# 6. TESTING AND VALIDATION

## 6.1 INTRODUCTION

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and a finished product. It is the process of exercising software with the intent of ensuring that the software system meet sits requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

The main objective of testing is to find defects in requirements, design, documentation, and code as early as possible. The test process should be such that the software product that will be delivered to the customer is defect less. All Tests should be traceable to customer requirements.

Test cases must be written for invalid and unexpected, as well as for valid and expected input conditions. A necessary part of a test case is a definition of the expected output or result. A good test case is one that has high probability of detecting an as-yet undiscovered error.

### Basic Principles of Testing

- Define the expected output or result.
- Don’t test your own programs.
- Inspect the results of each test completely.
- Include test cases for invalid or unexpected conditions.
- Test the program to see if it does what it is not supposed to do as well as what it is supposed to do.
- Avoid disposable test cases unless the program itself is disposable.
- Do not plan tests assuming that no errors will be found.

The probability of locating more errors in any one module is directly proportional to the number of errors already found in that module.

## 6.2 TYPES OF TESTS

In Testing, there are mainly four types of tests. They are:

- **Unit Testing**
- **Integration Testing**
- **Functional Testing**
- **System Testing**

## ▪ Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application and or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

## ▪ Integration Testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

## ▪ Function Testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation and user manuals.

Functional testing is centered on the following items:

Valid Input : Identified classes of valid input must be accepted.

Invalid Input : Identified classes of invalid input must be rejected.

Functions : Identified functions must be exercised.

Output : Identified classes of application outputs must be exercised.

Systems/Procedures: Interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify business process flows; data fields, predefined processes and successive processes must be considered for testing. Before functional testing is completed, additional tests are identified and the effective value of current tests is determined.

- **System Testing**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

## Defects and Failures

Not all software defects are caused by coding errors. One common source of expensive defects is caused by requirement gaps unrecognized requirements that result in errors of omission by the program designer. A common source of requirements gaps is non-functional requirements such as testability, scalability, maintainability, usability, performance and security.

Software faults occur through the following processes. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure. Not all defects will necessarily result in failures. For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new computer hardware platform, alterations in source data or interacting with different software. A single defect may result in a wide range of failure symptoms.

**Static and Dynamic Testing** : There are many approaches to software testing. Reviews, walk throughs or inspections are referred to as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. Static testing

can be omitted, and unfortunately in practice often is. Dynamic testing takes place when the program itself is used. Dynamic testing may begin before the program is 100% complete in order to test particular sections of code and are applied to discrete functions or modules. Typical techniques for this are either using stubs-drivers or execution from a debugger environment. Static testing involves verification whereas dynamic testing involves validation. Together they help improve software quality.

### 6.2.1 Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

### Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

### Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

### Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed

All links should take the user to the correct page.

### 6.2.2 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications. E.g. components in a software system or – one step up – software applications at the company level interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

### 6.2.3 Functional Testing

### Functional testing is centered on the following items

Valid Input : Identified classes of valid input must be accepted.

Invalid Input : Identified classes of invalid input must be rejected.

Functions : Identified functions must be exercised.

Output : Identified classes of application outputs must be exercised.

Systems/ Procedures : Interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

### 6.2.4 System Testing

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

### White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software or at least its purpose. It is used to test areas that cannot be reached from a black box level.

### Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document such as specification or requirements document such as specification or requirements document. It is a testing in which the software

under test is treated, as a black box. you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

### Error Guessing

Error guessing is an informal testing technique where testers rely on their intuition and experience to anticipate potential errors in a system. In the context of a blood donation system with blockchain technology, error guessing involves predicting where errors might occur based on factors such as input validation, transaction handling, blockchain integration, user interaction, concurrency and security. While not a formal method, error guessing can help identify issues that may be overlooked by more structured testing approaches, contributing to the overall assurance of data integrity in the system.

## 6.2.5 Unit Testing

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a modules control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is unit testing.

### Test strategy and approach

Field testing will be performed manually and functional tests will be written in details.

### Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

### Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page

### 6.2.6 Integration Testing

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

The following are the types of Integration Testing:

### 1. Top-down Integration

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner.

In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards.

### 2. Bottom-up Integration

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated. The bottom up integration strategy may be implemented with the following steps:

- The low-level modules are combined into clusters into clusters that perform a specific Software sub-function.
- A driver the control program for testing is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure.
- The bottom-up approaches tests each module individually and then each module is module is integrated with a main module and tested for functionality.

## System Testing

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and or a finished product.

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

## 6.2.7 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

### Test Cases Screens

Test Cases of the project and this testcases shows the project performance and working of the project.

- Test case 01 - Validate Admin Login
- Test case 02 - Validate Blood Donor Registration Process
- Test case 03 - Validate Blood Donor Login
- Test case 04 - Validate Blood Banker Registration Process
- Test case 05 - Validate Blood Banker Login
- Test case 06 - Validate Hospital Registration Process
- Test case 07 - Validate Hospital Login

**Test case 01: Admin Login**

| | |
|---|---|
| **Test Case ID** | TC - 1 |
| **Test Case Name** | Admin Login |
| **Test Case Description** | Check whether the input data is taken in the form of Username and Password. |
| **Requirements** | Eclipse IDE, jdk 1.7, Servlet, MySQL |
| **Excepted Results** | Login to Admin page |
| **Actual Results** | Login to Admin page Successfully. |
| **Test Results** | Pass |
| **Remarks** | No Remarks |

**Test case 02: Blood Donor Registration**

| | |
|---|---|
| **Test Case ID** | TC - 2 |
| **Test Case Name** | Blood Donor Registration |
| **Test Case Description** | Check whether the input data is taken as text such as Name, Mail id, password, Address, Location, Mobile Number, photo |
| **Requirements** | Eclipse IDE, jdk 1.7, Servlet, MySQL |
| **Excepted Results** | Blood Donor Registration Process Successfully Completed |
| **Actual Results** | Blood Donor Registration Process has executed Successfully |
| **Test Results** | Pass |
| **Remarks** | No Remarks |

**Test case 03: Blood Donor Login**

| | |
|---|---|
| **Test Case ID** | TC - 3 |
| **Test Case Name** | Blood Donor Login |
| **Test Case Description** | Check whether the input data is taken in the form of Username and Password. |
| **Requirements** | Eclipse IDE, jdk 1.7, Servlet, MySQL |
| **Excepted Results** | Login to Blood Donor Successfully |
| **Actual Results** | Login to Blood Donor Module Successfully. |
| **Test Results** | Pass |
| **Remarks** | No Remarks |

**Test case 04: Blood Banker Registration**

| | |
|---|---|
| **Test Case ID** | TC - 4 |
| **Test Case Name** | Blood Banker Registration |
| **Test Case Description** | Check whether the input data is taken as text such as Name, Mail id, password, Address, Location, Mobile Number, photo |
| **Requirements** | Eclipse IDE, jdk 1.7, Servlet, MySQL |
| **Excepted Results** | Blood Banker Registration Process Successfully Completed |
| **Actual Results** | Blood Banker Registration Process has executed Successfully |
| **Test Results** | Pass |
| **Remarks** | No Remarks |

**Test case 05: Blood Banker Login**

| | |
|---|---|
| **Test Case ID** | TC - 5 |
| **Test Case Name** | Blood Banker Login |
| **Test Case Description** | Check whether the input data is taken in the form of Username and Password. |
| **Requirements** | Eclipse IDE, jdk 1.7, Servlet, MySQL |
| **Excepted Results** | Login to Blood Banker Successfully |
| **Actual Results** | Login to Blood Donor Banker Successfully. |
| **Test Results** | Pass |
| **Remarks** | No Remarks |

**Test case 06: Hospital Registration**

| | |
|---|---|
| **Test Case ID** | TC - 6 |
| **Test Case Name** | Hospital Registration |
| **Test Case Description** | Check whether the input data is taken as text such as Name, Mail id, password, Address, Location, Mobile Number, photo |
| **Requirements** | Eclipse IDE, jdk 1.7, Servlet, MySQL |
| **Excepted Results** | Hospital Registration Process Successfully Completed |
| **Actual Results** | Hospital Registration Process has executed Successfully |
| **Test Results** | Pass |
| **Remarks** | No Remarks |

**Test case 07: Hospital Login**

| | |
|---|---|
| **Test Case ID** | TC - 7 |
| **Test Case Name** | Hospital Login |
| **Test Case Description** | Check whether the input data is taken in the form of Username and Password. |
| **Requirements** | Eclipse IDE, jdk 1.7, Servlet, MySQL |
| **Excepted Results** | Login to Hospital module |
| **Actual Results** | Login to Hospital module Successfully. |
| **Test Results** | Pass |
| **Remarks** | No Remarks |

## 6.3 Validation

Testing validation in a blood donation system with blockchain technology involves assessing the system's ability to maintain data integrity. It includes validating input validation mechanisms, transaction handling processes, blockchain integration, user interactions, concurrency control, and security measures. Through various testing methodologies such as gray box testing, boundary value analysis, error guessing and security testing. The system's functionality, reliability, and security are evaluated. The validation process aims to identify and resolve any issues or vulnerabilities that could compromise data integrity, ensuring the system's readiness to securely manage blood donation data on the blockchain.

## 6.4 Summary

In conclusion, testing of the blood donation system with blockchain technology has validated its ability to ensure data integrity effectively. Through comprehensive testing methodologies such as security testing, the system's functionalities, reliability and security measures have been thoroughly assessed. Identified issues and vulnerabilities have been addressed and the system demonstrates readiness to securely manage blood donation data on the blockchain, enhancing trust and reliability in the blood donation process.