**CHAPTER 8**

**TESTING**

A process of executing a program with the explicit intention of finding errors, that is making the program fail. It is the process of detecting errors and performs a very critical role for quality assurance, also for ensuring the reliability of software. The results of testing are used later on during maintenance also.

Testing Approaches:

There are three types of software testing approaches.

* **White Box Testing**: It is also called as Glass Box, Clear Box, Structural Testing. White Box Testing is based on applications internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing is usually done at the unit level.
* **Black Box Testing**: It is also called as Behavioral/Specification-Based/Input-Output Testing. Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure.
* **Grey Box Testing**: Grey box is the combination of both White Box and Black Box Testing. The tester who works on this type of testing needs to have access to design documents. This helps to create better test cases in this process.

**8.1. UNIT TESTING**

Unit Testing is done to check whether the individual modules of the source code are working properly. i.e. testing each and every unit of the application separately by the developer in the developer’s environment. It is AKA Module Testing or Component Testing. It concentrates on each unit of the software as implemented in source code and is a white box oriented. Using the component level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. In the unit testing, the steps can be conducted in parallel for multiple components in

this project we tested all the modules individually related to main function codes and attacks also.

Below are the unit test cases that are run manually:

|  |  |  |
| --- | --- | --- |
| Test Case | Expected Result | Result |
| After the start of system | Display correct occupancy status messages for all the parking slots | Pass |
| Upon receiving occupancy status change from client for parking slot X | Change in the parking slot X only, and should not affect other parking slot status messages. | Pass |
| Upon receiving Occupancy Status as ‘Occupied’ | Change Occupancy Message from ‘Vacant’ to ‘Occupied’ | Pass |
| Upon receiving Occupancy Status as ‘Vacant’ | Change Occupancy Message from ‘Occupied’ to ‘Vacant’ | Pass |
| Upon a vehicle occupying a vacant parking slot | Display the presence of vehicle at that particular time in the system | Pass |

Table 2: Test cases for Unit Testing

**8.2. INTEGRATION TESTING**

Here focus is on design and construction of the software architecture. Integration Testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design. The goal here is to see if modules can be integrated properly, the emphasis being on testing interfaces between modules.

This testing activity can be considered as testing the design and hence the emphasis on testing module interactions. In this project the main system is formed by integrating all the modules. When integrating all the modules we have checked whether the integration effects working of any of the services by giving different combinations of inputs with which the two services run perfectly before integration.

Below are the few Integration Test Cases that are done manually:

|  |  |  |
| --- | --- | --- |
| Test Case | Expected Result | Result |
| After the start of the system | Display correct occupancy status messages for all the parking slots and also display occupancy trends of all parking slots in the chart at all the recorded timings | Pass |
| Upon receiving Occupancy Status as ‘Occupied’ | Change in Occupancy Message from ‘Vacant’ to ‘Occupied’ | Pass |
| Upon receiving Occupancy Status as ‘Vacant’ | Change in Occupancy Message from ‘Occupied’ to ‘Vacant’ | Pass |
| Upon a vehicle occupying a vacant parking slot | Display the presence of vehicle at that particular time in the chart and also change in the occupancy status and message | Pass |

Table 3: Test cases for Integration Testing

**8.3. SYSTEM TESTING**

It’s a black box testing. Testing the fully integrated application this is also called as end to end scenario testing. To ensure that the software works in all intended target systems. Verify thorough testing of every input in the application to check for desired outputs. Testing of the users experiences with the application.