Faculty of Engineering and Applied Science

ENGR 4941U Capstone Systems Design for ECSE II

Design and Development of a Continuous Integration Framework for Automotive Software Development

R5: Final Engineering Report

Team Members

Taha Akhtar, 100704774

Rushi Amin, 100701929

Umar Malik, 100705241

Jayash Singh, 100695587

Lefrancois Valenski, 100700573

Faculty Advisor: Dr. Akramul Azim

Capstone Coordinator (Fall 2021 & Winter 2022): Dr. Q. Mahmoud

Table of Contents

1: Executive Summary 6

2: Report 1 6-12

2.1: Problem Identification 6-7

2.2: Background and Research Review 7-8

2.3: Design Process 8-9

2.4: Scenarios and Use Cases 9-10

2.5: Stakeholder Requirements and Traceability Matrix 10-12

2.6: Definition of Acceptance Tests 12

3: Report 2 12-20

3.1: Concept Generation and Analysis 12-16

3.2: Conceptual System Design 17-18

3.3: Definition of Integration Testing 18-20

3.4: Estimated Costs 20

4: Revised design report 20-25

4.1: Detailed Design 20-24

4.2: Unit and Integration Testing 24-25

5: Updated Project Plan 25-28

6: Test Results 28-32

6.1: Integration Testing 28

7: Ethical Considerations 32

8: Safety Considerations 32-33

9: Conclusions 33

10: Acknowledgements 34

11: References 35-36

12: Appendices 37

13: Contribution Matrix 38

List of Tables

Table	Page
2.1: Build Of Materials (BOM)	9
2.2: Traceability Matrix	11-12
3.1 Test Cases	19-20
3.2: Updated Build Of Materials (BOM)	20
6.1 Updated Test Cases	29-32
13.1: Contribution matrix	

List of Figures

Figure	Page
2.1: Visual Depiction of a DevOps Pipeline [7]	8
3.1 Design Idea 1	13
3.2 Design Idea 2	14
3.3 Design Idea 3	15
3.4 System Flow Chart	16
3.5 Updated Design Idea 3	17
3.6 Updated Design Idea 2	18
4.1 High-level Overview of the System Design	21
4.2 Detailed Design of CARLA Communication Interfaces [11]	22
4.3 Detailed Design of Attention Monitoring System [1]	23
4.4 Detailed Design of Communication Between CARLA and Cluster Unit [11]	23
4.5 Detailed Design of Communication Between GitHub Webhook and Jenkins Server	24
4.6 Detailed Design of Framework Web UI	24
5.1 Research Portion of Semester One Project Plan	26
5.2 Development Portion of Semester One Project Plan	26
5.3 Report Portion of Semester One Project Plan	26
5.4 Integration Portion of Semester Two Project Plan	27
5.5 Refinement Portion of Semester Two Project Plan	27
5.6 Final Testing Portion of Semester Two Project Plan	27
5.7 Presentation Portion of Semester Two Project Plan	28

1. Executive Summary

The Continuous Integration Framework for Automotive Software Development is a project that aims to develop an integration framework for the specific purpose of testing automotive software applications. In the current automotive software development industry, there is a clear lack of testing environments for the apps developed for the infotainment units found within modern cars. This project aims to fill this gap in the industry by building on top of the CARLA Hardware In the Loop (CHIL) system and leveraging the hardware components and configurations of these components with the CARLA simulator done by the previous team to represent a digital twin of a car system. This digital twin will then be used by our framework to do integration tests with, to represent testing specific functions of a vehicle. End-to-end, our framework is built upon technologies such as Python, Jenkins, Git, Elastic Search, Logstash, and Kibana. Our team added an Android based infotainment unit to the existing test bench, established a communication bridge, and wrote an extensive integration testing test suite for the CHIL system bench. These tests are run on a Jenkins environment configured by our team, which communicates with Git to execute the test anytime a change is made. Finally, this Jenkins environment is connected to DataDog, an ELK-stack service, to display log data about current and previous builds. The framework was run with various apps, both malicious and safe to test functionality of the framework. In addition to the development of the framework, our team extended the CHIL system by adding a gear shifter hardware component. Conclusively, through the combined efforts of all team members, our team was able to develop a proof of concept integration framework which effectively tests the digital twin of the car, hardware components in the CHIL system and the infotainment unit our team added.

2. Report 1

2.1 Problem Identification

In recent years, the number of technological advancements has grown exponentially in every industry, and the automotive industry is no different. The classic infotainment found in cars has changed from being a simpler system controlled by knobs and dials to a more complex set of software and hardware involving touch screens and hands-free controls. This increase in capabilities means an increase in complexity for testing anything involved with these infotainment systems, including software applications developed for them. Automotive software developers cannot push new code to these infotainment systems without testing rigorously as they cannot test how the app will integrate with the rest of the system. This is crucial to securely develop new software for these infotainment systems on time. To address this issue, our team will be developing a continuous integration framework specifically designed for automotive software development. Using the set of DevOps practices, developers will be able to push new applications and new updates to older applications through our pipeline framework which will

allow these developers to automatically complete integration testing with the rest of the car's system.

2.2 Background and Research Review

With new vehicles coming out every year, each with its own new and improved infotainment systems which fully interact with all aspects of the car, it is important that these systems are not causing any functional defects within the rest of the car. In today's market, we have a few platforms that major car companies use in their infotainment system, these include Android, QNX, Linux, windows and the newest of them all which is Google's Android Automotive OS (AAOS). Big car brands like Honda, Volvo, GMC, Polestar and more will be releasing their 2022 vehicles with AAOS [1,2]. For this reason, focusing on testing using this latest release of AAOS is promising as it deems to be the future of all infotainment systems.

A part of our research involved determining how we would set up an infotainment system with the AAOS. Currently, Google does not distribute images of AAOS, however, an emulator for this OS is provided through Android Studio [3]. Specific car manufacturers, such as Volvo, also provide their versions of images to use with this emulator [3]. In addition to this emulator, one of our team's main focuses of research was to determine if AAOS could be put onto a physical infotainment system since Google does not distribute official images of the OS. Through our research, we were able to find distributions of AAOS for specific devices through an open-source android distribution called LineageOS [4]. Using this android distribution, an AAOS port was made for a Samsung S5E Tablet, which could be used as a touch screen infotainment system [4]. Snapp Automotive, the company that was able to create this port also provides images for other devices including a Raspberry Pi 4B, which we could use to act as a physical infotainment system [5]. Snapp Automotive provides step-by-step instructions on you to set up and build AAOS onto the Raspberry Pi [5].

In addition to all the research about infotainment systems and their operating systems, our team also focused our research on the DevOps set of practices. Understanding continuous development and continuous integration (CI/CD) pipelines are important since the main focus of our project is the development of a CI framework specifically for automotive software development. DevOps is described as the "collaboration" between the development and operations during the lifecycle of a project, which begins with planning and ends with product support [6]. Below is a clear depiction that we used to understand the CI/CD process for ourselves. It shows how developers contribute to a build that gets run through a CI/CD pipeline and test for any failures.

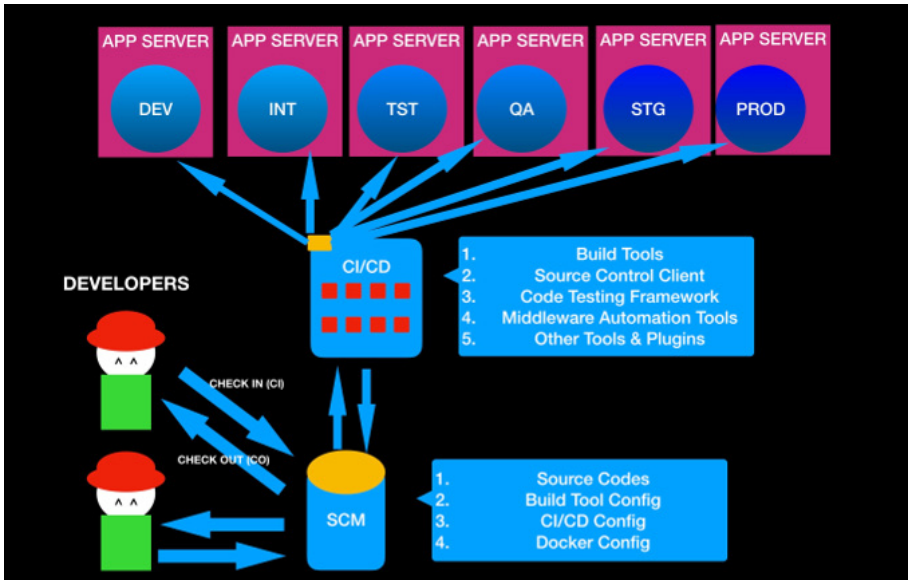

Figure 2.1: Visual Depiction of a DevOps Pipeline [7]

2.3 Design Process

For this capstone project, our team will be following the agile set of methodologies through iterative development, sprints and scrums. Agile development is incremental however in addition to this, it is a set of practices and not just how development occurs [8]. Agile will allow our team to develop small increments of code that can be reviewed by the stakeholders, or in this case, our team's faculty advisor. In addition to code being reviewed, the stakeholders can continually review requirements to make sure the projects are up to standard [8]. In scrum-based agile, development is done in fixed-length iterations called sprints [8]. Our team has decided to keep these sprints at a 1-2 week length.

With this development process, our team gathered requirements and did planning on physical hardware required as well as the software used to combine the different pieces of the system. With all this research, our team has split up the project into 3 different parts: design and development of the DevOps (CI/CD) framework, enabling communication between the different parts of the system such as Carla, the infotainment system, and the framework, and the testing of the framework. We will split our team into groups to work on these phases of the project, however since everything is closely related, we will all work closely together as well. For preliminary planning, Jayash, Taha and Rushi will work on developing the framework and Lefrancois and Umar will work on allowing communication between the parts of the system, with each group being flexible with the members. For example, if during one sprint the smaller

group requires extra manpower, one member will swap over for the duration of the sprint. Below, we have outlined the build of materials (BOM) we have come up with thus far for this project.

Product	Quantity	Price (CAD)
Android Infotainment System	1	~$450
Power supply	1	~$100

Table 2.1: Build Of Materials (BOM)

For Quality assurance, our team will tackle a testing phase when more development on the rest of the system and framework continues. With this, we will be able to test the communication between the system and make sure everything is interacting as intended. Every separate part of the system including all the information our framework will require from Carla as well as the infotainment system will be tested separately. Additionally, some members will code an app that we will put through the pipeline to test our DevOps framework. This app will be a simple Android app for the infotainment system, which our framework will run integration tests on to test the main purpose of our framework.

Our team will structure itself in a fairly unstructured way, with no particular person acting as a “team leader”, but instead each individual will manage their own time and work, which will be presented by each team member at every meeting. To make sure we stay on track for every meeting, we will designate a different meeting manager every time who will make sure everything that needs to be discussed is discussed and that everything stays on topic and on time. Without a team leader coordinating every task and managing everyone’s work, we will use a KanBan board to track activities and make sure everything is moving as intended. Our team has decided to use Trello Board, which is an electronic KanBan board tool in which each member will create and manage their tasks [9]. Along with this, if the stakeholders have requirement changes, new tasks to accommodate these tasks can be made or older tasks can be updated so that everything continues to stay on track with changes. Trello board will make management easier as it will give us a physical depiction of where the project is and how it is moving, which we expect to greatly increase our workflow [9].

2.4 Scenarios and Use Cases

Use Case 1:

Adding apps to a cars’ infotainment system can cause the car to behave in unwanted ways, our framework is going to check the app and make sure it does not interfere with the car negatively.

UC1-Scenario 1: Bob, a software engineer, decides to spend his free time making an app to upload to the infotainment system in his brand new car to track how many times he stopped at Tim Hortons on his way to work. Bob then passes his app through our framework and finds out that his app failed to pass all the tests because a section of the code causes the car blinkers to go off.

UC1-Scenario 2: Bob reviews his code and fixes the bug and passes the new code to our framework and it passes all tests. Bob then uploads the app to his infotainment system knowing that it is not going to break his new car.

Use Case 2:

Car manufactures often send updates to their infotainment system to optimize it and add new features, our framework will double-check if the update has any negative impact on the system.

UC2-Scenario 1: Volkswagen wants to send their new update for their infotainment system, they pass it through our framework and it passes and pushes the update.

UC2-Scenario 1: Volkswagen wants to send their new update for their infotainment system, they pass it through our framework and it fails. Volkswagen then looks at what failed and tries to fix it and then uploads it to our framework to recheck if the update is ready to deploy.

2.5 Stakeholder Requirements and Traceability Matrix

The primary stakeholders of the project are Dr. Azim and the engineering faculty of Ontario Tech University. These are the investors in the project as well as the main force driving the requirements of the project. The secondary stakeholders include the customers that would use the framework such as large car companies like Volvo, Honda etc. and automotive software developers looking to run integration tests on their builds. The following are the stakeholder's requirements:

Requirement 1: Automotive developers must be able to use the framework to run integration testing on their builds.

Requirement 2: Users should have a login to access and monitor their build

Requirement 3: The framework should run the entire test suite through the entire build each time a change has been made, even if an early test fails.

Requirement 4: Results of the failed test cases should be displayed through a UI for the client to fix.

Requirement 5: The system must have a physical infotainment system that must be able to directly communicate with the rest of the system for integration testing using the framework.

Requirement 6: The project must be within a budget of $1000.

A traceability matrix is constructed to demonstrate the relationship between requirements and other artifacts, in our case it is based on the priority of each requirement mentioned by the stakeholder and will demonstrate how each of these requirements will be fulfilled. Using a traceability matrix will allow us to focus on the highest priority requirement first, and allow us to keep track of the overall progress of each stakeholder requirement.

Requirement ID #	Requirement Description	Priority Status	Fulfillment
1	Developers can upload their build onto our integration framework for testing	High	Developers will be given access to our continuous integration framework, allowing them to interact with it to test their builds
2	Users can register and then log in to access the progress of their build	Medium	A portal with a UI will be created allowing clients to register and then log in, which will then allow them to access the framework and test on the framework
3	The framework will run the entire test suite for every change made to the build, even if there are tests that fail	High	The framework will be set up to continue running regardless of failure while storing those failures for the user to see later
4	Users will be able to see all the tests that failed after the entire build has been tested	High	Developers will be able to access our framework through a UI, which will allow them to see all the failures that show up after the entire suite has been tested

5	The system must have a physical infotainment system that must be able to directly communicate with the rest of the system for integration testing using the framework.	Medium	A physical infotainment unit will be configured through an android debug bridge to directly interact with the entire system.

Table 2.2: Traceability Matrix

2.6 Definition of Acceptance Tests

Acceptance testing is a crucial part of project development as its purpose is to check if the stakeholder's needs are satisfied based on the developed solution.

Acceptance Test 1: The framework runs 100% of the test suite every time the user initiates their build to the framework.

Acceptance Test 2: The framework validates or invalidates the user's login information currently 97% of the time.

Acceptance Test 3: The framework still runs 100% of the remaining tests after a failed test.

Acceptance Test 4: The framework displays a rundown of the test suite every time it is run.

Acceptance Test 5: The framework can communicate with the rest of the system correctly 95% of the time.

Acceptance Test 6: The project is completed under $1000 for all hardware and software requirements.

3. Report 2

3.1 Concept Generation and Analysis

Concept Analysis

For the purpose of this project, our team has decided to go with a software approach with physical components involved. These physical components will represent the parts of a car so that our framework can be integrated with a system representing a car. Currently, our system will use CARLA as our simulator for the mechanical parts of the car system, such as the car engine, steering wheel, acceleration/brake pedals etc. In addition to this, our team will use a physical

Android-based infotainment system to represent the infotainment systems found within modern cars.

Due to the nature of this project, an all hardware approach would not be feasible at all, as the main focus of this project is to develop a DevOps framework. This is why we went for a hybrid software-hardware approach to accomplish the main focus of the project but also have a system that represents a twin of a car system as closely as possible. Currently, the goal we are working towards is with CARLA handling all mechanical parts of the care system, but our team's ultimate goal is to incorporate the previous team's Carla Hardware in the Loop (CHIL) system to handle more realistic car inputs with more physical components. These extra components include steering wheeling, acceleration pedals and a physical headboard with a speedometer. Below, design ideas that our team has considered throughout the development of this system will be covered, and explained with diagrams.

Design Idea 1

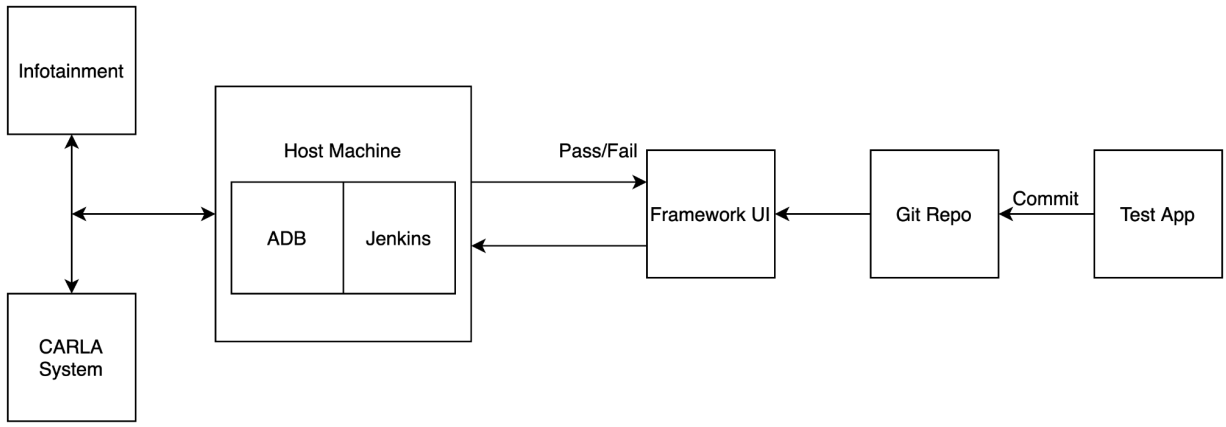

Figure 3.1 Design Idea 1

This was the initial design our team considered and served as the main concept diagram for much of the early development of this project. In this proposed design, an actor, in this case being an automotive app developer, would push their app to a git repository which will contain tests to run. The Jenkins UI will allow the developer to start the build and test runs from this repository and will output results to a screen connected to the host machine. This host machine will communicate with the infotainment and CARLA system through the use of ADB and the infotainment system and CARLA system will communicate with each other. Through experimentation and development, our team deemed this design infeasible due to communication issues between the CARLA system and the infotainment system. A middleman is required for any communication between these two and for the test app to have any interaction with the system to be tested, it must be installed temporarily on the infotainment. With these conclusions in mind, our team generates the next design idea.

Design Idea 2

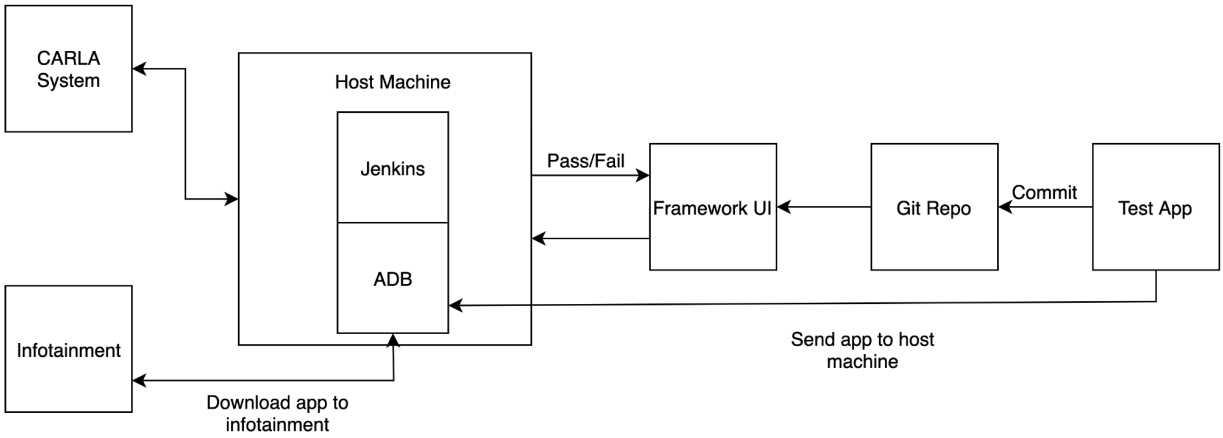

Figure 3.2 Design Idea 2

This design idea incorporates all the changes that needed to be made to make design idea 1 more feasible. This design serves as the design our team is currently working towards and serves as our goal for this semester. This design works the same as design idea 1, with all communication issues solved. The test app will be pushed to the infotainment system through the Android Debug Bridge (ADB) connection on the host machine, which will serve as the basis for all communication between this machine and the infotainment system. This host machine will also serve as the middle man for the communication between the CARLA system and the infotainment system. CARLA will communicate with it through the use of various python scripts, and the host machine will transport any information required to the infotainment through the ADB if required. Using this design, any tests between the app and the twin of the car system can be done. Below, an extension of this design is explained that our team aims to make our goal for the next semester, incorporating extra hardware for a more realistic test bench.

Design Idea 3

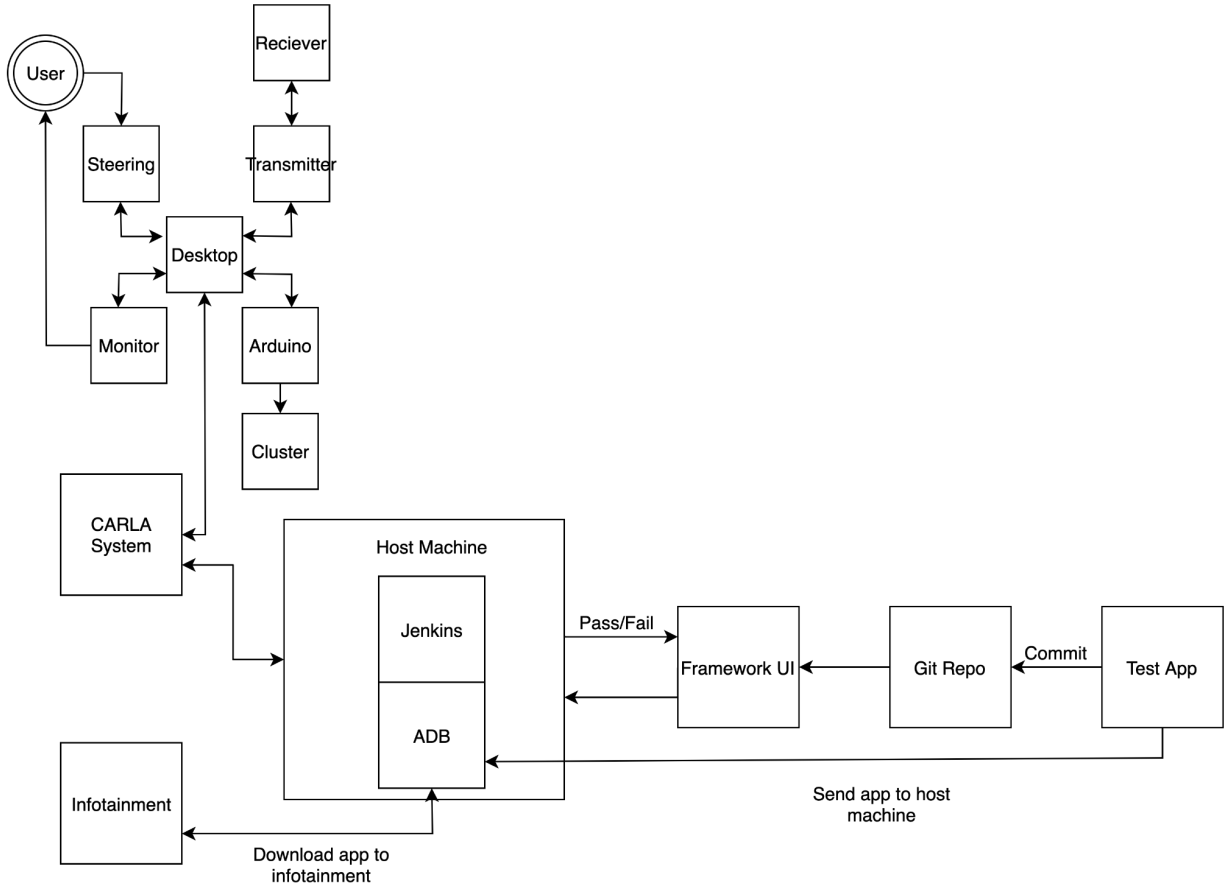

Figure 3.3 Design Idea 3

This design concept presents the exact same system as design idea 2 with an extra extension for more car hardware. This hardware includes all hardware included in the previous group's CHIL system, a steering wheel, an output display for the reverse camera of the car, a physical headboard with a speedometer, an infotainment unit and all Arduino controllers for these components. The purpose of this design is to serve as the ultimate end goal for our team and to provide a more realistic test bench alongside our framework. The extra hardware will allow for this as it will represent the mechanic system of the car more realistically than a pure CARLA system.

Communication Design

There are various components to the complete system and as such require different interfaces and scripts to deal with communication between these components. To deal with any communication with the infotainment system, the use of the Android Debug Bridge (ADB) is required, as it is an Android based machine. Any communication with CARLA will be done

through Python and APIs provided by CARLA. Communication with the Jenkins server and the rest of the pipeline will be done through a Python script written by the team, that will allow for authentication to the server, installation of the app to the infotainment and control of the testing jobs. Below is a flow chart describing all the different communication interfaces and APIs working together to allow for communication between all components. User interaction will happen through a python script written by the team, which will handle all inputs from different interfaces such as ADB and CARLA's API to pull values from a client.

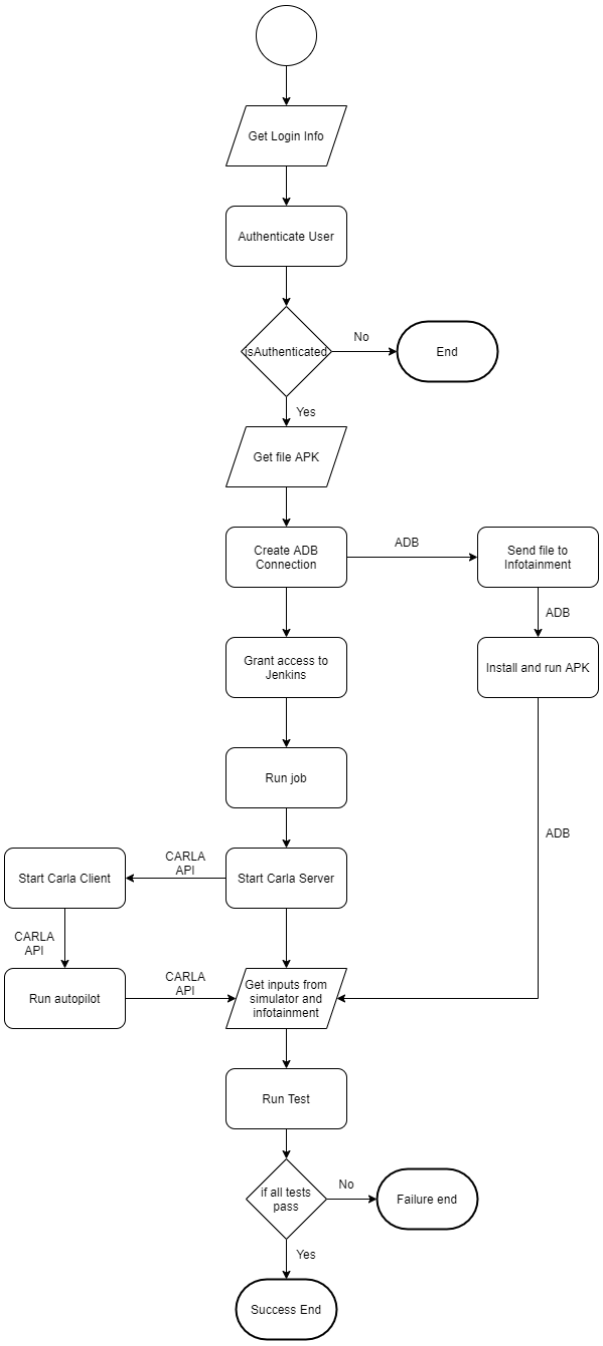

Figure 3.4 System Flow Chart

3.2 Conceptual System Design

After carefully looking at each of the possible design concepts, we decided on concept 3 to pursue as the final design. This design is the most thorough and involves the most comprehensive architecture. This architecture allows for the most intricate testing, as it is able to extend the framework further by integrating testing through physical components.

Concept 3 is the most involved concept but allows for the most rigorous testing of an application within the pipeline. It enables the application to be tested within physical components that mimic real-time car systems. Some of these components include, for example, the cluster and the backup camera. This entire extension has been provided by the previous Capstone group's CHIL system which we are building on top of. The host machine within the pipeline is able to connect with the previous group's CARLA client and check for values from the simulation to verify that the car is correctly operating as the application is running. These values are passed through to the host machine and compared to pre-determined constants which are compared to the passed values. In the end, the user is notified of any failures in either aspect of the build which involves the infotainment or the CHIL system. This lead to our decision to use this design concept.

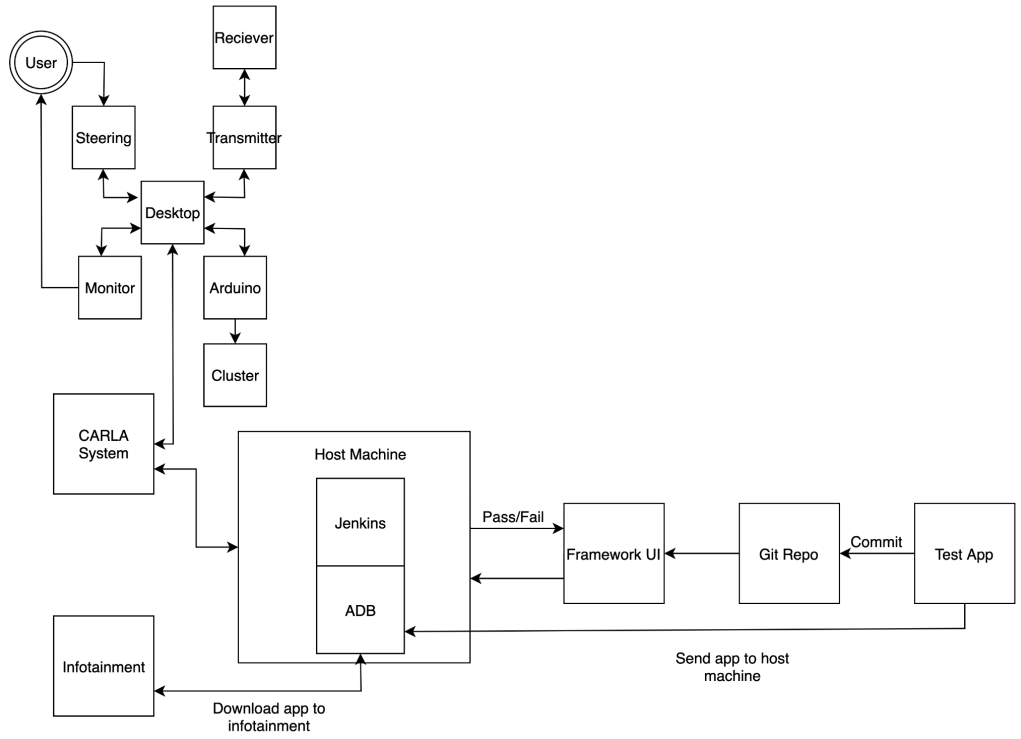

Figure 3.5 Updated Design Idea 3

Although concept 3 is the most thorough of all the concepts, the most straightforward and bare minimum integration testing would be possible with concept 2. This concept is the current goal for the fall semester as it will still allow for testing of the entire application within the framework but without the physical components. The majority of the tests will still only be able to work through the infotainment system and very few will run on CARLA. The infotainment side of this architecture will allow for full-scale testing of the app with the system. On the CARLA side, the only aspects that will be able to be tested are the software aspects from CARLA. Components such as the gauge cluster and backup camera cannot be fully tested because these are hardware components. Without testing the application's effects on a physical system, it cannot achieve complete integration testing. This led to our decision to not use this design concept.

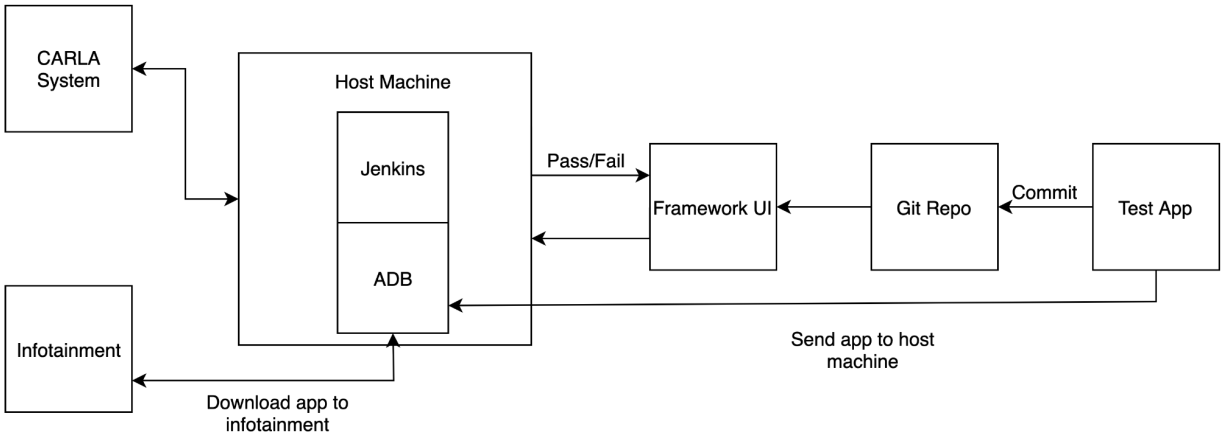

Figure 3.6 Updated Design Idea 2

The entire pipeline involves many components working together in unison. It starts off with the app being committed in the Git repository. Once this is done, it triggers the framework UI. This framework is built with Python and tasked with starting the host machine once the user logs in. Once the user is logged in, the pipeline continues and the Jenkins portal will allow for the user to select the test suite that they wish to perform on the app that is currently queued within the pipeline. Once this is selected, the app is downloaded to the infotainment system via the Android Debug Bridge (ADB). The tests are then conducted by Jenkins and passed through to the infotainment system via ADB and the results are returned and processed by Jenkins to determine the pass and fail cases. When results are received, they are passed to the framework UI to inform the user of the results. With this, a full log of the tests will be available to the user to view the exact passes and fails in detail.

3.3 Definition of Integration Tests

Integration testing will achieve testing of all the independent modules working together successfully. The purpose of this is to demonstrate the compliance of the system as a whole

against the functional requirements[10]. In our case, we will be utilizing a hybrid integration testing method which is considered to be a combination of both bottom-up and top-down integration methods [10].

The entirety of the system can be split into 3 main high-level modules. The first module is the DevOps pipeline, created with technologies like Git and Jenkins. The second main module is the Android-based infotainment unit. The third main module is the entire car system, represented by the CARLA simulator currently, and ultimately the CHIL system. Each of these main modules will be unit tested separately prior to integration testing, in order to guarantee each module is functional individually. To apply integration testing to these modules, the team will write test cases in order to test any communication between the Jenkins job and the infotainment. These tests will involve testing if the ADB connection is established and if information can be passed over this connection. Additionally, to test the CARLA module, tests will be written to test if a CARLA client can be established and if values can be pulled to the host machine from CARLA. Later, our team will review the previous team's integration tests written for the CHIL system and refactor where necessary. Below each module's integration test that our team has currently planned will be outlined

Test Case ID	Test Case Objective	Test Case Description
1	Test establishment of CARLA simulator server/client	Testing the CARLA system making sure that both the client and server run successfully
2	Test communication between CARLA and host machine	Testing if CARLA can output values to the host machine using the use of python scripts
3	Test establishment of the ADB connection over USB	Testing if an ADB connection can be established between the host machine and the infotainment unit
4	Test installation of the app onto infotainment from python script	Test if the app can be installed over the ADB connection by passing an APK through a python script
5	Test if information can be passed over the ADB	Test if the infotainment unit can pass any information to

	connection	the host machine

Table 3.1 Test Cases

3.4 Estimated Cost

The project estimated cost is derived in 2 forms, one being the material cost needed for the development and testing for the project. The second is an estimated workforce cost for the total time spent on the development.

Cost of Materials:

Product	Quantity	Price (CAD)
Android Infotainment System	1	$447.47
Power supply	1	$110.71
Usb-Usb connector	1	$11.29

Table 3.2: Updated Build Of Materials (BOM)

Currently, for this semester we are over budget by $69.47 but with next semester's budget, we will be covered as we no longer anticipate any more major expenses and predict to have a very small cost overflow, if any moving forward with development.

Workforce Cost:
Number of days you plan to work on the project: 96 days
Number of workers: 5
Hours for each workers: 2 hours
Workforce total calculation: 96 x 5 x 2
Workforce = 960 hours of work

Percentage of this time on each of the project tasks:
Design: 30%
Implementation: 40%
Testing: 20%
Documents: 10%

4. Revised design report

4.1 Detailed Design

Our team has previously defined and designed a final conceptual design for the overall system. This design encompasses the existing CHIL system created by another team, as well as the CI/CD integration framework created by our team. This design was reviewed and compared to the physical system after interaction within the lab and a revised diagram was created.

After physically interacting with the CARLA system in the lab, our team was able to confirm that this side of the overall system consists of the CARLA client/server, a steering wheel and pedal kit, a webcam as part of the attention monitoring system, a BMW cluster and Arduinos to interface communication to the cluster, and an Android-based infotainment unit. After this initial review, our team also added a traditional gear shifter to the system.

The CI/CD pipeline side of the project consists of the main Jenkins automated testing server, Android Debug Bridge (ADB) connections between the host machine and the infotainment unit, webhooks between GitHub and the Jenkins server and a small web application to serve as a frontend UI to the Jenkins server. After defining all these components of the overall system, our team created the following high-level diagram to incorporate all the revised changes and additions.

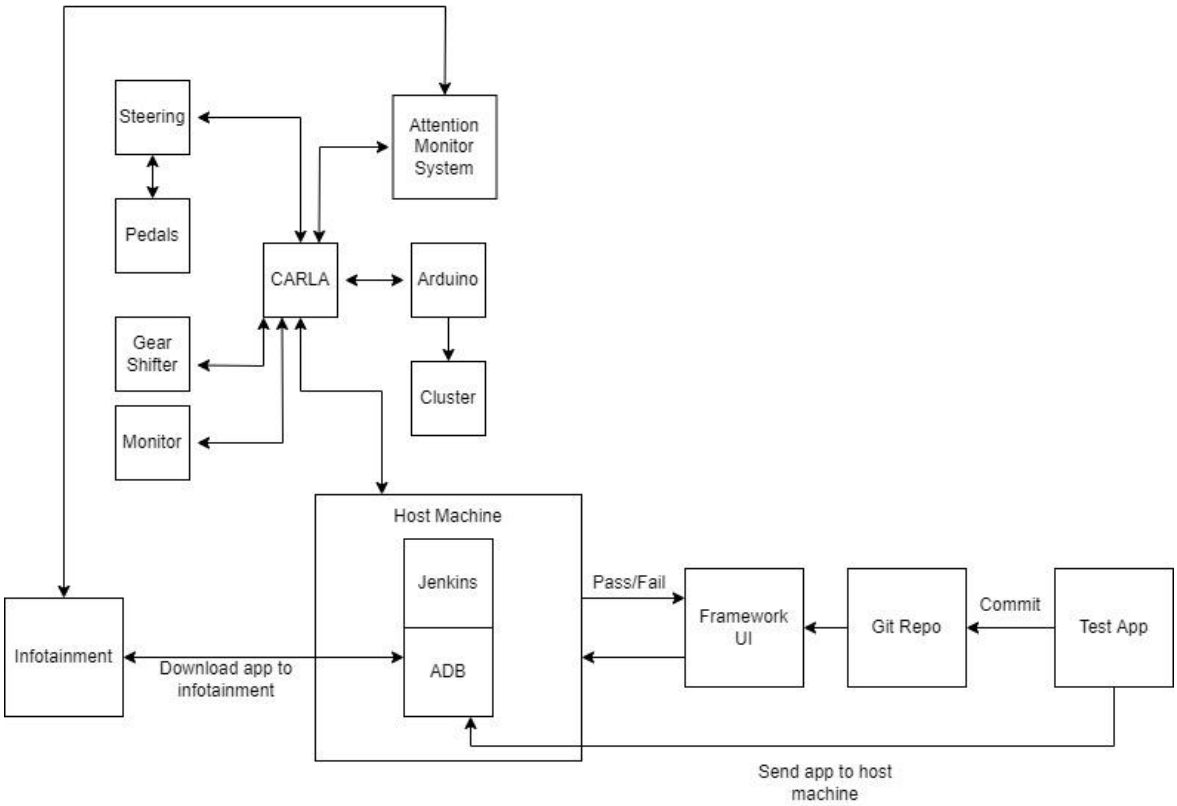

Figure 4.1 High-level Overview of the System Design

This high-level diagram represents the overall system architecture and high-level interaction between components. Each group of components is detailed further and broken down into specifics about the interfaces used for communication and the modules that make up the components.

Beginning with the CHIL system side of the overall system, there are many components within this part of the project that can be detailed further. To start, the CARLA client can be detailed to showcase the computation involved in the object detection, lane assist, and steering controls within the simulation. This was specifically detailed by the CHIL system team and is shown in the following diagram.

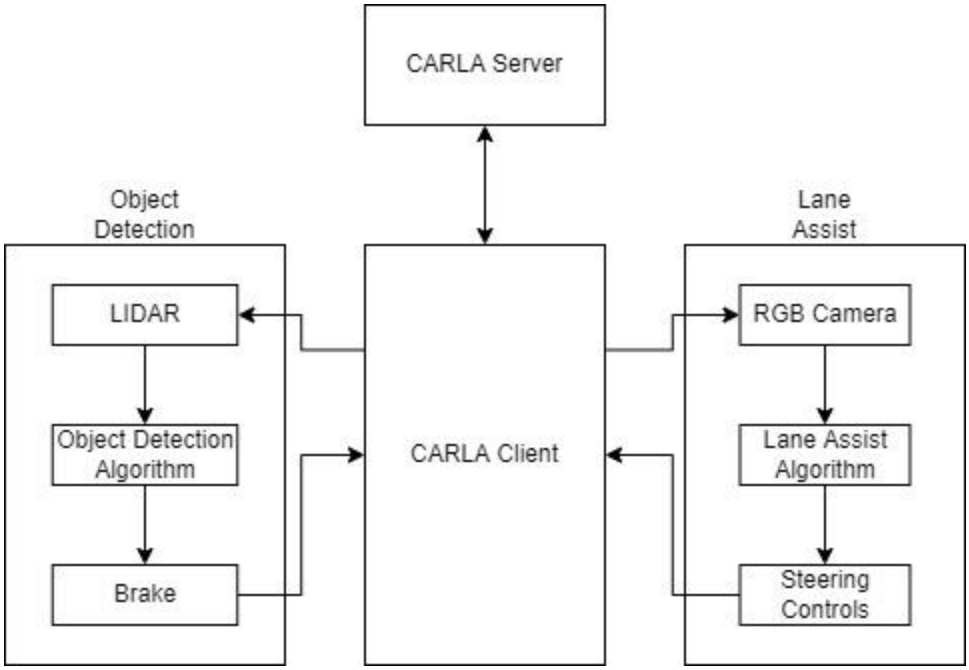

Figure 4.2 Detailed Design of CARLA Communication Interfaces [11]

This diagram represents the internal computation going on while CARLA is processing the lane assist and object detection using built-in cameras, LIDAR technology, and algorithms. The next components of the CHIL system that can be detailed are the attention monitoring system as well as the BMW cluster. Each of these components is made up of smaller modules that serve a specific purpose to allow for full functionality of the components. These components are detailed in each of the respective diagrams below.

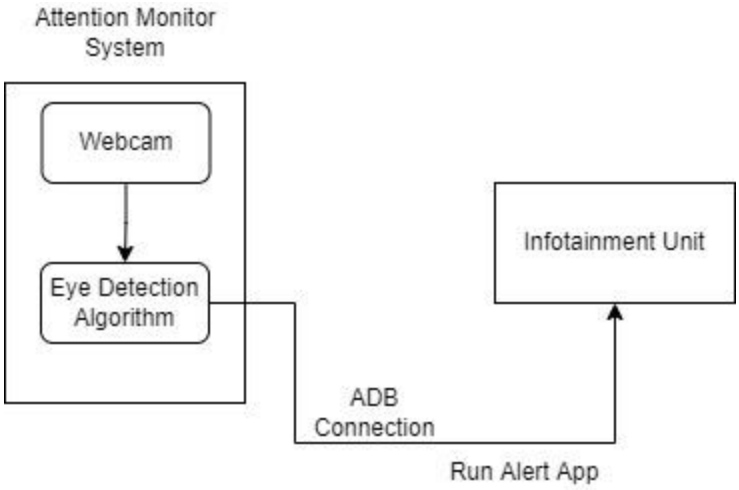

Figure 4.3 Detailed Design of Attention Monitoring System [1]

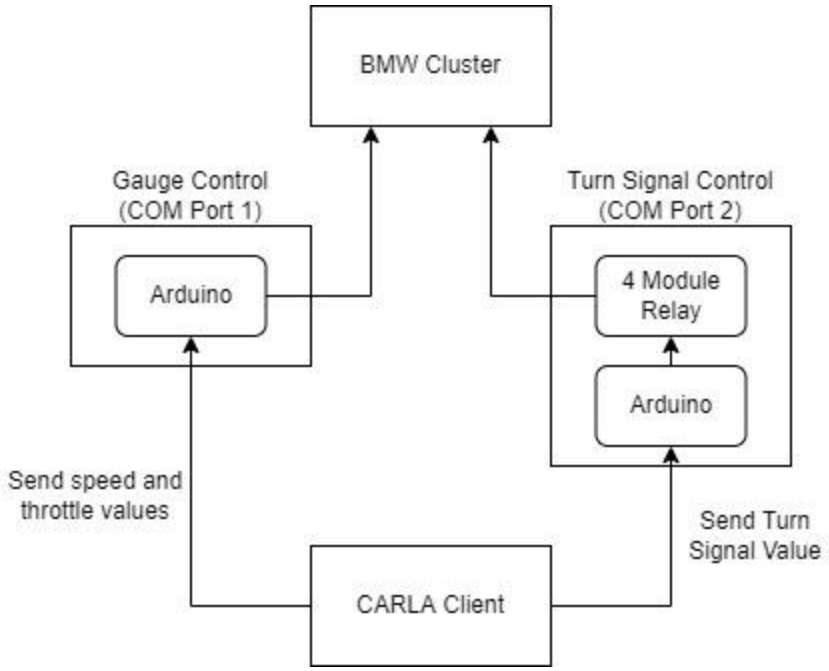

Figure 4.4 Detailed Design of Communication Between CARLA and Cluster Unit [11]

These diagrams detail exactly how communication between the attention monitor system and the infotainment unit, as well as the cluster system and CARLA, takes place. The attention monitor system consists of a webcam and an algorithm developed by the CHIL system team, which will detect when the driver is not paying attention, and then will launch an application on the infotainment unit over an ADB connection. The cluster system consists of 2 Arduinos which parse different values from the CARLA client. The first is used to modulate speed and throttle values, to allow for gauge (speed and RPM) control. The second uses a 4 module relay to control the turn signal indicators, based on the values sent from the client.

With the entire CHIL system side of the overall system detailed, a few components of the CI/CD framework side can be detailed further. Beginning with the Jenkins server and the communication interface between the Github webhooks and this server. As it is now, the server is hosted locally, and without other tools, cannot communicate with Github. To solve this problem, our team uses Ngrok to allow for communication with the webhooks. Ngrok is a tunnelling service, which can be used to expose locally hosted servers to the web for various purposes [12]. The following diagram details this connection and communication in its entirety.

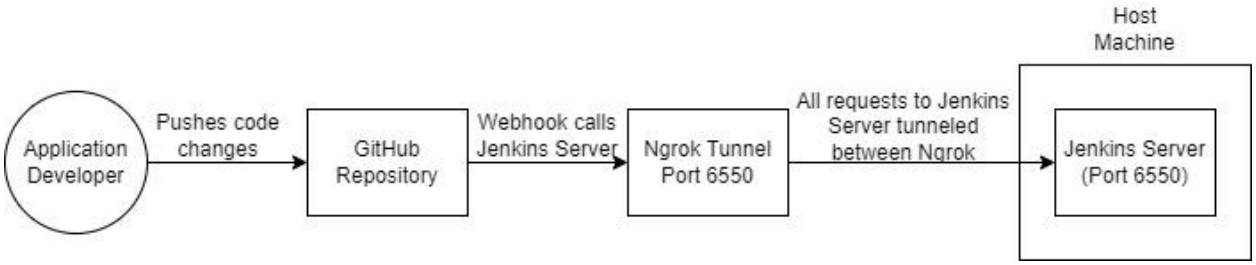

Figure 4.5 Detailed Design of Communication Between GitHub Webhook and Jenkins Server

The final part of the system that can be detailed further is the front-end UI. This part of our system is where the user will have the ability to see the current status of the build as well as the passed/failed test cases. The UI will be a single-page web application with a username and password input at the top left of the page and the center of the page will display all the data of the build and test cases. All the data displayed will be pulled from the Jenkins API using Flask as a backend for the python code. The Framework UI will be built with the react javascript framework. This UI web application is detailed entirely in the diagram below.

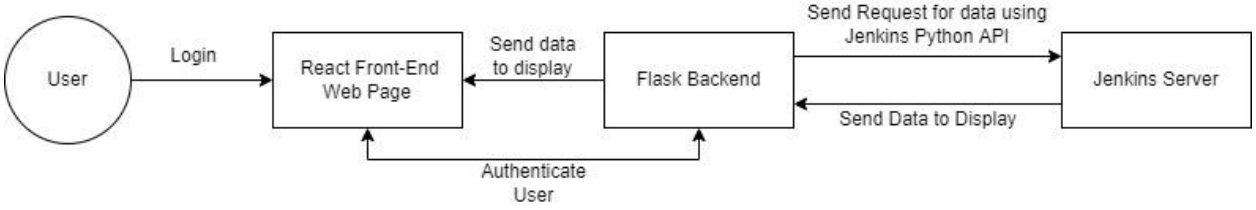

Figure 4.6 Detailed Design of Framework Web UI

4.2 Unit Testing

Before continuing with any sort of integration tests at this stage of the project, it was decided that it would be a good idea for every component of the system to be individually unit tested to confirm the functionality of every single part of the system before further work is done. This was done in order to avoid any sort of backtracking as a result of incorrectly functioning components. The key components are the CHIL system, infotainment, Jenkins host machine, and front-end UI.

It was very crucial to unit test the CHIL system, as it was a system that had not been used in some time and could have had issues in working as intended. There are many components of the system so it was a priority to start on unit testing it right away. Due to the lack of documentation provided by the previous group, some time was required to go through the CHIL system computer to understand how all the components work together. This involved going through the project files and combing through the code to fully understand how everything works together. After the system was thoroughly understood, we were able to create a checklist of all components that needed to be verified. When the simulator was started, we were able to go through all the features one by one in order to verify their functionality.

For the unit testing of the infotainment system, all its basic features were verified to be working. These were features such as navigation, wifi, music player, and Bluetooth. The infotainment was turned on and all of these features were verified to be working.

The Jenkins host machine contains the pipeline configurations for the system and is in charge of executing the various test cases for the system. The Jenkins system was unit tested by committing a change to a Github repository and verifying that it has triggered a build within Jenkins. The build log was inspected afterwards to verify that all of the test cases were executing and that there were no present dependency errors.

The front-end UI that is currently being developed by our team will require unit testing to see if it is displaying information correctly to the user. It uses the Jenkins API to make calls to the configuration for data that it can clearly display to the user about its build statistics. The unit testing for this will require unit tests to be written for the React/Flask scripts to ensure that the methods within the program are functioning exactly as intended.

5. Updated Project Plan

Figures 5.1-5.7 below represent the plan that our group has followed for the first semester of our project and will continue to follow till the end. This was initially created and estimated to be changed around 1-2 days, however, even following some minor inconveniences we were able to follow through and keep on track with the timeline we had created to keep us on track. The tasks were broken down into three separate categories which are research, development, and the report. The research portion had a majority of it completed within the first month of planning. The development portion is the longest portion of the first-semester timeline for this project. It consists of the key development periods that will enable us to build our initial prototype. The plan for this prototype is to be able to send input and get an output throughout the entire development pipeline.

First Semester:

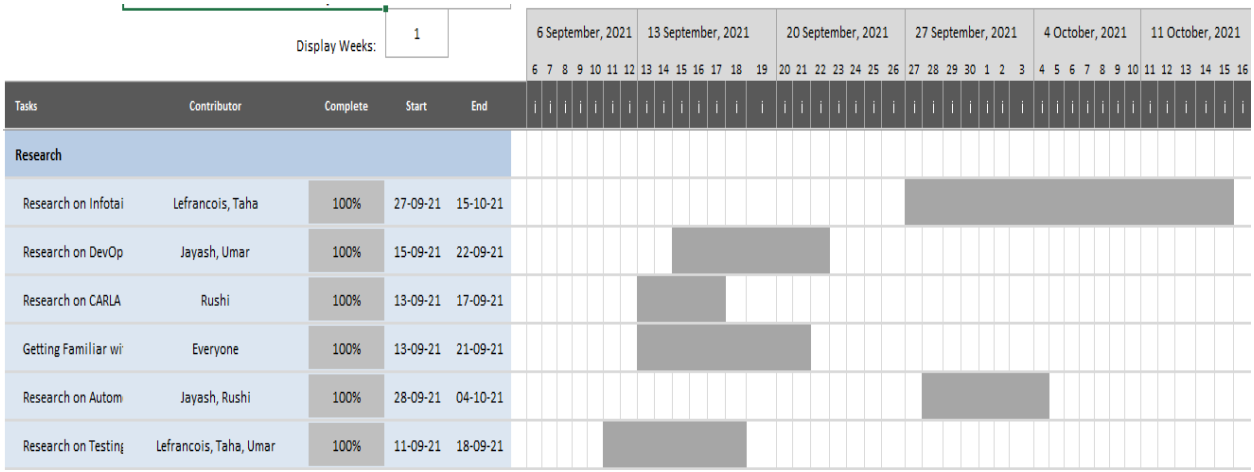

Figure 5.1 Research Portion of Semester One Project Plan

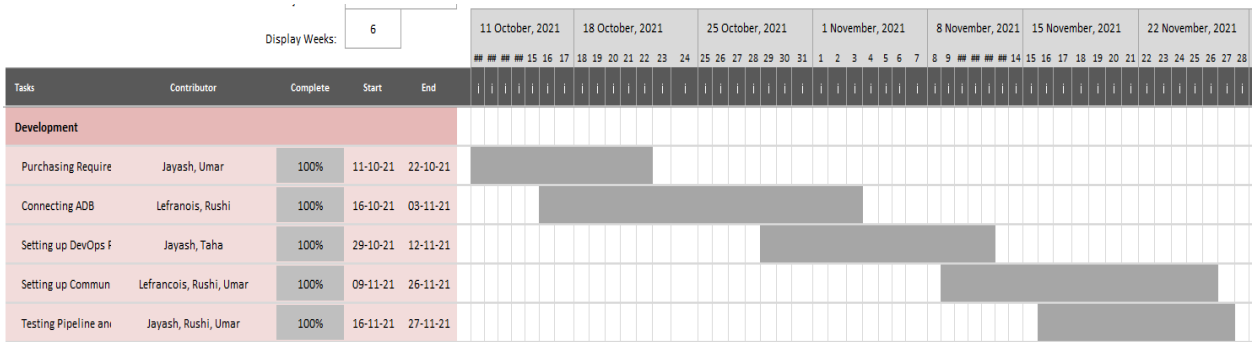

Figure 5.2 Development Portion of Semester One Project Plan

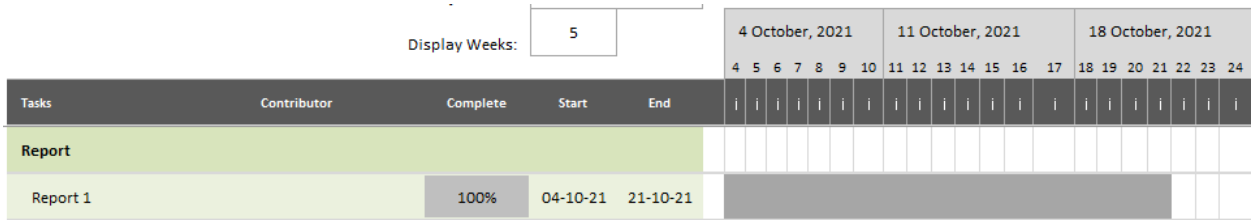

Figure 5.3 Report Portion of Semester One Project Plan

Second Semester:

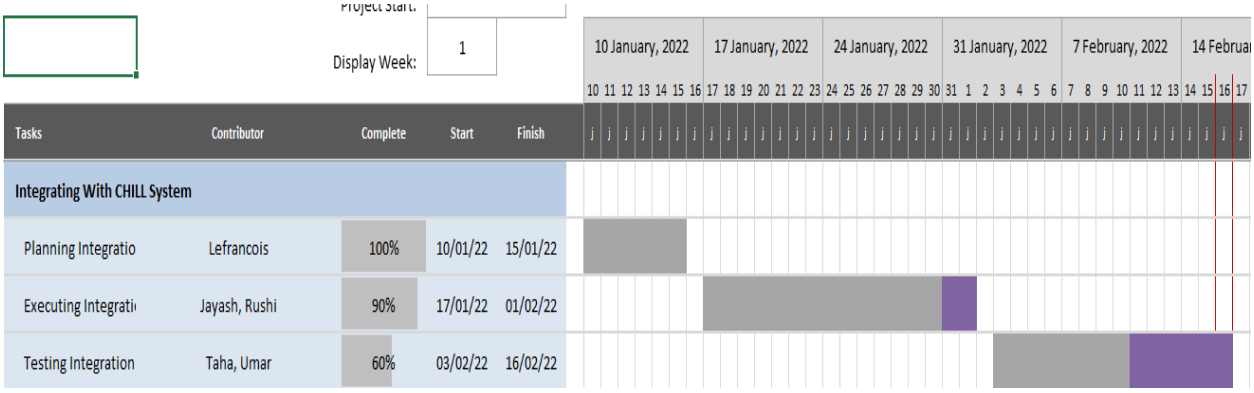

Figure 5.4 Integration Portion of Semester Two Project Plan

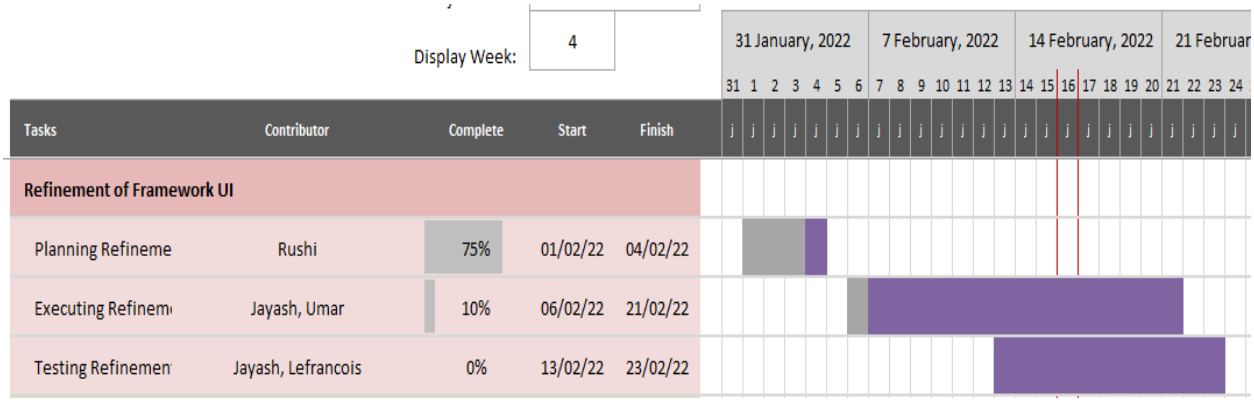

Figure 5.5 Refinement Portion of Semester Two Project Plan

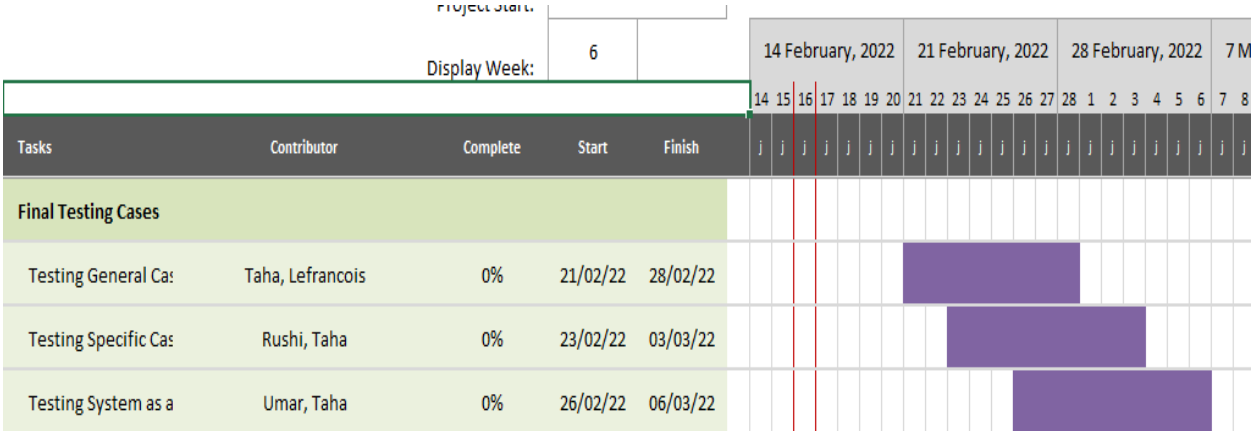

Figure 5.6 Final Testing Portion of Semester Two Project Plan

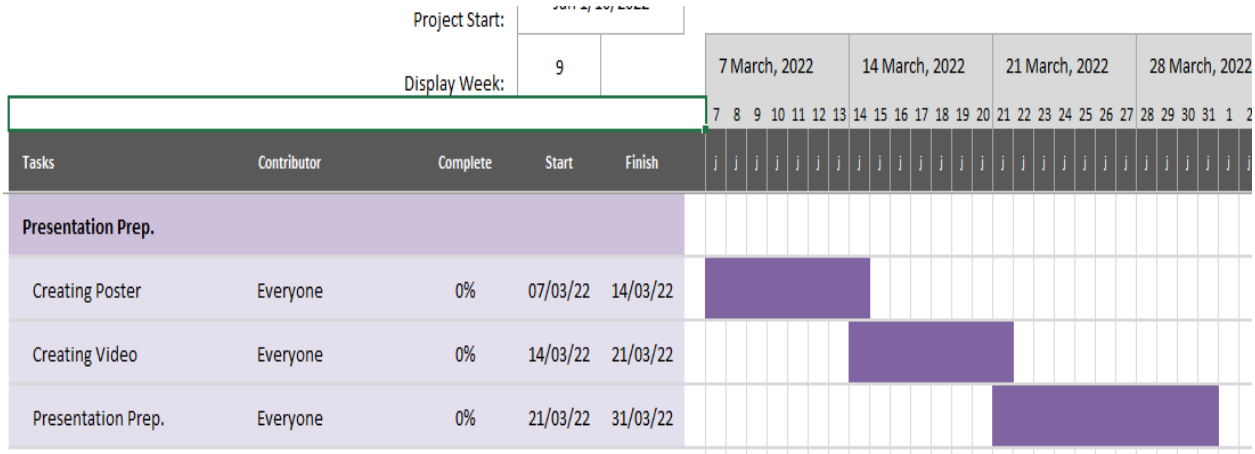

Figure 5.7 Presentation Portion of Semester Two Project Plan

Through the first semester of working on this project, everything went according to plan allowing us to stay on track with our timeline. However, with the start of the second semester and finally beginning the work on the integration process we had to go back and revise our timeline due to some unfortunate circumstances. Executing our integration was pushed longer than initially expected because we were put in a position to have to spend more time with the old CHIL system and inspect their code to see why certain things were not working. In addition to that, we added a physical gear shifter to the system prolonging the time that we had initially prescribed to the integration process.

6. Test Results

6.1 Integration Testing

Integration testing was done in order to verify that all of the components of the framework were correctly working with each other. Due to the nature of our project, our work entailed creating integration tests for developers who are testing their projects against a testing framework. There were some key integration tests that we had to create to test the components between each other. These tests involved the CHIL system, infotainment, Jenkins host machine, and front-end UI.

There were a few special integration tests that had to be written between specific components that are not already being tested within normal usage of the system. One of these integration test cases was between the Jenkins host machine and the front-end UI. These integration tests would rely on using API calls in the front-end UI to pull specific pieces of data from the Jenkins server to confirm that everything is visible on the UI side.

Below are the integration tests that would be used in every build to test the user's application against the framework.

Test Case ID	Test Case Objective	Test Case Description	Completion Statues
1	Test establishment of CARLA simulator server/client	Testing the CARLA system making sure that both the client and server run successfully	Completed
2	Test communication between CARLA and host machine	Testing if CARLA can output values to the host machine using the use of python scripts	Completed
3	Test establishment of the ADB connection over USB	Testing if an ADB connection can be established between the host machine and the infotainment unit	Completed
4	Test installation of the app onto infotainment from python script	Test if the app can be installed over the ADB connection by passing an APK through a python script	Completed
5	Test if information can be passed over the ADB connection	Test if the infotainment unit can pass any information to the host machine	Completed
6	Test if the steering wheel is communicating with Carla	Test if when the physical steering wheel is turned, it turns the vehicle in Carla the same way.	Completed
7	Test if the acceleration pedal is communicating with Carla	Test if when the acceleration pedal is pressed the vehicle in Carla Accelerates when in drive or reverse.	Completed

8	Test if the brake pedal is communicating with Carla	Test if when the brake pedal is pressed the vehicle in Carla slowly comes to a complete stop when in drive or reverse.	Completed
9	Test if the gear shifter is communicating with Carla	Test if when the shifter is in the park position, the vehicle in Carla is parked. Test if when the shifter is in the neutral position, the vehicle in Carla is in neutral position (Vehicle can still move based on road elevation, unlike the parked gear position.) Test if when the shifter is in the drive position, the vehicle in Carla is in drive, allowing you to accelerate and brake until placed into another gear position. Test if when the shifter is in the reverse position, the vehicle in Carla is in reverse, allowing you to accelerate and brake backwards until placed into another gear position.	Completed
10	Testing communication	Testing that when put the gear shifter into	Completed

	between the reverse gear position, the reverse camera and the infotainment system.	the reverse position, it will detect and utilize the reverse camera in Carla and then display the reverse camera to the infotainment system.	
11	Testing facial detection camera with attention system	Testing that when the vehicle is moving the user is paying attention to the road and if not the attention system will activate to tell you to look back at the road.	Completed
12	Testing the Bluetooth component on the infotainment system	Testing that after any changes have been made to the system, the Bluetooth functionality is still operating correctly.	Completed
13	Testing the wifi component on the infotainment system	Testing that after any changes have been made to the system, the wifi functionality is still operating correctly	Completed
14	Testing the radio component on the infotainment system	Testing that after any changes have been made to the system, the radio functionality is still operating correctly	Completed
15	Testing the navigation component on the infotainment system	Testing that after any changes have been made to the system, the GPS receiver's functionality is still operating correctly	Completed

16	Testing the main control unit components on the infotainment system	Testing the CPU and ram in the system to make sure that the CPU and ram usage is at an acceptable threshold after any changes have been made to the system. Testing that the gauge cluster and other components integrated throughout the CANBUS are working correctly after any changes have been made to the system.	Completed

Table 6.1 Updated Test Cases

7. Ethical Considerations

Our group was able to take all ethical considerations into account that were relevant to this project. We specifically took into account the code of ethics for software engineers as outlined by the IEEE Computer Society. One of the main ethical points we took into consideration for this project was the duties we had to our client and employer [13]. In accordance with this, we ensured to keep our stakeholders updated with our progression and to show them the results of the developments that had been made [13]. According to principal 2.03, we also ensured to use all the equipment we were given to borrow as intended for the project and not for any other purposes [13]. Any concerns that we had with the project and its functionality were reported to the stakeholders in accordance with principal 2.07 [13]. We have also done our duty to document and report all the work that we have done to ensure that any future work done with our project can be easily understood, reproduced, and extended further in accordance with 3.08 [13]. These ethical considerations were understood from the start of the project and we ensured to keep them in our minds to ensure that we were performing the duties of the project ethically.

8. Safety Considerations

Initially when beginning with this project, we came to realize that with the little number of hardware components we will be assembling we would need to follow some safety

procedures. Also, to make sure the ample level of safety was being taken into consideration we went over and reviewed the hardware components of the work we were building upon. We went back and double-checked that the existing hardware was not overloading the Arduinos, instrument cluster, and from a high level made sure everything was working as it should and the safety considerations were followed. Moving forward, with any new hardware that we added to the test bench to make sure that the adequate level of safety considerations were met. When adding our infotainment system we had to wire it ourselves, to make sure the safest procedure was followed we followed the manual that was provided from the manufacturer, and used the recommended voltage and current. The wiring to connect the infotainment system was also done using the provided manual and done in the standard way making it as safe as it can possibly be. To make sure safety was the number one priority with our hardware components we made sure to be an industry standard rate power supply unit, and to never leave it running overnight or prolonged periods of time for maximum safety. All in all, we made sure to follow safety protocols that are meant to be followed in a lab, as well as assembling our hardware keeping safety considerations as one of the main priorities. Our project has a non-negative environmental impact as it is something that can be used fully virtually and will not require any physical hardware. Since users only have to push their application build to a github repository for our system to start and run the integration tests.

9. Conclusions

In the final days of the project, we were able to successfully create the product that we had first set out to build. The hours of research and planning that we had initially done at the beginning phases of the project managed to pay off and we were able to build something that we were proud of. A twin system was developed that was capable of testing Android applications for its safe integration with a vehicle system. It is able to check if the application has any effect on the infotainment core systems and it also checks if the application has any effects on the driving systems. It checks for any malicious API calls being made to the system and gives the developer a pass or fail for the situation. This allows the developers of infotainment applications to be able to do full end-to-end testing of their applications and view the results afterwards.

Looking to the future of this project, our team sees lots of extendibility and potential for the project. Currently, our system is communicating using the ADB as the main bridge of communication between all components. As we were nearing the deadline of this project, we were introduced to a proof of concept project, which allowed for CAN-bus communication with CARLA. We believe this is the perfect next step for this project. Introducing a CAN interface to our project, communication will go from being proprietary to systems using android infotainment units, to the very practical use of an interface that modern day vehicles themselves would use. Incorporating the CAN project into this one will take the testing framework to another level.

10. Acknowledgements

We would like to start this by thanking Dr. Akramul Azim for this project and trusting us to carry it out. Development and testing for our bench was thanks to the lab Dr. Azim provided us with all the necessary hardware and software tools that we needed to complete this.

Moving forward, we would like to thank PhD student Md Al Maruf, MASc student Rezwana Mamata, and PhD student Joelma Peixoto. They were in the lab allowing us access, and being there for us whenever we needed.

We would also like to thank Dr. Quasay Mahmoud for the help that came along with the capstone reports. Whenever any clarification was needed he was there. He also worked hard scheduling, reserving rooms, and setting ideal time slots for every group. He made sure everything was communicated with every group and held very useful lectures to make sure groups knew what they were doing.

11. References

[1] M. Gallagher, “Android Automotive Review: The future of car entertainment,” *T3*, 05-Oct-2021. [Online]. Available: https://www.t3.com/reviews/android-automotive.

[2] A. Tarantola, “Google's Android Automotive OS is coming to Honda Cars in 2022,” *TechCrunch*, 23-Sep-2021. [Online]. Available: https://techcrunch.com/2021/09/23/googles-android-automotive-os-is-coming-to-honda-cars-in-2022/.

[3] “What is Android Automotive?   Android Open Source Project,” Android Open Source Project. [Online]. Available: https://source.android.com/devices/automotive/start/what_automotive.

[4] M. Rahman, “Android automotive is made for cars but this developer ported it to a Samsung Tablet,” xda, 29-Jun-2021. [Online]. Available: https://www.xda-developers.com/android-automotive-ported-samsung-tablet/.

[5] A. Sutton, “Android Automotive OS 11 on a raspberry pi,” *Medium*, 27-May-2021. [Online]. Available: https://medium.com/snapp-automotive/android-automotive-os-11-on-a-raspberry-pi-2abaa133f468.

[6] “What is DevOps?,” the agile admin, 12-Jan-2019. [Online]. Available: https://theagileadmin.com/what-is-devops/.

[7] 08 Apr 2019Bryant Son (Correspondent) Feed362up14 comments, “A beginner's guide to building DevOps pipelines with Open source tools,” Opensource.com, 08-Apr-2019. [Online]. Available: https://opensource.com/article/19/4/devops-pipeline.

[8] Atlassian, “What is agile?,” *Atlassian*. [Online]. Available: https://www.atlassian.com/agile.

[9] Trello, “What is Trello?,” *Trello Help*. [Online]. Available: https://help.trello.com/article/708-what-is-trello.

[10] T. Hamilton, “Integration testing: What is, types, top down & bottom up example,” *Guru99*, 08-Oct-2021. [Online]. Available: https://www.guru99.com/integration-testing.html#5.

[11] S. Patel, J. Kaipada, A. Parameswaran, and G. Zakharov, “Design and Development of CARLA Hardware in the loop.”

[12] "ngrok – documentation", *Ngrok.com*, 2022. [Online]. Available: https://ngrok.com/docs.

[13]“Code of ethics: IEEE Computer Society,” *IEEE Computer Society Code of Ethics Comments*. [Online]. Available: .

12. Appendices

A. https://github.com/Tahaa17/capstone-CICD-framework
B. https://github.com/Tahaa17/capstone-app
C. https://github.com/Tahaa17/auto-maintenance-android/tree/main/JULTRAutoMaintenance
D. http://carla.org/
E. https://www.jenkins.io/doc/book/
F. https://www.datadoghq.com/

13. Contribution Matrix

Table 13.1: Contribution matrix

Tasks	**People**				
	Jayash	Taha	Umar	Lefrancois	Rushi
Report Formatting	20%	20%	20%	20%	20%
Revised Design Report	10%	30%	10%	30%	20%
Executive Summary	30%	25%	20%	X	25%
Ethical Considerations	10%	10%	20%	40%	20%
Safety Considerations	X	X	50%	X	50%
Conclusion	40%	40%	X	20%	X
Acknowledgements	20%	20%	20%	20%	20%
References	20%	20%	20%	20%	20%
Appendices					
Contribution Matrix	20%	20%	20%	20%	20%