**Workshop Exercises 02-SW-Testing**

**Workshop: Test Plan Creation & SW Specification**

The purpose of this workshop is to expand your experience of constructing test cases that will form the basis of unit tests. From the specification below, you will need to derive a set of functions, and tests that exercise those functions to attempt to identify errors in the code

• Software Specification

The system you will test is an online booking system for a ferrycompany. The specification requires that users can:

• Register to create an account.

• Search for a ferry sailing.

• Book spaces on a sailing.

• Cancel a sailing.

**Workshop: Exercise 1**

Take the software specification above and derive a set of functions (or classes and methods, if you prefer) that can be used to implement the system. You should document:

• The name of the function

• What it does (briefly)

• Its inputs

• Its outputs

• Any errors that might occur

You should have at least one function for each item from the functional specification – though you may have more. We will then discuss your answers before moving on to the next exercise.

*This exercise is about problem decomposition – good program design comes from the ability to take a problem and break it down to its smallest components.*

**Classes:**

User {

Int userID

string Username

string Email

string Password

string [] bookedFerries

}

Sails {

String sailingID

String name

Int dateSailing

Int bookingPrice

Int[] usersBooked

}

**• Register to create an account.**

**• The name of the function**

registerUser(Username, Email, Password)

**• What it does (briefly)**

Takes the user’s details such as username, email and password, to create a new object with the User class, and assigns a new userID to the object, logging the new user under the program with it’s userID

**• Its inputs**

Username, Email, Password

**• Its outputs**

Returns userID to log within the program as a new user object.

**• Any errors that might occur**

* Same email given as another created user.
* One or more parameters are left empty by user.
* User entities wrongly created.
* User entities wrongly logged.
* Duplicate userID’s

**• Search for a ferry sailing.**

**• The name of the function**

searchFerrySailing()

**• What it does (briefly)**

Reads the logged Sails’s entities created from the sail’s class that are represented by their sailingID number, and displays all their details for the user minus the sailingID

**• Its inputs**

none

**• Its outputs**

Returns none but writes sail’s details to UI.

**• Any errors that might occur**

* No sails are read and displayed, and sails entities are created.
* Displays the wrong number of sails entities.
* Displays the wrong details (includes sailingID for example)
* Sailing entities are displayed but no sails have been created yet.

**• Book spaces on a sailing.**

**• The name of the function**

bookSailing(userID, sailingID) {

…

addUserBooking(userID, sailingID)

addUserToFerry(sailingID , userID)

…

}

**• What it does (briefly)**

* bookSailing(userID, sailingID)

Function to call functions addUserBooking(userID, sailingID) and addUserToFerry(sailingID , userID) and pass through the parameters

* addUserBooking(userID, sailingID)

Searches for the User object under the name of the userID passed using a loop function. Once match is made, it appends the sailing ID to the user object’s list variable: bookedFerries.

* addUserToFerry(sailingID , userID)

Similar to addUserBooking(userID, sailingID), but instead searches for the sails object under the name of the sailingID passed using a loop function. Once match is made, it appends the userID to the user object’s list variable: usersBooked.

**• Its inputs**

sailingID respective to the ferry sailing the user clicked on and userID respective to which user is logged on.

**• Its outputs**

All functions don’t return data (for future testing the sub methods of the bookSailing() function could return their object’s list variables to test the userID’s and sailingID’s were successfully added) but writes to the UI relative details within the respective sails object for the user to confirm they successfully booked a ferry.

**• Any errors that might occur**

* userID is not added successfully to the sails object’s list attribute: usersBooked
* sailingID is not added successfully to the user object’s list attribute: bookedFerries
* Function does not write to console the appropriate or any ferry details after sail is booked
* No match is found for passed userID
* No match is found for passed sailingID
* Details added to wrong objects

**• Cancel a sailing.**

**• The name of the function**

removeSailing(userID, sailingID) {

…

removeUserBooking (userID, sailingID ()

removeUserFromFerry(sailingID, userID)

…

}

**• What it does (briefly)**

**• Its inputs**

**• Its outputs**

**• Any errors that might occur**

**Workshop: Exercise 2**

• For each function (or class/method) you outlined in the previous exercise, design a set of tests for each function’s operation. The information must be formally specified as a test plan. Once you have completed the task we will talk through your ideas.

*Think about how you set out the information describing each test – refer to the Lectures 1&2 to recall how to do this.*