Experiment 8

Aim

Design test cases and perform white box testing. (evaluate code and the internal structure of software.)

Theory

Testing in software engineering is a critical phase of the software development lifecycle (SDLC) that involves evaluating a software application or system to identify and address defects, errors, and quality issues. The primary purpose of testing is to ensure that the software functions correctly, meets its intended requirements, and performs reliably in various scenarios and conditions.
Key objectives of software testing include

- Finding defects: The primary goal of testing is to identify defects or bugs in the software, including logic errors, coding mistakes, and usability issues.
- Ensuring functionality: Testing verifies that the software functions as intended and meets the specified requirements. This includes checking if all features work as expected.
- Performance assessment: Testing evaluates the software's performance under different conditions, such as load testing to assess its scalability and stress testing to determine its reliability under extreme conditions.
- Security testing: This type of testing is performed to identify vulnerabilities and security risks in the software, ensuring that it can withstand potential threats and attacks.

Good Testing : Good testing in software engineering is characterized by a set of principles, practices, and qualities that aim to ensure the quality, reliability, and effectiveness of the software being developed.

Here are some key attributes and principles that define good testing

- **Clear Objectives** : Good testing begins with well-defined objectives. Testers should have a clear understanding of what needs to be tested, the goals of testing, and the expected outcomes. This includes understanding the software's requirements, user expectations, and project goals.

- **Comprehensive Coverage** : Effective testing aims to cover as many aspects of the software as possible. This includes testing different functionalities, scenarios, inputs, and conditions. Test coverage should encompass unit testing, integration testing, system testing, and user acceptance testing.
- **Relevance** : Testing should focus on the most critical and high-risk areas of the software. Prioritize testing efforts based on the impact of potential failures and user priorities.

White-box testing : White box testing, also known as clear box testing, structural testing, or glass box testing, is a software testing technique that examines the internal structure and logic of a software application's code. Unlike black box testing, which focuses on testing the software from an external, functional perspective without knowledge of its internal workings, white box testing is concerned with understanding and evaluating the code's internal design, architecture, and implementation.

Here are the key aspects of white box testing in detail:

- **Objective**: The primary objective of white box testing is to ensure that the software's internal logic is correct and that all code paths are tested thoroughly. This type of testing aims to find defects related to code, such as logic errors, boundary conditions, and code optimization issues.
- **Testing at the Code Level** : White box testing is typically performed at the code or program level. Testers have access to the source code and use it to design test cases. This level of testing is often associated with unit testing and code review processes.
- **Coverage Criteria** : White box testing uses various coverage criteria to measure the extent to which the code has been tested. Common coverage criteria include statement coverage, branch coverage, path coverage, and condition coverage. These criteria help ensure that all code paths and conditions are exercised during testing.
- **Test Case Design** : Testers design test cases based on their knowledge of the code's internal structure. They may use techniques such as control flow graphs and data flow analysis to identify paths through the code that need testing. Test cases are created to follow specific code paths and exercise different code branches and conditions.
- **Code Inspection** : White box testing often involves code inspection and static analysis tools to identify potential issues in the code, such as syntax errors, code style violations, and code complexity problems.

- **Code Profiling** : Profiling tools can be used to analyze the runtime behavior of the code, identifying performance bottlenecks and memory leaks.

Basis Path Testing : Basis path testing is a white-box testing technique used in software engineering to systematically design test cases for a program's control flow. This method helps identify errors and defects in the program's logic, such as incorrect branching, loops, and decision-making processes.

Here's how basis path testing works:

- **Control Flow Graph (CFG)**: The first step in basis path testing is to construct a Control Flow Graph (CFG) for the program. A CFG is a graphical representation of the program's control flow, showing how different statements and branches are connected. Nodes in the graph represent program statements, and edges represent possible transitions between statements.
- **Cyclomatic Complexity**: Cyclomatic complexity is a metric used to quantify the complexity of the program's control flow. It is calculated based on the number of regions (or paths) in the CFG. The formula for cyclomatic complexity is:

$M = E - N + 2P$

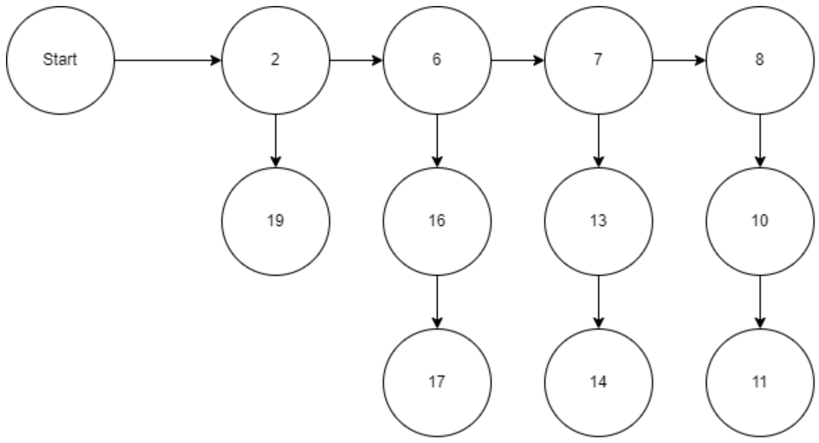
Where:

- M: Cyclomatic complexity
- E: Number of edges in the CFG
- N: Number of nodes in the CFG
- P: Number of connected components (usually 1 for a single program)

Cyclomatic Complexity

Code

```
1.  function loginUser(username, password) {
2.      if (!username || !password) {
3.          displayErrorMessage('Please provide a username and password.');
4.          return;
5.      }
6.      if (isValidUsername(username)) {
7.          if (isValidPassword(username, password)) {
8.              if (isUserAccountActive(username)) {
9.                  redirectToDashboard(username);
10.             } else {
11.                 displayErrorMessage('Your account is currently inactive. Please contact support.');
12.             }
13.         } else {
14.             displayErrorMessage('Invalid password. Please check your password.');
15.         }
16.     } else {
17.         displayErrorMessage('Invalid username. Please check your username.');
18.     }
19. }
```



The cyclomatic complexity can be calculated using McCabe's Cyclomatic Complexity formula:

M = E - N + 2P

Where:
- `M` is the cyclomatic complexity.
- `E` is the number of edges in the control flow graph.
- `N` is the number of nodes in the control flow graph.
- `P` is the number of connected components (usually 1 for a single program).

1. E (Edges) is the number of decision points (if statements) plus one for the "Start" node. :- There are 4 decision points (if statements) in your code.
2. N (Nodes) is the number of nodes in the control flow graph. :- There are 5 nodes (including the "Start" node).
3. P(Connected Components) is usually 1 for a single program.

Now, let's calculate the cyclomatic complexity using the formula:

M = E - N + 2P = 4 - 5 + 2 * 1 = 4 - 5 + 2 = 1

So, the cyclomatic complexity of your code is 1.

Test Case 1: User Login

Test Case ID : UI_AUTH_TC001

Test Objective : To verify that users can successfully log in to the music streaming platform.

Preconditions:

1. The music streaming platform is accessible.
2. The user is on the login page.

Test Steps :

1. Enter a valid username and password.
2. Click on the "Login" button.

Expected Results :

1. The login page should have fields for entering a username and password.
2. After clicking the "Login" button:
3. The system validates the provided username and password.
4. If the credentials are valid, the user is redirected to the platform's main dashboard.
5. If the credentials are invalid, an error message is displayed, indicating that the login failed.

Postconditions :

1. If login is successful, the user is on the platform's main dashboard.
2. If login fails, the user remains on the login page with an error message.

Test Data:

1. Valid username: [provide a valid username]
2. Valid password: [provide a valid password]
3. Invalid username: [provide an invalid username]
4. Invalid password: [provide an invalid password]

Test Case 2 : Search for a Song

Test Case ID : UI_SEARCH_TC001

Test Objective : To verify that users can search for a song and see relevant results.

Preconditions :

1. The user is logged into the platform.
2. The user is on the platform's main dashboard.

Test Steps :

1. Enter the name of a song in the search bar.
2. Click on the "Search" button.

Expected Results :

1. The user should be able to enter text in the search bar.
2. After clicking the "Search" button, a list of songs matching the search query should be displayed.

Postconditions :

1. The user can see a list of search results.

Test Data :

1. Song name to search for: [provide a song name]

Test Case 3: Play a Song

Test Case ID : UI_PLAY_TC001

Test Objective : To verify that users can play a song from the search results.

Preconditions :

1. The user is logged into the platform.
2. The user has searched for a song using the search functionality.

Test Steps :

1. Click on the "Play" button next to a song in the search results.

Expected Results :

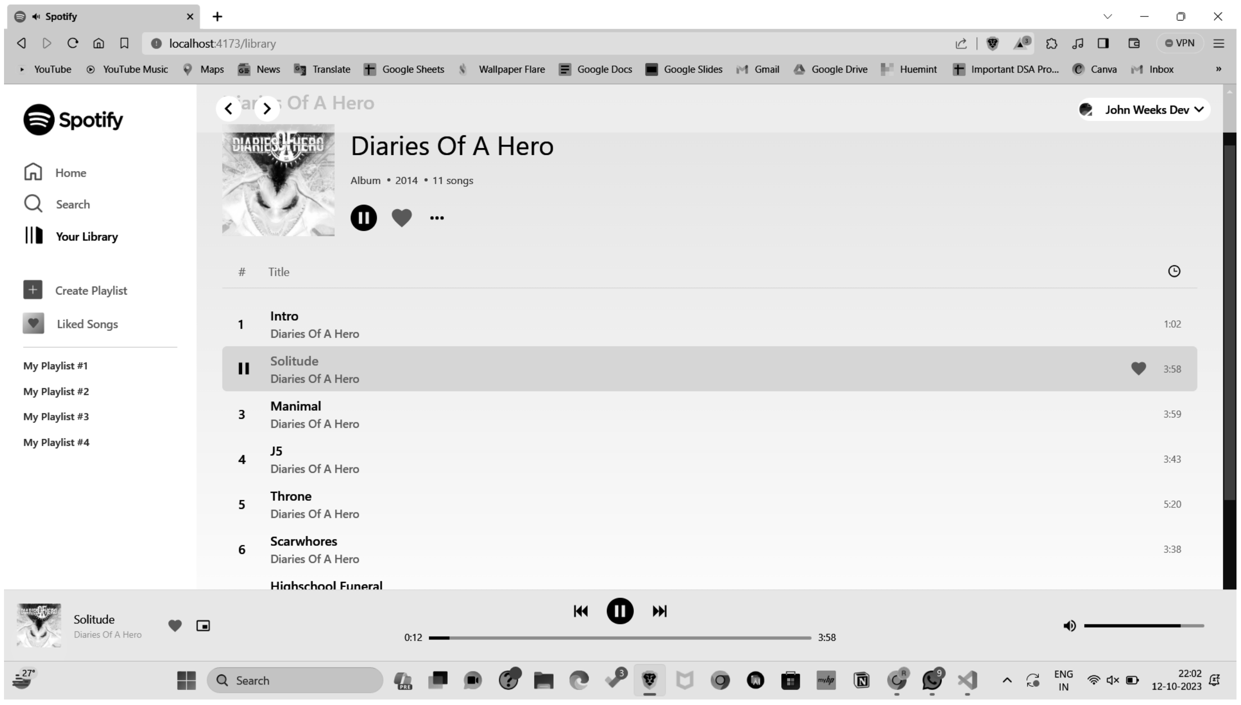
1. When the "Play" button is clicked, the selected song should start playing.

Postconditions :

1. The selected song is playing in the user's player.

Test Data :

1. Song to play: [provide a song from the search results]



Diaries Of A Hero

Album • 2014 • 11 songs

#	Title	
1	Intro Diaries Of A Hero	1:02
	Solitude Diaries Of A Hero	3:58
3	Manimal Diaries Of A Hero	3:59
4	J5 Diaries Of A Hero	3:43
5	Throne Diaries Of A Hero	5:20
6	Scarwhores Diaries Of A Hero	3:38
	Highschool Funeral	

Solitude
Diaries Of A Hero
0:12 3:58

Test Case 4: Create a Playlist

Test Case ID : UI_PLAYLIST_TC001

Test Objective : To verify that users can create a new playlist.

Preconditions :

1. The user is logged into the platform.
2. The user is on the platform's main dashboard.

Test Steps :

1. Click on the "Create Playlist" button.
2. Enter a name for the new playlist.
3. Click on the "Create" button.

Expected Results :

1. The user should be able to click the "Create Playlist" button.
2. After entering a name and clicking "Create," a new playlist with the provided name should be created.

Postconditions :

1. A new playlist with the specified name is created and appears on the user's dashboard.

Test Data :

1. Playlist name: [provide a name for the new playlist]

Test Case 5: Add Song to Playlist

Test Case ID : UI_PLAYLIST_TC002

Test Objective : To verify that users can add a song to an existing playlist.

Preconditions :

1. The user is logged into the platform.
2. The user has created at least one playlist.
3. The user is on the platform's main dashboard.

Test Steps :

1. Click on a playlist to open it.
2. Click on the "Add Song" button.
3. Search for and select a song to add to the playlist.
4. Click on the "Add" button.

Expected Results :

1. The user should be able to open a playlist, search for songs, and add a selected song to the playlist.

Postconditions :

1. The selected song is added to the chosen playlist.

Test Data :

1. Playlist to add the song to: [select a playlist]
2. Song to add: [provide a song to add to the playlist]

Test Case 6: Logout

Test Case ID : UI_AUTH_TC002

Test Objective : ** To verify that users can successfully log out of the music streaming platform.

Preconditions:

1. The user is logged into the platform.
2. The user is on the platform's main dashboard.

Test Steps :

1. Click on the user profile icon.
2. Click on the "Logout" option.

Expected Results :

1. After clicking the "Logout" option, the user should be logged out of the platform and redirected to the login page.

Postconditions :

1. The user is logged out and on the login page.

Test Data :

1. N/A (No specific data required for this test case)

Conclusion

The white box testing stands as a crucial pillar in the realm of software testing, offering an in-depth examination of a program's internal code structure. By scrutinizing the logic, branches, and decision points within the code, this approach aids in the early detection of errors and discrepancies, facilitating their resolution at a stage when they are less costly and time-consuming to address. Integrated seamlessly with the development process, white box testing fosters a culture of code quality, with developers often crafting unit tests as they code, promoting robust and maintainable software.