analysis [32]. Besides, Nan Ye and Xin Chen proposed regression test cases generation based on

Automatic Model Revision with the purpose to make comparison between affected paths and new paths in activity diagrams to identify the software changes [33].

In summary, it is important to perform the software testing at different levels so that defects or errors that occur in the software can be identified easily and solution can be prepared in short period. Each level of testing is playing important role to provide a working subsystem that can be compiled and used without error besides maintaining the quality of the product. All of the software testing level can be summarized in Table 2.1.

Table 2.1: Level of Software Testing [34]

Testing Level	**Testing Type**	**Specification**	**Party involved**	**General Scope**
Unit	White Box Testing	Basic and lowest level	Developer	Smallest unit of code
Integration	White & Black Box Testing	Low and High Level Design	Developer	Multiple classes
Functional	Black Box Testing	High Level Design	Tester	Entire product
System	Black Box Testing	Requirements Analysis	Tester	Entire product in Production environments
Acceptance	White & Black Box Testing	Requirements Analysis	Tester/ Customer	Entire product in customer's environment
Regression	White & Black Box Testing	Changed Documentation	Developer/ Tester	Any of above phase

2.3.2. Black Box Testing

Black Box Testing is also known as functional testing. The goal of black-box testing is to identify any circumstances in the program that does not behave according to its expected behaviors. In this approach, test data are derived without taking advantage of knowledge of the internal structure of the program as shown in Figure 2.5.

In theory, this testing approach requires an exhaustive input testing in order to identify all possible errors as there are a large amount of possible inputs available in the system. The number of test cases may increase up to infinity and make it almost impractical to be implemented. Hence, in black box testing, it is important to develop the test cases in a suitable design so that the number of test cases are on acceptable level while having a large test coverage. The importance of black box testing is it can handle both valid and invalid inputs from customer's perspective [23].

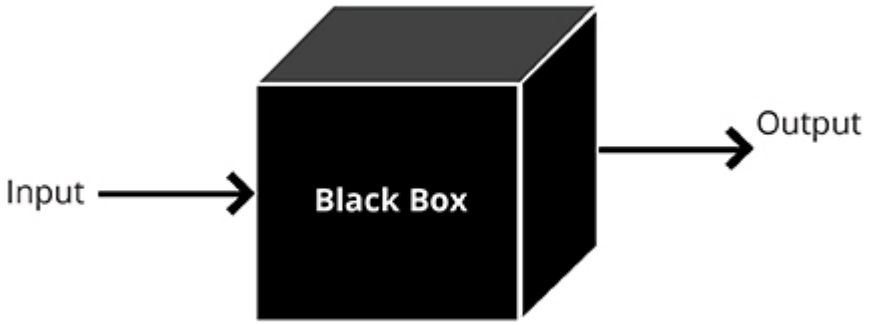

Figure 2.5: Black Box Testing

Typical black-box testing techniques include boundary value analysis, equivalence partitioning, decision table testing, combinatorial testing [38] and state transition tables. For example, M. Sharma and B. S. Chandra have implemented decision table technique to automate the test suites generation [24].

Equivalence partitioning method is one of the black-box testing technique that providing the tester with a systematic method for decomposing a functional specification into test specifications for every individual functions [8]. Based on the available features of the device under test, the input values are divided into two or more partitions. The testing values that are set in each of the partitions can be either valid or invalid input.

A test specification language can be used to determine test cases by specifying the interaction of parameters and environment factors relevant to the target program. All test cases that are determined in the same single partition are assumed to have the similar functionality or exhibit the same behavior [35]. From each partition, only one test case is required to run after analyzing the input data range. This can help to reduce the number of test cases required to achieve maximum functional coverage. With this, it can eliminate the need for exhaustive testing, which is not practical.

For example, taking measurement of current which accepting input I in range of 0 to 10 A, the values of I can be further categorized into three different partitions which are valid input between 0 and 10 and invalid input which less than 0 and more than 10. By using this method, the expected output for the test case will be pass when input value is valid and fail when input value is invalid.

Besides the advantages that can be obtained from equivalence class, there are some limitations in this method as well. It is always difficult to validate the assumption that the data value that belongs to the same equivalence class is processed in the same way in the system. Furthermore, equivalence partitioning is not an independent method in the testing process as it has to be supported by boundary value analysis method.

In boundary-value analysis, boundary conditions of the test cases are evaluated to create suitable test cases for the system [35]. The upper and lower limits of both input and output classes are identified and exercised thoroughly. For example, for a measurement unit like current I, the boundary condition test cases can be 0.0, 1.0, -0.001 and 1.001. Error can be detected through the comparison of the value set using the boundary values.

However, it might be time consuming to analyze the output boundaries to develop a test case that can validate the condition where the output value go beyond its specified boundaries. Besides, it cannot be used to determine boundary analysis for certain cases such as for Boolean or logical variables.

Another black box testing method is Decision Tables method. They are made up with series of rules that are used to express the design expert's knowledge in a compact form. They are normally implemented in the case where a set of rules and regulations that need to be followed. Basically there are four areas that are involved in decision table which are condition stub, condition entry, the action stub and finally action entry [36].

Other than this, a defect guessing technique can also be implemented to predict the outcome of test case to cover the area that are not tested in other testing techniques. This technique is much more difficult to be implemented as it requires experience and knowledge on the device under test. Test cases that are developed using this method can be either valid or invalid as it may require experiment to verify the feasibility of the test cases.

Black-box testing is a direct and easier way to create test cases as it does not require the inspection of the internal logic in the overall program. It can be an effective

way to identify faults for system with thorough specification. However, there are some limitations when testing using black-box testing since it does not consider the internal structure of the program. Hence, to increase the coverage of the testing, another testing method which is white box testing can be used.

2.3.3. White Box Testing

White Box testing is another type of testing method that are used widely in software testing area. In this method as reflected in Figure 2.6, the internal structure of the program is examined and used as a reference to develop and create test cases. It provides a clearer overview of the system and allow testers to determine how thoroughly the program needs to be tested based on test criteria that has been pre-defined.

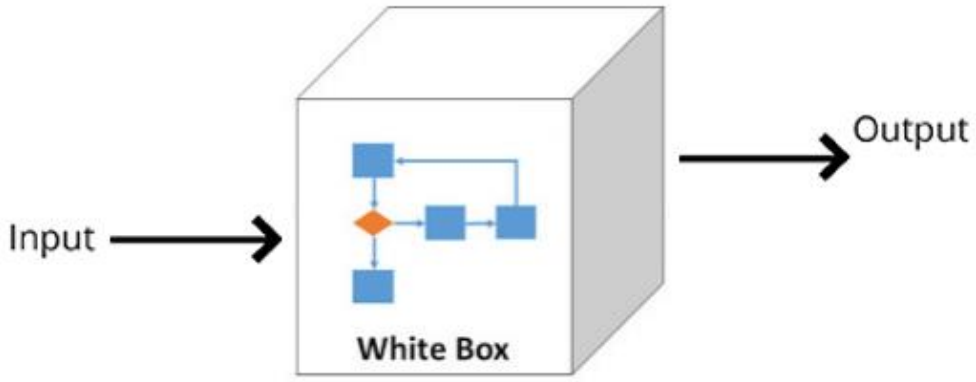

Figure 2.6: White Box Testing

In order to check whether the testing coverage is sufficient, a test criteria is used as an indicator. Some popular white-box testing methods that are widely used including data flow testing, condition coverage testing, statement coverage testing,