Learning Outcome 2 Supporting Document

During the development of the HARty system, I followed a variation of the Test-Driven Design (TDD) methodology. TDD is a design methodology in software development which involves writing tests for a feature, before actually implementing the feature. By doing this, it becomes easier to implement code that models the correct behaviour.

In the case of HARty, instead of writing the code required for testing a feature, I chose to write the goal and results of the test out on paper before implementing the feature. Although it is a little bit of a shortcut, I had limited time to do the work and this approach worked well for me. This approach allowed me to clearly plan out the expected behaviour of the feature, and helped me identify all the relevant edge cases before starting the implementation.

Attached below is my test plan for some of the requirements of HARty discussed in LO1 . Each test plan has been treated independently, so that it can reflect the real-world testing where multiple testers handle different requirements. To provide as much detail as possible, some of the information has been repeated in multiple test plans.

Email Address Change (Functional Requirement) Test Plan:

Project Phase - Design and execute subsystem tests

Test Objective - Test the functionality of changing the email address associated with a User's HARty account in the Android application.

Type of Testing - Functional Testing

Time Estimate - 2 hours

Priorities and Pre-requisites of Feature :-

- The user has to be logged into their account, before they can change their email address.
- The user should be able to change the email address successfully in a user-friendly and accessible manner.
- The updated email address should be saved properly in the Firebase Authentication service, and displayed in the Profile UI.

Instrumentation and Scaffolding Required :-

- Android Studio Emulator - This is used to run the HARty application on a virtual device, and test the functionality of changing the email.
- Espresso Test Framework - This is used to write automated UI tests, for verifying the functionality of changing the email.
- JUnit - This is used to write a unit testing framework, which is used for verifying the business logic of changing the email.

Positive Test Cases :-

- The user is able to change the address successfully.
    - Input - A valid email address, and the correct account password.
    - Output - Toast message of success, and redirection to the Profile UI.

Negative Test Cases :-

- The user is unable to change the email address if the email address provided is already in use.
    - Input - A valid email address already in use by another user, and the correct account password.
    - Output - Toast message of failure.
- The user is unable to change the email address if the email address is not valid.
    - Input - An invalid email address, and the correct account password.
    - Output - Toast message of failure.
- The user is unable to change the email address if either the email address or current password provided are empty.

    - Input - Either the valid email, the correct account password or both are empty.
    - Output - Toast message of failure.
- The user is unable to change the email address if no new email address is provided.
    - Input - User's current email address, and the correct account password.
    - Output - Toast message of failure.
- The user is unable to change the email address if they are not able to authenticate their request.
    - Input - A valid email address, and the wrong account password.
    - Output - Toast message of failure.

Validation and Verification :-

- The Espresso test cases will validate the tests by checking if the expected output is produced in the UI.
- The JUnit test cases will validate if a valid email address has been provided, and validate the tests by checking if the expected output is produced in the backend.

Potential Vulnerabilities / Omissions :-

- The test plan does not consider the risk of the network connectivity issues during the change of the email address process. This vulnerability can be ignored, as the virtual device created in the Android Studio Emulator ensures a clean and reliable network connection every time.
- The test plan does not consider the security and confidentiality risks related. This vulnerability can be ignored, as the Firebase API used to perform this action has proven to be secure while keeping the user data confidential. Only stakeholders with access to the hash parameters, can decrypt the data.

Evaluation of Instrumentation - The test plan provides a comprehensive approach for testing the email address change feature in HARty's Android application. The test plan describes in detail the different tests to create, the I/O to expect, and also manages to identify potential vulnerabilities. The instrumentation used to test the code, checks all the different use cases that can occur, and also cleans up any redundant data created during the testing phase. Therefore, I believe that the test plan is good, and meet all the checkpoints.

Password Change (Functional Requirement) Test Plan:

Project Phase - Design and execute subsystem tests

Test Objective - Test the functionality of changing the password associated with a User's HARty account in the Android application.

Type of Testing - Functional Testing

Time Estimate - 2 hours

Priorities and Pre-requisites of Feature :-

- The user has to be logged into their account, before they can change their password.
- The user should be able to change the password successfully in a user-friendly and accessible manner.
- The updated password should be saved properly in the Firebase Authentication service.

Instrumentation and Scaffolding Required :-

- Android Studio Emulator - This is used to run the HARty application on a virtual device, and test the functionality of changing the password.
- Espresso Test Framework - This is used to write automated UI tests, for verifying the functionality of changing the password.
- JUnit - This is used to write a unit testing framework, which is used for verifying the business logic of changing the password.

Positive Test Cases :-

- The user is able to change the password successfully.

    - Input - A new password greater than or equal to 6 characters, and the correct account password.
    - Output - Toast message of success, and redirection to the Profile UI.

Negative Test Cases :-

- The user is unable to change the password if it is less than 6 characters.
    - Input - A new password greater less 6 characters, and the correct account password.
    - Output - Toast message of failure.
- The user is unable to change the password if either the new password or current password provided are empty.
    - Input - Either the new password, the correct account password or both are empty.
    - Output - Toast message of failure.
- The user is unable to change the email address if they are not able to authenticate their request.
    - Input - A new password greater than or equal to 6 characters, and the wrong account password.
    - Output - Toast message of failure.

Validation and Verification :-

- The Espresso test cases will validate the tests by checking if the expected output is produced in the UI.
- The JUnit test cases will validate if a valid password has been provided, and validate the tests by checking if the expected output is produced in the backend.

Potential Vulnerabilities / Omissions :-

- The test plan does not consider the risk of the network connectivity issues during the change of the password process. This vulnerability can be ignored, as the virtual device created in the Android Studio Emulator ensures a clean and reliable network connection every time.
- The test plan does not consider the security and confidentiality risks related. This vulnerability can be ignored, as the Firebase API used to perform this action has proven to be secure while keeping the user data confidential. Only stakeholders with access to the hash parameters, can decrypt the data.

Evaluation of Instrumentation - The test plan provides a comprehensive approach for testing the password change feature in HARty's Android application. The test plan describes in detail the different tests to create, the I/O to expect, and also manages to identify potential vulnerabilities. The instrumentation used to test the code, checks all the different use cases that can occur. Therefore, I believe that the test plan is good, and meet all the checkpoints.

User Authentication (Functional Requirement) Test Plan:

Project Phase - Design and execute subsystem tests

Test Objective - Test the functionality of authenticating in and out of HARty's Android Application.

Type of Testing - Functional Testing

Time Estimate - 3 hours

Priorities and Pre-requisites of Feature :-

- The user has to have a registered account, before they are able to log in.
- The user should be able to log in and out successfully, in a user-friendly and accessible manner.
- The user has to be logged in, before they are able to log out.
- The user has provided a valid email and password greater than 6 characters.

Instrumentation and Scaffolding Required :-

- Android Studio Emulator - This is used to run the HARty application on a virtual device, and test the functionality of authenticating.

- Espresso Test Framework - This is used to write automated UI tests, for verifying the functionality of authenticating.
- JUnit - This is used to write a unit testing framework, which is used for verifying the business logic of authenticating.

Positive Test Cases :-

- The user is able to log into the Android application successfully.
  - Input - A registered email address, and the correct account password.
  - Output - Toast message of success, and redirection to the Profile UI.
- The user is able to log out of the Android application successfully.
  - Input - N/A
  - Output - Redirection to the Profile UI.

Negative Test Cases :-

- The user is unable to log into their account, if either the email address or password do not correspond to an account.
  - Input - A combination of email address or password that are wrong or do not exist.
  - Output - Toast message of failure.
- The user is unable to log into their account if either the email address or password provided are empty.
  - Input - Either the email address, the correct account password or both are empty.
  - Output - Toast message of failure.

Validation and Verification :-

- The Espresso test cases will validate the tests by checking if the expected output is produced in the UI.
- The JUnit test cases will validate if a valid email and password has been provided, and validate the tests by checking if the expected output is produced in the backend.

Potential Vulnerabilities / Omissions :-

- The test plan does not consider the risk of the network connectivity issues during the authentication process. This vulnerability can be ignored, as the virtual device created in the Android Studio Emulator ensures a clean and reliable network connection every time.
- The test plan does not consider the security and confidentiality risks related. This vulnerability can be ignored, as the Firebase API used to perform this action has proven to be secure while keeping the user data confidential. Only stakeholders with access to the hash parameters, can decrypt the data.

Evaluation of Instrumentation - The test plan provides a comprehensive approach for testing the authentication feature in HARty's Android application. The test plan describes in detail the different tests to create, the I/O to expect, and also manages to identify potential vulnerabilities. The instrumentation used to test the code, checks all the different use cases that can occur. Therefore, I believe that the test plan is good, and meet all the checkpoints.

Account Creation (Functional Requirement) Test Plan:

Project Phase - Design and execute subsystem tests

Test Objective - Test the functionality of creating a user's HARty account through the Android Application.

Type of Testing - Functional Testing

Time Estimate - 1.5 hours

Priorities and Pre-requisites of Feature :-

- The user does not have an existing HARty account.
- The user should be able to create an account successfully, in a user-friendly and accessible manner.

• The user has provided a valid email and password greater than 6 characters.

Instrumentation and Scaffolding Required :-

- Android Studio Emulator - This is used to run the HARty application on a virtual device, and test the functionality of creating a HARty account.
- Espresso Test Framework - This is used to write automated UI tests, for verifying the functionality of creating a HARty account.
- JUnit - This is used to write a unit testing framework, which is used for verifying the business logic of creating a HARty account.

Positive Test Cases :-

- The user is able to create a new HARty account through the Android application successfully.
  - Input - A name, email address, and password.
  - Output - Toast message of success, and redirection to the Login UI.

Negative Test Cases :-

- The user is unable to create a new HARty account, if the email provided already exists.
  - Input - A name, an email address already linked to an existing HARty account, and a password.
  - Output - Toast message of failure.
- The user is unable to create a new HARty account if either the name, email address or password provided are empty.
  - Input - Either the name, email address, password or all are empty.
  - Output - Toast message of failure.

Validation and Verification :-

- The Espresso test cases will validate the tests by checking if the expected output is produced in the UI.
- The JUnit test cases will validate if a valid email and password has been provided, and validate the tests by checking if the expected output is produced in the backend.

Potential Vulnerabilities / Omissions :-

- The test plan does not consider the risk of the network connectivity issues during the authentication process. This vulnerability can be ignored, as the virtual device created in the Android Studio Emulator ensures a clean and reliable network connection every time.
- The test plan does not consider the security and confidentiality risks related. This vulnerability can be ignored, as the Firebase API used to perform this action has proven to be secure while keeping the user data confidential. Only stakeholders with access to the hash parameters, can decrypt the data.

Evaluation of Instrumentation - The test plan provides a comprehensive approach for testing the account creation feature in HARty's Android application. The test plan describes in detail the different tests to create, the I/O to expect, and also manages to identify potential vulnerabilities. The instrumentation used to test the code, checks all the different use cases that can occur, and also cleans up any redundant data created during the testing phase. Therefore, I believe that the test plan is good, and meet all the checkpoints.

Account Deletion (Functional Requirement) Test Plan:

Project Phase - Design and execute subsystem tests

Test Objective - Test the functionality of deleting a user's HARty account through the Android Application.

Type of Testing - Functional Testing

Time Estimate - 1 hour

Priorities and Pre-requisites of Feature :-

• The user has to be logged into their account, before they can delete their account.

- The user should be able to delete their account successfully, in a user-friendly and accessible manner.
- The user has provided a valid email and password greater than 6 characters.

Instrumentation and Scaffolding Required :-

- Android Studio Emulator - This is used to run the HARty application on a virtual device, and test the functionality of deleting a HARty account.
- Espresso Test Framework - This is used to write automated UI tests, for verifying the functionality of deleting a HARty account.
- JUnit - This is used to write a unit testing framework, which is used for verifying the business logic of deleting a HARty account.

Positive Test Cases :-

- The user is able to delete their HARty account through the Android application successfully.
  - Input - The email address and password of an existing HARty account.
  - Output - Toast message of success, and redirection to the Profile UI.

Negative Test Cases :-

- The user is unable to delete their HARty account, if the email address and/or password provided do not belong to the account they are logged in as.
  - Input - A valid address and password that are different than the account's.
  - Output - Toast message of failure.
- The user is unable to delete their HARty account if either the email address or password provided are empty.
  - Input - Either the email address, password or both are empty.
  - Output - Toast message of failure.

Validation and Verification :-

- The Espresso test cases will validate the tests by checking if the expected output is produced in the UI.
- The JUnit test cases will validate if a valid email and password has been provided, and validate the tests by checking if the expected output is produced in the backend.

Potential Vulnerabilities / Omissions :-

- The test plan does not consider the risk of the network connectivity issues during the authentication process. This vulnerability can be ignored, as the virtual device created in the Android Studio Emulator ensures a clean and reliable network connection every time.
- The test plan does not consider the security and confidentiality risks related. This vulnerability can be ignored, as the Firebase API used to perform this action has proven to be secure while keeping the user data confidential. Only stakeholders with access to the hash parameters, can decrypt the data.

Evaluation of Instrumentation - The test plan provides a comprehensive approach for testing the account deletion feature in HARty's Android application. The test plan describes in detail the different tests to create, the I/O to expect, and also manages to identify potential vulnerabilities. The instrumentation used to test the code, checks all the different use cases that can occur, and also cleans up any redundant data created during the testing phase. Therefore, I believe that the test plan is good, and meet all the checkpoints.

Shared Preferences (Functional Requirement) Test Plan:

Project Phase - Code and integration

Test Objective - An example test plan, which tests if the HARty application is interacting with the Shared Preferences Android file as expected.

Type of Testing - System Testing

Time Estimate - 0.5 hours (More time required, if more instances of testing the Shared Preferences file is created)

Priorities and Pre-requisites of Feature :-

- N/A

Instrumentation and Scaffolding Required :-

- Android Studio Emulator - This is used to run the HARty application on a virtual device, and test the objective.
- Espresso Test Framework - This is used to write automated UI tests, for verifying the objective.
- JUnit - This is used to write a unit testing framework, which is used for verifying the business logic of the objective.

Positive Test Cases :-

- Manipulating the Shared Preferences file, correctly reflects the changes in the Android application.
  - Input - Change step counter target in the Shared Preferences file.
  - Output - Step target is changed in the Android application.
- Manipulating the Android application, correctly reflects the changes in the Shared Preferences file.
  - Input - Change step counter target in the Android application.
  - Output - Step target is changed in the Shared Preferences file.

Negative Test Cases :-

- N/A

Validation and Verification :-

- The Espresso test cases will validate the tests by checking if the expected output is produced in the UI.
- The JUnit test cases will validate the tests by checking if the expected output is produced in the backend.

Potential Vulnerabilities / Omissions :-

- The test plan does not consider the security and confidentiality risks involved. This vulnerability should be investigated later, as any private or confidential information stored in the Shared Preferences file could possibly exposed.

Evaluation of Instrumentation - The test plan provides a comprehensive approach for testing the interaction between HARty's Android application and it's Shared Preferences file. The test plan describes in detail the different tests to create, the I/O to expect, and also manages to identify potential vulnerabilities. The instrumentation used to test the code, achieves the required goal but it could be improved to test more interactions. Therefore, I believe that the test plan is good, but it could be made a little better if more time was available.

Accuracy of ML models (Quality Requirement) Test Plan:

Project Phase - Code and integration

Test Objective - Test the accuracy of the HAR algorithms. In this test plan we focus on testing and improving the accuracy of the Essential Features HAR model.

Type of Testing - Quality Testing

Time Estimate - 10 hours

Priorities and Pre-requisites of Feature :-

- The Essential Features algorithm should be ready.

Instrumentation and Scaffolding Required :-

- Jupyter Notebook - This is used to run the Leave One Out Cross Validation method (LOOCV, which is used to evaluate and improve the model.
- Essential Features HAR algorithm - This is the basic model which we test and improve on.
- Dataset of activities - This is the data used to train our ML model.

Test Cases :-

- Use LOOCV to determine the average accuracy of the model, repeat this until the accuracy of the HAR algorithm is greater than 95%.
- Test the HAR algorithm on previously unseen data, and ensure accuracy is greater than 95%.

Validation and Verification :-

- The LOOCV will validate our model and help us obtain robust accuracy rate. In LOOCV, each data point is used once as the training set while the model is trained on the remaining data points. This essentially provides us with an estimate of how well the model will perform on unseen data. By averaging this performance across all iterations of LOOCV, we get a more reliable and robust accuracy rate.
- Testing on previously unseen data allows us to see how the HAR algorithm performs, outside the development atmosphere. We are able to realise the true accuracy of the model.

Potential Vulnerabilities / Omissions :-

- The test plan does not consider the accuracy of the All Features HAR algorithm. This is because the All Features model will be tested in a similar way in another test plan.

Evaluation of Instrumentation - The test plan provides a comprehensive approach for testing and validating the accuracy rate of the Essential Features HAR algorithm. Using LOOCV and testing on previously unseen data, we are able to see how our ML model might perform in the real world compared to the development environment.

User Acceptance (Quality Requirement) Test Plan:

Project Phase - Just before Product Delivery. Enough time before Product Delivery should be left to make any improvements or changes suggested by end users.

Test Objective - Give a completed version of the HARty system to a group of end users, who will be able to use it in a real-world setting. We can then obtain feedback from them regarding the functionality, usability, and overall performance of the system

Type of Testing - System Testing

Time Estimate - 1 week

Priorities and Pre-requisites of Feature :-

- The system must be fully developed and throughly tested.
- All known errors should be fixed.
- The system should be checked against the list of requirements to ensure all of the interests of the stakeholder have been implemented.

Instrumentation and Scaffolding Required :-

- Android Device - The Android device is used to run the HARty application, and it must have an operating system of Android 11 or greater.
- Respeck - The Respeck IMU sensor is used to provide the relevant information to the Android application.

Validation and Verification :-

- Users have the opportunity to go through the system, and analyse if all the agreed functional and non-functional requirements have been met.

Potential Vulnerabilities / Omissions :-

- End users do not have the technical expertise to thoroughly test the system, so some security, performance or other such issues could go unnoticed.

- End users are under a time constraint, which can lead to incomplete testing of the system. This can result in potential vulnerabilities and omissions from being discovered.
- Due to limited knowledge, end users may assume the existence of a certain functionality in the system, while in reality it is not present at all. This can mean that some requirements might not come into notice.
- UAT is usually performed in a small group of end users, who do not necessarily represent the entire user base of the system. This can result the acceptance of the system to vary, as acceptance is subjective.
- End users might use the system in an unrealistic manner or testing environment. This can lead to the result of this testing to be wrong or misleading.

Evaluation of Instrumentation - The test plan provides a comprehensive approach for testing the HARty system from the perspective of the end users. By completing this test we will be able to identify how well our system works in the real world, and if the system meets the requirements and needs of potential end users. However this type of testing does come with little flaws that may produce misleading results. This is due to the end users lacking a deep understanding and knowledge of the system.

Application Demand (Quality Requirement) Test Plan:

Project Phase - Design and execute system tests

Test Objective - Tests the HARty system, to see if the common Firebase service can handle a large number of concurrent number of users.

Type of Testing - System Testing

Time Estimate - 4 hours

Priorities and Pre-requisites of Feature :-

- The system must be fully developed and throughly tested.
- All known errors should be fixed.
- The system should be checked against the list of requirements to ensure all of the interests of the stakeholder have been implemented.

Instrumentation and Scaffolding Required :-

- Android Studio Emulator - This is used to run the HARty application on a virtual device, and test the objective.
- Espresso Test Framework - This is used to write automated UI tests, for verifying the objective.
- JUnit - This is used to write a unit testing framework, which is used for verifying the business logic of the objective.

Test Cases :-

- Determine how changing the number of concurrent users affect the system as a whole.

Validation and Verification :-

- The Espresso test cases will validate the tests by checking if the expected output is produced in the UI.
- The JUnit test cases will validate the tests by checking if the expected output is produced in the backend.

Potential Vulnerabilities / Omissions :-

- The test plan does not consider the security and confidentiality risks involved. This vulnerability can be ignored, as within this test we are focusing on working with concurrent users and not about user data.

Evaluation of Instrumentation - The test plan provides a comprehensive approach for testing the reliability of the system. The test plan describes in detail the test to create, instrumentation and scaffolding to consider, and also manages to identify potential vulnerabilities. Therefore, I believe that the test plan is good, and meet all the checkpoints.

Validation (Functional Requirement) Test Plan:

Project Phase - Code and integration

Test Objective - Test the user input (Email, Password, and MAC address) to determine the validity of it.

Type of Testing - Unit Testing

Time Estimate - 1 hour

Priorities and Pre-requisites of Feature :-

- An email address is considered valid, if it matches the agreed pattern in the Specification document.
- A password is considered valid, if it is greater than 6 characters.
- A MAC address is considered valid, if it matches the agreed pattern in the Specification document.

Instrumentation and Scaffolding Required :-

- JUnit - This is used to write a unit testing framework, which is used for verifying the business logic of validating input.

Positive Test Cases :-

- The email address is valid.
  - Input - An email address that matches the agreed pattern.
  - Output - True.
- The password is valid.
  - Input - A password that is greater than 6 characters.
  - Output - True.
- The MAC address is valid.
  - Input - A MAC address that matches the agreed pattern.
  - Output - True.

Negative Test Cases :-

- The email address is not valid.
  - Input - An email address that does not match the agreed pattern.
  - Output - False.
- The password is not valid.
  - Input - A password that is shorter than 6 characters.
  - Output - False.
- The MAC address is not valid.
  - Input - A MAC address that does not match the agreed pattern.
  - Output - False.

Validation and Verification :-

- The JUnit test cases will validate if the user input that has been provided, and validate the tests by checking if the expected output is produced in the backend.

Potential Vulnerabilities / Omissions :-

- N/A

Evaluation of Instrumentation - The test plan provides a comprehensive approach for testing the Validation unit in the Android application. The test plan describes in detail the different tests to create, and the I/O to expect. The instrumentation used to test the code, checks all the different use cases that can occur. With this particular test plan, I have been unable to identify any particular vulnerability or omission, as this unit of code is not too complex. Therefore, I believe that the test plan is good, and meet all the checkpoints.

Evaluation of Instrumentation:

The process of creating the instrumentation of the tests using the provided test plans was easy and required no additional input or scaffolding to implement. I simply followed the guidelines provided by the tests plans and was able to implement the tests effectively. The end result of this was a very successful run, with the codebase passing every single test. I also received positive feedback from the user testing, which confirmed that I had met all the required needs of the end users and other stakeholders within the project.

I do not believe that the instrumentation needs to be improved, because of the successful test run and the identification of vulnerabilities and omissions beforehand. However, there is still a need to create additional tests to fully test all the functionality and quality requirements in the HARty system. Unfortunately, due to the time constraints, I was only able to create 10 test plans and 38 test cases.