**Chapter 8**

**TEST SPECIFLCATIONS**

**8.1 MANUAL TESTING**

Manual testing is a crucial part of software development that involves the use of human testers to check the behavior and functionality of a software application or system. Unlike automated testing, manual testing requires testers to follow predefined steps to verify if the application or system meets its specified requirements.

Manual testing involves a range of activities, including test planning, test design, test execution, and test reporting. Test planning involves identifying the scope of the test, selecting the test cases and scenarios, and choosing the testing environment. Test design involves creating test cases and scenarios based on the requirements and specifications of the software application. Test execution involves running the test cases and scenarios manually to check the software's behavior and functionality. Test reporting involves recording and documenting the test results and defects found during the testing process.

Manual testing is a time-consuming and labor-intensive process, but it can be highly effective in finding defects that might be missed by automated testing. However, to get reliable results, it is crucial to ensure that the manual testers are skilled and experienced in testing. Manual testing is an important part of the software development lifecycle, as it helps to identify issues early on in the development process and ensure that the software meets the end-user's requirements. However, manual testing can be time-consuming and labour - intensive, and it is often complemented by automated testing to improve efficiency and test coverage.

**8.1.1 Unit testing**

Unit testing is a technique where software developers write automated tests for individual parts of their code to check if they are working correctly. These parts are typically functions, methods, or classes that are tested independently from the rest of the application. By running these tests early and often, developers can catch and fix issues before they become more serious problems later on.

The main goal of unit testing is to ensure that each piece of code works as expected and produces the expected output for a given input. This helps developers to catch and fix bugs quickly, leading to higher-quality software that is less likely to break or fail when deployed.

Overall, unit testing is a simple but powerful technique that can help software developers to build better software by catching and fixing issues early in the development process. By breaking down their code into smaller, testable pieces, developers can build software that is more reliable, easier to maintain, and more resilient to change.

**8.1.2 Integration testing**

Integration testing is a type of software testing that evaluates how individual software components work together as a group. The objective of integration testing is to identify and resolve issues in the interface between these components, which can include data transfer, communication protocols, and other interactions. By testing how different components interact with each other, integration testing aims to ensure that the software system or application is reliable, functional, and meets the desired requirements.

Integration testing is performed after unit testing, where each individual unit of the software is tested in isolation. Once the individual units are tested and verified, they are integrated into the software system, and integration testing is performed to ensure that the integrated components work together as intended.

There are different types of integration testing, including big bang integration, top-down integration, bottom-up integration, and hybrid integration. Big bang integration involves integrating all the components of the software system at once, while top-down and bottom-up integration involve integrating the components in a specific order.

The benefits of integration testing include the early detection of defects in the interface between different components, increased software reliability and quality, and a reduction in development time and costs.

8.1.3 Acceptance testing

Acceptance testing is like giving the software a final test to make sure it meets the needs of the people who will be using it. It's like when you're baking a cake and you take a small piece to taste it before serving it to your guests to make sure it tastes good and meets their expectations. In the same way, acceptance testing ensures that the software is ready to be used by its intended audience and meets their requirements.

# **8.2 TEST CASE AND TYPICAL WRITTEN TEST CASE FORMAT**

Test case: Student login

Objective: To verify that students can successfully log in to the generic job readiness application.

Test steps:

* Open generic job readiness application login page.
* Enter a valid username and password.
* Click on the "Login" button.
* Verify that the user is directed to the dashboard page.

Expected result: The user is successfully logged in and directed to the dashboard page.

Test case: Quiz functionality

Objective: To verify that the quiz functions generic job readiness application are working properly.

Test steps:

* Open the generic job readiness application dashboard page.
* Click on the "Start Quiz" button.
* Answer all questions in the quiz.
* Click on the "Submit" button.
* Verify that the user is directed to the quiz results page.
* Verify that the quiz results page displays the number of correct and incorrect answers, as well as the percentage score.

Expected result: The quiz functions of generic job readiness application are working properly and the quiz results page displays accurate information.

Test case: Faculty panel functionality

Objective: To verify that the faculty panel functions of the generic job readiness application are working properly.

Test steps:

* Open the generic job readiness application faculty panel login page.
* Enter a valid username and password.
* Click on the "Login" button.
* Verify that the user is directed to the faculty panel dashboard page.
* Schedule a quiz in a time slot.
* Verify that the quiz is scheduled in the specified time slot.
* View the quiz results of a completed quiz.
* Verify that the quiz results page displays accurate information.

**8.2.1 Typical Structure of a Test Case**

1. Information

* Test case name: Verify user login functionality
* Test objective: To ensure that users can log in to the system successfully
* Preconditions: The user has a valid account and internet connection
* Test data: Valid username and password

2. Activities

* Launch the Interview Quiz application.
* Enter the valid username and password in the login page.
* Click on the login button.
* Verify that the user is successfully logged in to the system.
* Verify that the user is directed to the dashboard page.

3.Results

* Expected result: The user should be able to log in to the system successfully and be directed to the dashboard page.
* Actual result: The user is able to log in to the system successfully and is directed to the dashboard page.
* Pass/Fail: Pass

**8.2.2 Functional Dependency Graph**

1. admin login ()

2. admin profile()

3. admin add ()

4. admin edit ()

5. admin delete ()

6. admin Full Name ()

7. admin Email ()

8. admin Username ()

9. admin phone number ()

10. admin Password ()

11. admin dashboard ()

12. admin status ()

13. admin search ()

14. total admin ()

15. active admin ()

16. inactive admin ()

17. all college ()

18. college id ()

19. college address ()

20. college name ()

21. college logo()

22.college edit ()

23. notification ()

24.log out()

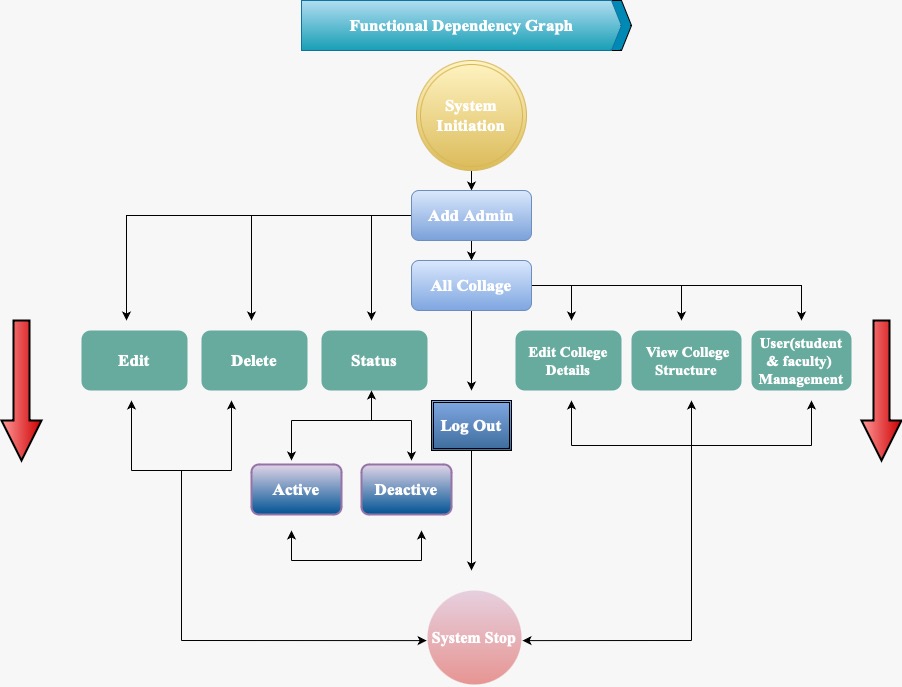


Figure 8.1 Function Dependency Graph