Lab 1B – Use Case Testing

# Lab Intro

This lab will introduce you to the concept of **Use Case Testing**. You will download the supplied zip file on Moodle and work your way through the programs, listing which parts of the use case descriptions satisfy their requirements with respect to being tested.

**Key Term – Use Case Testing**Use Case Testing verifies that a software system's functionality aligns with specific user interactions, ensuring it meets user requirements.



## Learning Outcomes

By the end of this session, you will be able to:

* Apply use case testing principles to identify and validate critical user interactions within software systems systematically.
* Demonstrate proficiency performing tests based on use cases, ensuring comprehensive functional testing.

## Resources

* **Lab1B-Apps.zip** – available on Moodle

## Lab Setup

* Download and extract **Lab1B-Apps.zip** to your W: drive (or O: drive if you do not have access to the W: drive).
* Open **IntelliJ** and open the extracted folder as we did with the previous lab

# Exercise 1 – ToDoListApp.java

Perform an exploratory test on the ToDoListApp.java (this is a console-based application, so there’s no GUI). While testing the different program functionalities, list the defects you find relating to the different Use Case Descriptors provided in Table 1. You can use Table 2 to link the defects you discover to specific Use Case IDs (add more rows as necessary and feel free to add/remove columns as you deem relevant). You should also highlight parts of the cells within the table to reflect if you believe it to be defect-free, or has defects, or if you’re not sure.

Table 1 – ToDoListApp Use Case Descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID** | **Use Case Name** | **Use Case Description** | **Preconditions** | **Postconditions** |
| UC01 | Add Item | The user can add a new item to the to-do list. | The user has launched the application. | A new item is added to the to-do list with the provided details. |
| UC02 | Remove Item | The user can remove an item from the to-do list. | The user has launched the application. | The selected item is removed from the to-do list. |
| UC03 | Display List | The user can view the current to-do list. | The user has launched the application. | The application displays the current to-do list to the user. |
| UC04 | Exit | The user can exit the To-Do List application. | The user is using the application. | The To-Do List application is closed. |

Table 2 – ToDoListApp Defect Log

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | **Defect Name** | **Defect Description** |
| UC04 | Exit | The program uses a **while (true)** loop to keep running until the user chooses to exit. While this works, it's generally better to provide a more graceful way to exit the program, such as when the user selects the "Exit" option. |
| UC02 | Remove Item | No removal confirmation |
| UC02 | Remove Item | Not be able to remove a specified item, we could implementation Item id, for remove specify items. |
| UC01 | Adding Item | The user should not be able to add an empty item to the to-do-list |
|  |  |  |

# Exercise 2 – TicTacToeGame.java

The next program you’ll be testing is TicTacToe (noughts and crosses). Games can be a mess to test – especially when the rules themselves are complex. Fortunately, the game of noughts and crosses is not as complex as something like Chess, but this will serve as a good introduction to testing game-related logic. As with the previous exercise, identify defects with the TicTacToeGame.java program, and document/highlight them as necessary.

Table 3 – TicTacToe Use Case Descriptions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID** | **Use Case Name** | **Use Case Description** | **Preconditions** | **Postconditions** |
| UC01 | Make a Move | Allows the player to make a move during their turn by clicking on an empty cell on the game board. | The game is ongoing, and it's the player's turn. | The selected cell is marked with the player's symbol (X or O), and it becomes the opponent's turn. |
| UC02 | Check for Win | Checks if a player has achieved a winning combination of their symbol (either X or O) on the game board. | A player has just made a move. | If a winning combination is found, the game announces the winning player and ends. |
| UC03 | Check for Tie | Checks if the game has ended in a tie, meaning that all cells on the game board are filled, and no player has won. | A player has just made a move. | If all cells are filled, and no winning combination is found, the game announces a tie and ends. |
| UC04 | Game Over | Signifies the end of the game, either due to a player winning or a tie. | A winning condition or tie condition has been met. | The game announces the winner (if applicable) or a tie, and the game concludes. |

Table 4 – TicTacToe Defect Log

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | **Defect Name** | **Defect Description** |
|  |  |  |
|  |  |  |

# Exercise 3 – TipCalculator.java

Same as earlier exercises. Document and highlight as appropriate.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Use Case ID** | **Use Case Name** | **Use Case Description** | **Preconditions** | **Postconditions** |
| UC01 | Calculate Tip | Calculates the tip based on user input | User opens the application | Tip amount is displayed on the UI, rounded to two decimal places. |
| UC02 | Invalid Input | Handles invalid input gracefully | User enters non-numeric values | Error message is displayed |
| UC03 | Change Tip Percentage | Allows the user to change the tip percentage while calculating the tip | User enters a valid tip percentage | Tip amount is recalculated and displayed |
| UC04 | Reset Calculator | Allows the user to reset the calculator to its initial state | User clicks on a "Reset" button | All input fields and result labels are cleared |
| UC05 | Zero Bill Amount | Handles the scenario when the bill amount is zero | User enters a bill amount of zero | Tip amount is zero, and a message is displayed |
| UC06 | Negative Bill Amount | Handles the scenario when the bill amount is negative | User enters a negative bill amount | An error message is displayed, and tip calculation is not performed |

Table 5 TipCalculator Defect Log

|  |  |  |
| --- | --- | --- |
| **Use Case ID** | **Defect Name** | **Defect Description** |
|  |  |  |
|  |  |  |

# Lab Summary

This lab has started to bring some documentation into the mix of things by providing clear Use Cases that we can test against and link specific defects back to. As the unit progresses, you will be exposed to more types of documentation and use the knowledge gained from them to help shape your testing practices.