Learning Outcome 5 Supporting Document

Review Criteria:

Applying review criteria to our code, helps us identify issues in out code. It helps us improve our code quality and identify any potential issues. To better explain this question, I am going to be reviewing the SignUpActivity (which can be found in my GitHub repository) from the HARty Android application against the review criteria I identified.

Readability:

- SignUpActivity makes use of meaningful and descriptive variable names. Examples of these include the names of the variables *signUpButton*, *loginButton*, *name*, *email*, *password* and *auth* which are appropriate and descriptive.
- The code for SignUpActivity appears to be well formatted and indented, making it easier to read and work with.

Maintainability:

- SignUpActivity contains no code duplication, making it easier to maintain.
- The code complexity for the SignUpActivity looks manageable and easy to understand, therefore making it easier to maintain, troubleshoot or add further features to.
- The code for the SignUpActivity is organised into relevant functions and methods, making it easier to maintain.

Functionality:

- The code for the SignUpActivity checks for input validation using functions from the custom Validator class. This helps to prevent incorrect or malicious input from being processed by the HARty Android application.
- Error handling is implemented in the form of Toast messages within the SignUpActivity. When an error occurs Toast messages are displayed to the end user, this is done to provide them with the information on the particular error that has occurred.

Security:

- When a user completes signing up for the HARty Android system, the email addresses and passwords are stored in an encrypted manner on the secure Firebase Authentication system.
- The code for the SignUpActivity checks for input validation as described above. This helps to prevent incorrect or malicious input from being processed or stored by the HARty Android application.

Review Techniques:

- Automated Code Analysis - This review technique occurs when the code is analysed using automated tools, such as linters, to identify potential issues. Linters are usually built into the IDE, in my case this was Android Studio.
- Pair Programming - This review technique occurs when two developers look and work on a piece of code together. This helps them identify any potential errors and quality issues, which they might has missed earlier on.
- Peer Review - This review technique occurs when code you have written is reviewed by other development team members to ensure that it does not have any errors and meets the expected standards.

Construction of CI Pipeline:

An appropriate Continuous Integration (CI) pipeline for HARty should ensure that the code is built, tested, and deployed regularly and in an efficient manner. The CI pipeline should also ensure that code quality is maintained throughout the code base and any potential issues or errors are detected and resolved quickly.

To construct an appropriate CI pipeline for HARty, we need to consider the following steps:

1. Code Repository - The first step is to set up a code repository for the HARty Android application, which can be achieved using Git. The code repository will be where all the code changes will be made, and from where the CI pipeline will retrieve the code from.
2. CI Server - The next step is to set up Continuous Integration pipeline server to automate the build and test process, which can be achieved using Jenkins. The CI server should then be configured to poll the code repository at regular intervals to detect changes in our codebase. Whenever changes are detected in the code repository, the CI server should automatically trigger a build phase.
3. Build Phase - The next step is to set up a build environment, during which the Android code will be compiled, and the generated APK file be installed and tested on a Virtual Android Device. This can be achieved through a locally installed build environment such as Jenkins or a cloud based build environment.
4. Testing - After the build phase is complete, the CI server should run a set of automated tests which will ensure that the Android application is functioning correctly. We can use the Espresso and JUnit testing frameworks to create unit, integration and system tests.
5. Code Quality Check - The next step is to set up the CI pipeline to conduct a code quality check, where code quality checks such as listing, and formatting are used to ensure that the codebase follows the best programming practices and meets the required standards.
6. Deployment - If the tests and the code quality check are passed, the CI server should deploy the APK file to the Google Play Store or other Android application distribution channels.
7. Notifications - The CI server should then be configured to send notifications to the development team (Me in the case of HARty) indicating the results of the build and test process, including any information about the errors or failures.

The Continuous Integration pipeline should be configured to allow instances of it to run in parallel of each other. This reduces the overall time taken for the build and test processes, and makes most efficient use of the resources. In addition to this the design of the CI pipeline for HARty should be scalable and flexible. It should allow for the easy integration of new tests, deployment channels and code analysis tools as the Android application continues to be developed.

Automation of Testing:

To automate some aspects of the testing for HARty, the following steps can be followed:

1. Integrate with CI/CD Pipeline - The first step is to take our existing test suite created in LO2, and integrating this automated test suite with the CI pipeline. The CI pipeline will be responsible for building, testing and deploying the Android application as desired above. To integrate these test, we can use tools such as GitLab CI, or Jenkins.
2. Configure Test Environment - The next step is to set a test environment, so that tests can be run in a controlled testing environment.
3. Executing Tests - Once the test environment is configured, the automated test suite should be executed as part of the CI pipeline. The tests should be automatically run whenever code changes are made to source code repository.
4. Maintenance of Test Suite / Environment - The final step is to maintain the automated test suite and the test environment regularly to ensure that the necessary functionalities are being tested, and that the code coverage is met.

The required level of testing in a CI environment is usually dependent on the specific requirements and expectations of a project. In the case of HARty, the testing in a CI environment should include unit, integration, functional, and system tests. However the core focus is likely to be the functional tests.

CI Pipeline Functions:

Expected Functions of my proposed CI pipeline include:

- Code Integration - The CI pipeline should continuously integrate changes from multiple branches and developers to the main source code repository.
- Build and Compilation - The CI pipeline should automatically build and compile the Android application, whenever a change is detected in the source code repository.
- Automation of Test Suite - The CI pipeline should automatically run the automated test suite, when the Android application has finished the Build and Compile stage.
- Code Quality Checks - The CI pipeline should then run code quality checks, to ensure that the changes made to the source code repository meet the required standards.
- Automation of Deployment - The CI pipeline should automatically deploy the APK of the Android application if all the checks and tests pass.

Possible issues that can be identified by proposed CI pipeline include:

- Compilation Errors - The CI pipeline should detect any errors that occur during the build and compilation stages. This includes syntax errors, type mismatches, and missing files or dependencies.
- Test Failures - The CI pipeline should detect any issues with the testing process, this includes failing tests or the errors in running the test suite.
- Code Quality Issues - The CI pipeline should detect any code quality issues in the source code repository. This includes low code coverage, security vulnerabilities or not meeting the required programming standards.

Examples of how these potential issues can be identified by the proposed CI pipeline:

- Compilation Errors - The CI pipeline should be configured to run the Build and Compilation stage automatically, and to fail if an error occurs. The CI pipeline should then produce a detailed report of the errors and make it available to the relevant parties in the development team (Me in the case of HARty).
- Test Failures - The CI pipeline should be configured to run the automated test suite after the Build and Compilation stage, and to fail if any of the tests fail. The CI pipeline should then produce a detailed report of the test results, including detailed information about the failed test cases and make it available to the relevant parties in the development team (Me in the case of HARty).
- Code Quality Issues - The CI pipeline should be configured to run the Code Quality Checks after the test suite, and to fail if any issues regarding the quality of the codebase are detected. The CI pipeline should then produce a detailed report of the code quality issues, including the location and type of error, and make it available to the relevant parties in the development team (Me in the case of HARty).