1. Project Title and Authors **1**

2. Preface **1**

3. Instruction **1**

4. Black-box Testing **1**

5. White-box Testing **5**

1. Project Title and Authors

Team No. 52
Carlos Subramanian Vidal (cksubram@usc.edu),
Christian Addei (caddei@usc.edu),
Henry Sazo (hsazo@usc.edu)

2. Preface

This document discusses the testing strategies and cases (both blackbox and whitebox) we have used to test our project, alongside the bugs we encountered and the steps we took to solve them.

3. Instruction

a. Run their designated test file for all white box and black box test cases.

4. Black-box Testing

a. Test Cases 1-5 - Carlos, 6-10 Henry, 11-15 Christian

1)

i) Folder: com.example.cs310project2 (androidTest), File: BlackBoxTest1, Test: testAddingFriendUpdatesFriendList

ii) On the friends page see the correct number of both suggested friends (1) and current friends based on interests (1), add a friend (hsazo), the page should then update to show this user as a friend. To execute run the tests in the file “BlackBoxTest1”

iii) The input was chosen because the one user selected (cksubram) which has all interests selected, should see all users (which have at least one interest) as potential friends. The selected user should then be able to add a friend from this list (which is the selected input).

iv) This test case helped uncover a bug that when adding a friend the program would crash. The bug has been fixed, it was fixed by looking at the code which updated the other user on the database and the update command/line was moved inside the event listener.

2)

i) Folder: com.example.cs310project2 (androidTest), File: BlackBoxTest1, Test: testRemovingFriendUpdatesFriendList

ii) On the friends page, see the correct number of both suggested friends and current friends based on interests, remove a friend (caddei), page should then update to show this user as no longer a friend. To execute run the tests in the file “BlackBoxTest1”

iii) The input was chosen because the one user selected (cksubram) has a friend. The selected user should then be able to see that friend on the list and remove them.

iv) This test helped uncover a bug that when removing a friend the program would show the same user over and over again in the friends list. The bug was resolved by changing the value listener type in the friends class when building the view.

3)

i) Folder: com.example.cs310project2 (androidTest), File: BlackBoxTest1, Test: testNavigationToHomeFromFriends

ii) On the friends page press on the home page button, the screen should transfer to that. To execute run the tests in the file “BlackBoxTest1”

iii) This test case was selected as it is important that there is a way to go from the friends page to the meetings page, which is what this input covers.

iv) A bug was found in this test case in that the button for the home page was too small, and could therefore not be clicked by the tester. This bug still has not been resolved as when used by a user clicking on the button works fine.

4)

i) Folder: com.example.cs310project2 (androidTest), File: BlackBoxTest1, Test: testNavigationToProfileFromFriends

ii) On the friends page, press on the profile page button, the screen should transfer to that. To execute run the tests in the file “BlackBoxTest1”

iii) This test case was selected as it is important that there is a way to go from the friends page to the profile page, which is what this input covers.

iv) No bugs were found in from this test case

5)

i) Folder: com.example.cs310project2 (androidTest), File: BlackBoxTest1, Test: testSendEmail

ii) On the friends page, press on the send invitation button, the screen should transfer to the send email page, enter an email address and then click send, the page should then switch to gmail. To execute run the tests in the file “BlackBoxTest1”

iii) This test case was selected to ensure that an invite email can be sent to a potential new user by a current user, and that a new account is generated for this user, hence the test for both items.

iv) Bugs found for this test include the fact that the email intent was not being created and that the user was not being generated. This bug has been resolved, alongside bugs relating to the creation of the new user.

6)

i) Folder: com.example.cs310project2 (androidTest), File: BlackBoxTest2, Test: testEmailInput

ii) Here we create an arbitrary test email address “hsazo@usc.edu”. What this test case does is check if the email we entered passes the validation of only USC affiliate emails being allowed to enter into the application To execute run the tests in the file “BlackBoxTest2”

iii) The input was chosen because it is one of our testers affiliate emails that we know is valid. So it meets the criteria of being able to enter into the app.

iv) This test case did not discover any bugs when run multiple times.

7)

i) Folder: com.example.cs310project2 (androidTest), File: BlackBoxTest2, Test: testPasswordInput

ii) Here we create a random and simple password for an email. This test checks to see if what the user is entering into the input is sent to the database. Thus checking if the information we are sending is making it into the database. To execute run the tests in the file “BlackBoxTest2”

iii) There is no hard requirement for the password so we just chose this input because it was unique enough for us to check and see if it was being stored into our cloud based database.

iv) This test case did not discover any bugs when run multiple times.

8)

i) Folder: com.example.cs310project2 (androidTest), File: BlackBoxTest2, Test: testingButtonSignupSignin

ii) Not much input was generated for this test case. Most of it was just checking that the button went to where it was designed to go into. It checked that the sign up page would direct the user to the correct activity page, while the sign in wouldn’t sign in without someone entering their credentials To execute run the tests in the file “BlackBoxTest2”

iii) We generated this test case because in early development we found that some of our buttons' functionalities were mixing their duties and functions together. So we simplified each button to be used for updating and changing activities rather than being used for saving information.

iv) This test case helped us discover that sometimes when no information was passed in and we clicked the sign in button it would log in without any credential checks. We fixed this by removing the hard coded credential checks at the beginning of the function that would always log into the same user without anyone entering any credentials

9)

i) Folder: com.example.cs310project2 (androidTest), File: BlackBoxTest2, Test: testUpdateProfile

ii) Here we are creating simple inputs that are very common for updating information on your profile. So we create some simple inputs and check if the database is receiving them and making sure that they are working. To execute run the tests in the file “BlackBoxTest2”

iii) The basis of this test case is to make sure that we are updating records correctly in the database when a user wants to update specific information on their profile. However, this also checks that only certain types of inputs are acceptable such as integers for the age and strings for password and native speakers.

iv) This test case showed us we did not have a validator for the age section of the updating profile so it would lead to a crash when users would enter non integer values. We fixed that by creating a validator that doesn’t allow anything but numbers to be inputted.

10)

i) Folder: com.example.cs310project2 (androidTest), File: BlackBoxTest2, Test: testSignUpEmailValid

ii) We created this input because it has what is considered to be invalid characters in an email address. This test case should check to see that the email is structurally valid and if not, not allow the user to use that email. To execute run the tests in the file “BlackBoxTest2”

iii) The basis of this test case was to check that it is a valid email being entered. Putting aside whether it is an affiliate email or not, we also wanted to check that not just any email with “@usc.edu” was allowed to be saved as an email.

iv) This test case showed us we didn't have any validation to make sure that no special characters were allowed to be used when entering an email when signing up.Thus, we put a simple regex-like function to check for validation.

11)

i) Folder: com.example.cs310project2 FIle: BlackBox3 Test: testAddNewMeeting

ii) On the meetings page, add a meeting, the page should show available meetings with the new meeting included

iii) Test case was chosen to test if the user could successfully add a meeting to the database using the app.

iv) No bugs were found from this test case.

12)

i) Folder: com.example.cs310project2 FIle: BlackBox3 Test: testReserveSpot()

ii) On the meetings page, reserve a spot on an available meeting. The page should show the meeting with the new user reserved.

iii) Test case was chosen to test if the user could successfully reserve a spot on an available meeting in the database using the app.

iv) NoMatchingViewException: No view matching view.getId()

13)

i) Folder: com.example.cs310project2 FIle: BlackBox3 testCancelSpot()

ii) On the meetings page, cancel a spot on a reserved meeting. The page should show the meeting without the name of the user.

iii) Test case was chosen to test if the user could successfully cancel a reservation on a meeting in the database using the app.

iv) NoMatchingViewException: No view matching view.getId()

14)

i) Folder: com.example.cs310project2 FIle: BlackBox3 testNavigationToProfileFromHome()

ii) On the home page, press the “Profile” button. The screen should show the profile page.

iii) Test case was chosen to test if the user could go to the profile page from the home page with the corresponding button.

iv) PerformException error performing single click

15)

i) Folder: com.example.cs310project2 FIle: BlackBox3 testNavigationToFriendsFromHome()

ii) On the home page, press the “Friends” button. The screen should show the friends page.

iii) Test case was chosen to test if the user could go to the friends page from the home page with the corresponding button.

iv) PerformException error performing single click

5. White-box Testing

a. Test Cases 1-5 - Carlos, 6-10 - Henry, 11-15 - Christian

1)

i) Folder: com.example.cs310project2 (test), File: WhiteBoxTest1, Test: custom_user_creation

ii) Creates an instance of the custom user class and ensures that all data passed in the constructor is correct. To execute it, run the tests in the file “WhiteBoxTest1”.

iii) For each of the members in the constructor a value is passed in, which is then later retrieved and shown to be what was expected.

iv) No bugs have been found from this test.

2)

i) Folder: com.example.cs310project2 (test), File: WhiteBoxTest1, Test: default_user_creation

ii) Creates an instance of the custom user class through the default constructor and ensures that the values are the expected ones. To execute it, run the tests in the file “WhiteBoxTest1”.

iii) This test case uses the default constructor, which initializes an instance of the user class to have certain values, these values are later retrieved from the instance and found to match the expected values.

iv) No bugs have been found from this test.

3)

i) Folder: com.example.cs310project2 (test), File: WhiteBoxTest1, Test: add_friend

ii) Creates an instance of the custom user class, adds a friend to the user and ensures that the friend was correctly added. To execute it, run the tests in the file “WhiteBoxTest1”.

iii) The test case checks whether the friend who was added as a friend to the user is in the friends list of the user and is the only user in that list.

iv) No bugs have been found from this test.

4)

i) Folder: com.example.cs310project2 (test), File: WhiteBoxTest1, Test: remove_friend

ii) Creates an instance of the custom user class, adds a friend to the user, ensures the user has that friend, then removes the friend and checks that the user has no friends. To execute it, run the tests in the file “WhiteBoxTest1”.

iii) The test case checks whether the friend who was added as a friend to the user is in the friends list of the user and is the only user in that list. After removing the friend the case checks that the friend is no longer in the friends list and that the friends list is empty.

iv) No bugs have been found from this test.

5)

i) Folder: com.example.cs310project2 (test), File: WhiteBoxTest1, Test: update_user

ii) Creates a user fensures that its contents are those that are expected, then updates values pertaining to that user and ensures that the user now has the correct updated values. To execute it, run the tests in the file “WhiteBoxTest1”.

iii) The test case checks that the user retrieved from the database is the same as the expected values (which are preset)

iv) No bugs have been found from this test.

6)

i) Folder: com.example.cs310project2 (test), File: WhiteBoxTest2, Test: testEmailEditztext

ii) This test case checks the internal validation that we created is working correctly, when the user is entering their email. So we have a valid email as input but also an invalid email as input to check that the valid one returns true and the invalid one is false. To execute it, run the tests in the file “WhiteBoxTest2”.

iii) The result should be that the invalid email is rejected by the program and the user has to enter a valid email address.

iv) No bugs have been found from this test.

7)

i) Folder: com.example.cs310project2 (test), File: WhiteBoxTest2, Test: testButtonClicks

ii) This test case checks to see that the main 3 buttons found in most of our pages in the applications “friends”, “meetings”, and “profile” work as intended when the user is navigating within the pages. To execute it, run the tests in the file "WhiteBoxTest2".

iii) The result should be that each button takes you to their designated activity page without crashes.

iv) No bugs have been found from this test.

8)

i) Folder: com.example.cs310project2 (test), File: WhiteBoxTest2, Test: testUpdateButtonPress

ii) This test case checks that when the update button is clicked when a user is updating something, we go back to the home page only when information is entered. To execute it, run the tests in the file "WhiteBoxTest2".

iii) The result should be the user goes back to the home page without leaving anything empty within the update form page

iv) No bugs have been found from this test.

9)

i) Folder: com.example.cs310project2 (test), File: WhiteBoxTest1, Test: testEnteredTextInEditTexts

ii) This test checks to see that when the user enteres information into the update part of the application the correct information is being entered and updated. To execute it, run the tests in the file "WhiteBoxTest2".

iii) Updated information on the database.

iv) No bugs have been found from this test.

10)

i) Folder: com.example.cs310project2 (test), File: WhiteBoxTest1, Test: testCheckBoxBehavior

ii) This test checks to see the behavior of the three check boxes used for interests works as anticipated. To execute it, run the tests in the file "WhiteBoxTest1".

iii) All check boxes should be checked thus they should all return true

iv) No bugs have been found from this test.

11)

i) Folder: com.example.cs310project2 FIle: WhiteBox3 custom_meeting_creation()

ii) Create instance meeting class and ensure parameters are valid.

iii) Test case passed.

iv) No bugs were found from this test case.

12)

i) Folder: com.example.cs310project2 FIle: WhiteBox3 create_and_cancel()

ii) Create instance meeting class and ensure parameters are valid. Then, cancel the meeting.

iii) Test case passed.

iv) No bugs were found from this test case.

13)

i) Folder: com.example.cs310project2 FIle: WhiteBox3 reserve_spot()

ii) Reserve a spot in an existing meeting class.

iii) Test case did not pass.

iv) Number of members in meeting should be 0, but there is 1.

14)

i) Folder: com.example.cs310project2 FIle: WhiteBox3 cancel_spot()

ii) Reserve a spot in an existing meeting class. Then cancel the reservation of the same meeting.

iii) Test case passed.

iv) No bugs were found from this test case.

15)

i) Folder: com.example.cs310project2 FIle: WhiteBox3 reserve_twice()

ii) Reserve a spot in an existing meeting class. Then attempt to reserve a spot into the same meeting.

iii) Test case passed.

iv) No bugs were found from this test case.

c. Explain the coverage criteria used in your white-box testing and what coverage level you are able to achieve in the end. *

For our white-box testing approach, we employed statement coverage as our chosen coverage criterion. This decision was based on the belief that statement coverage provides a comprehensive measure of the extent to which our low-level code has been executed during testing. By aiming for high statement coverage, we sought to ensure that each line of code in the tested components was exercised, helping us identify and address any potential bugs or issues that might arise at the code level.

In summary, our choice of statement coverage as the coverage criterion for white-box testing was driven by the desire to rigorously examine the execution of individual code statements, providing a solid foundation for ensuring the correctness and reliability of our low-level code.