Team - CC

Cruise Control System

Taylor Niedzielski, Ayden Flynn, Samuel Pitao, Evan Kupsch

Version 0.5

Overview of Document: This document gives the specifications of the cruise control system. It is broken up as follows:

Executive Summary

We are going to create a cruise control system that will be implemented in a car. The cruise control functions as an automatic speed regulator. When turned on, the speed that the driver desires will be maintained. Its purpose is to help make the driver maintain a consistent speed, creating a more luxurious and safer driving experience. The cruise control system will monitor the car's speed, and automatically accelerate or decelerate if the car is slower or faster than the speed set by the user.

Introduction

Purpose of this Document

The aim of this document is to provide succinct documentation for specific requirements for a new cruise control system for automobiles. The system will provide an easy way for users to set and maintain the speed of their car. This document provides the scope, objectives, and goals of the new system. In addition, this document will describe non-functional, functional, and security requirements. This document models the requirements with use cases, constraints, user stories, and product perspective. This document will serve to direct the design and implementation of the system.

Goals

Create a functioning cruise control system that is able to achieve its intended purposes and functions.

Project Scope

- The timeline of the project is from February 5 and will be completed by the week of April 19.
- The cruise control system will consist of software, operating system, and hardware.
- Functioning as a hard real-time mission-critical system, the cruise control will provide constant speed, easy management of the system, and security.
- The cruise control system will take input from sensors that are already in the car (speed, break, logging) and will send output to actuators, mainly the Engine Management system.
- The cruise control system will have an on and off switch on the windshield wiper that the driver interacts with.
- When the switch is turned on, the speed that the car is currently driving will be maintained.
- The cruise control system can accelerate or decelerate via gas pedal.
- The cruise control system will not be the mechanism that directly controls the speed of the car; the engine management system controls the speed of the car and the cruise control system sends a message to the engine management system of what speed it wants it to be.

Definitions, Acronyms, and Abbreviations

- Real time: a real-time constraint is a time constraint that must be specifically abided by.
- EMT: Engine Management System

General Constraints, Assumptions, Dependencies, Guidelines

- The system must work in realtime to provide a safe and comfortable user experience
- Reliability
    - Hardware 4-9 (99.99%) success rate
    - Software 5-9 (99.999%) success rate
- The system is dependent on active sensors with low-latency polling.
- Assumes that the actuator and sensors are functional and accurate.

User View of Product Use / User Personas and Characteristics

- The users will see this product as a new feature for their car that assists their driving.
- Drivers using this product are likely to be interested in improving their driving and/or seeking additional safety/comfort while driving.
- Some of these drivers may also have poor driving ability and are buying this product in order to prevent bad driving habits or violations, or in worst-case scenarios, events such as accidents.

User Stories

- As a driver, I want to interact with the system as little as possible during use, so that it is easy for me to use.
- As a driver, I want the system to help keep me and my vehicle safe by limiting my speed so that it isn't too fast or too slow while still putting me in control.

Product Perspective

- The product runs on a microchip that consists of software and hardware connected to sensors installed throughout the car. The software itself includes an OS that will monitor and perform calculations with the data from the sensors, and partially control the car's speed.

Users

- Those who will benefit and use the new system, as well as those affected, are as follows:
    - Customers: Customers can easily come into the possession of this cruise control system whenever they buy or lease a car that has one built-in.

    - Product Owners: Drivers will be able to drive more safely and effectively with this product, minimizing driving violations and accidents without encroaching too much on the drivers’ control of their vehicles.
    - Manufacturers: Manufacturers will be able to install the cruise control system in future cars, and be able to sell them at a higher price.
    - Sales Agents: Likewise, Sales Agents can advertise cars with the cruise control system installed and make a profit off of their sales.
    - Information Technology Department: This department will be responsible for keeping the software on the system up to date and working on any updates to the system when needed
    - Technicians: Car service technicians will be responsible for keeping the product up to date for product owners and running diagnostics in the unlikely event of a failure

Location

The system will be available to any customer who buys or owns the manufacturer’s brand of cars.

Responsibilities

- The responsibilities of the new system are as follows:
    - Monitor the vehicle’s current speed at all times.
    - Send messages to the car’s engine management system based on the current speed; the system should tell it to speed up if the car is going too slow and to slow down if the car is going too fast.

Need

The system is needed in order to provide a better experience for customers who are behind the wheel in highway driving environments.

Section 3 - Requirement section

Requirements

Inputs

1. Vehicle's current speed
2. Vehicle's acceleration/deceleration

Outputs

1. Constant speed sent to Engine Management System
2. accelerating/decelerating sent to Engine Management System

Functional Requirements

1. CC shall accept all inputs within 2 seconds after the engine starts
2. The CC system shall accept electric power from the alternator
3. CC shall turn on and await to set the speed within .5 seconds when the on button is pressed
4. CC shall shut off when the off button is pressed
5. CC shall toggle a light on the dashboard indicating that it has been activated and is currently on
6. CC shall set and keep current speed when the set button is pressed
7. CC shall stop maintaining speed and return to the on state, awaiting to be set again
8. CC shall gradually increase speed when the increase button is held until let go at which point the CC shall maintain that speed
9. CC shall gradually decrease speed when the decrease button is held until let go
10. CC shall completely turn off when any amount of braking is applied by the user
11. CC shall remember the speed and temporarily shut off when the pause button is activated
12. CC shall deactivate within .5 seconds after deactivation signal has been sent by user
13. The CC deactivation signal is set by brake or button on steering wheel
14. CC shall resume the speed that is held in memory when the resume button is activated This action shall only happen when it is in a paused state.
15. CC shall Log data from cruise control including:
    a. When the CC system is powered on
    b. When the CC system is turned off
    c. When CC is set to a speed
    d. When brakes have been applied
    e. When CC is paused

    f. When CC is resumed
    g. When speed is increased
    h. When speed is decreased

16. CC shall accept direct current from the car battery to support logging after the engine is shut down
17. CC shall receive continuous information from sensors about brake application every .5 seconds
18. CC shall accept signals from sensors(clutch, brake)
19. CC shall receive the time and date from the car’s clock every second
20. CC shall receive the current speed from EMS continuously every 1 second
21. CC shall provide visual feedback to the user about activation
22. CC shall provide visual feedback to the user about deactivation
23. CC shall provide visual feedback to the user about the setting of the speed
24. CC shall provide an activation request to the throttle (EMS)
25. System shall provide a deactivation request to the throttle (EMS)
26. CC shall provide an adjustment (increase or decrease) request to the throttle up to the allowable positions of speed
27. CC shall provide an approval request to the sensors before activation
28. CC shall provide a physical interference for Technicians to access the unit
29. System software shall be configurable by the Technician
30. CC shall have an accuracy range of +/- 1 mph of the user set speed
31. CC shall auto-shut off when the car is turned off by the user
    a. CC shall turn off and ensure logging data is kept
32. The CC system shall be supported by the battery and alternator of the car to ensure logging data is kept in the event of the user shutting the car off. The CC system shall shutdown after all logging data is successfully stored in memory to ensure that unnecessary battery is not drained.
33. CC shall adhere to the safe and secure communication protocol specified by the manufacturer, industry, and governments

Non-Functional Requirements

34. An interface for each of the required sensors listed in the following section
35. An interface to the software that allows loading/testing software, and viewing logging data
36. An interface to the EMS
37. An interface to the acceleration sensors that alert the system of a crash
38. Allocated memory of 1GB in the CC system for logging purposes
    a. Memory shall be wiped every week to ensure that memory shall always be available for logging

39. Response time of less than 100ms for any of the functional capabilities listed in the section above
40. Reliability of:
    a. Software 5-9s (99.999%) success rate
    b. Hardware: 4-9s (99.99%) success rate

Sensor Requirements

41. A sensor that shall have the ability to detect the current speed at all times
42. A sensor that shall have the ability to detect when the accelerator pedal is applied by the user
43. A sensor that shall have the ability to detect when the brake pedal is applied by the user
44. A sensor that shall have the ability to detect when the on button is pressed by the user
45. A sensor that shall have the ability to detect when the off button is pressed by the user
46. A sensor that shall have the ability to detect when the pause button is pressed by the user
47. A sensor that shall have the ability to detect when the resume button is pressed by the user
48. A sensor that shall have the ability to detect when the set speed button is pressed by the user
49. A sensor that shall have the ability to detect when the increase speed button is pressed by the user
50. A sensor that shall have the ability to detect when the decrease speed button is pressed by the user

Security Requirements

51. No opportunities for external influences to system
    a. Only interfaces are contained in the engine bay of the car
52. Admin interface needs to be physically based in the CC hardware
    a. Only technicians have the ability to connect physically to the unit to update software, download logs
    b. No outside connectivity in the form of bluetooth, internet/wireless, radio

Testing of Requirements

- Each individual functional requirement shall be tested 1000 times in the lab and the results shall be documented, including whether the requirement functions correctly or not, and the response time. If a functional requirement fails, the system shall be inspected of any problems or defects that caused the failure, and adjustments shall be made accordingly.
- Each non-functional requirement shall also be tested:
    - 1-4: Each interface shall be inspected to ensure that they appear satisfactory.

    - 5: System’s memory shall be tested by logging and wiping data to ensure that it functions properly.
    - 6: The response time from testing the functional requirements shall be analyzed to ensure that each requirement responds in less than 100ms. If any of the requirements do not reach this goal, adjustments shall be made to the system so that the requirement does perform quickly enough.
    - 7: Reliability of the software and hardware shall be tested again and the results shall be documented again. If either the software or hardware does not reach the required success rate, they shall be inspected and improved before being tested again.
- Each of the system’s sensors shall be tested 100 times in the lab to check whether they respond to the correct inputs and send the data to the system.

Section 4 - Uses Cases & Modeling

UML Use Cases & Diagram

Use Case 1: Turn on the Cruise Control via the button on the button on the steering wheel

Iteration: 1, Created 02/12/20
Primary Actor: Driver
Goal in Context: To set the speed of the car at a constant speed.
Preconditions: Car must be turned on and the driver must be driving.
Trigger: The driver decides to use the cruise control for ease of driving

1. User sends an activation signal.
2. System provides feedback, letting users know that Cruise Control is ready.
3. User tells the Cruise Control system to maintain the current speed.
4. Cruise Control system queries sensor data.
5. Sensor data tells Cruise Control system current speed, acceleration, break.
6. Cruise Control sends data to the Engine Management System with desired speed.
7. Engine Management System sets the throttle accordingly and responds to Cruise Control with a success signal.

8. Cruise Control continuously queues sensor data of vehicles speed.

Exceptions:
1. Sensors do not respond, or respond with blatantly faulty information - Cruise Control does not activate, alerts users.
2. Engine Management system does not respond to Cruise control with a success signal after query - Cruise Control does not activate, alerts users.

Use Case 2: Turn off the Cruise Control via the button on the steering wheel

Iteration: 1, Created 02/12/20
Primary Actor: Driver
Goal in Context: To turn off cruise control and let the driver control the speed or to shut down the speed of the car completely.
Preconditions: Car must be turned on and be set in cruise control.
Trigger: The driver decides to stop using cruise control to change the speed themselves. An accident happens and the cruise control must be shut down.

1. User sends a deactivation signal via pressing the button on the steering wheel.
2. The signal is sent to the Engine Management System to turn off the cruise control.
3. Cruise control sends signals to sensors to stop sending data.
4. Sensors stop sending data to cruise control.
5. Cruise control deactivated.

Exceptions
1. Sensors still send data to the system after being given the signal to stop - Cruise Control remains on, alerts user.
2. Cruise control fails to deactivate -Cruise Control remains on, alerts user

Use Case 3: Accelerate the Cruise Control via the gas pedal

Iteration: 1, Created 02/12/20
Primary Actor: Driver
Goal in Context: To let the driver change the speed of the car while the cruise control is turned on.
Preconditions: Car must be turned on and cruise control is turned on.
Trigger: The driver wants to accelerate the vehicle.

1. The user's vehicle is at a constant speed, and cruise control is on.
2. The user presses on the gas pedal, causing the vehicle to accelerate.
3. The cruise control temporarily pauses.
4. The Engine Management System detects the change in speed and changes the throttle.
5. Sensors queue speed data to cruise control.

6. After the driver releases the gas pedal, the cruise control re-enables itself with the vehicle's current speed.
7. Cruise Control continuously queues sensor data of vehicle's speed.

Exceptions:

1. Vehicle slows to a halt, but the engine is still running -Cruise control re-enables itself when the vehicle begins moving again.
2. Vehicle slows to a halt, and driver turns off the engine -Cruise control automatically turns off as if the driver manually deactivated it
3. Cruise control does not resume - Cruise Control does not activate, alert user.

Use Case 3: Decelerate the Cruise Control via the brake pedal

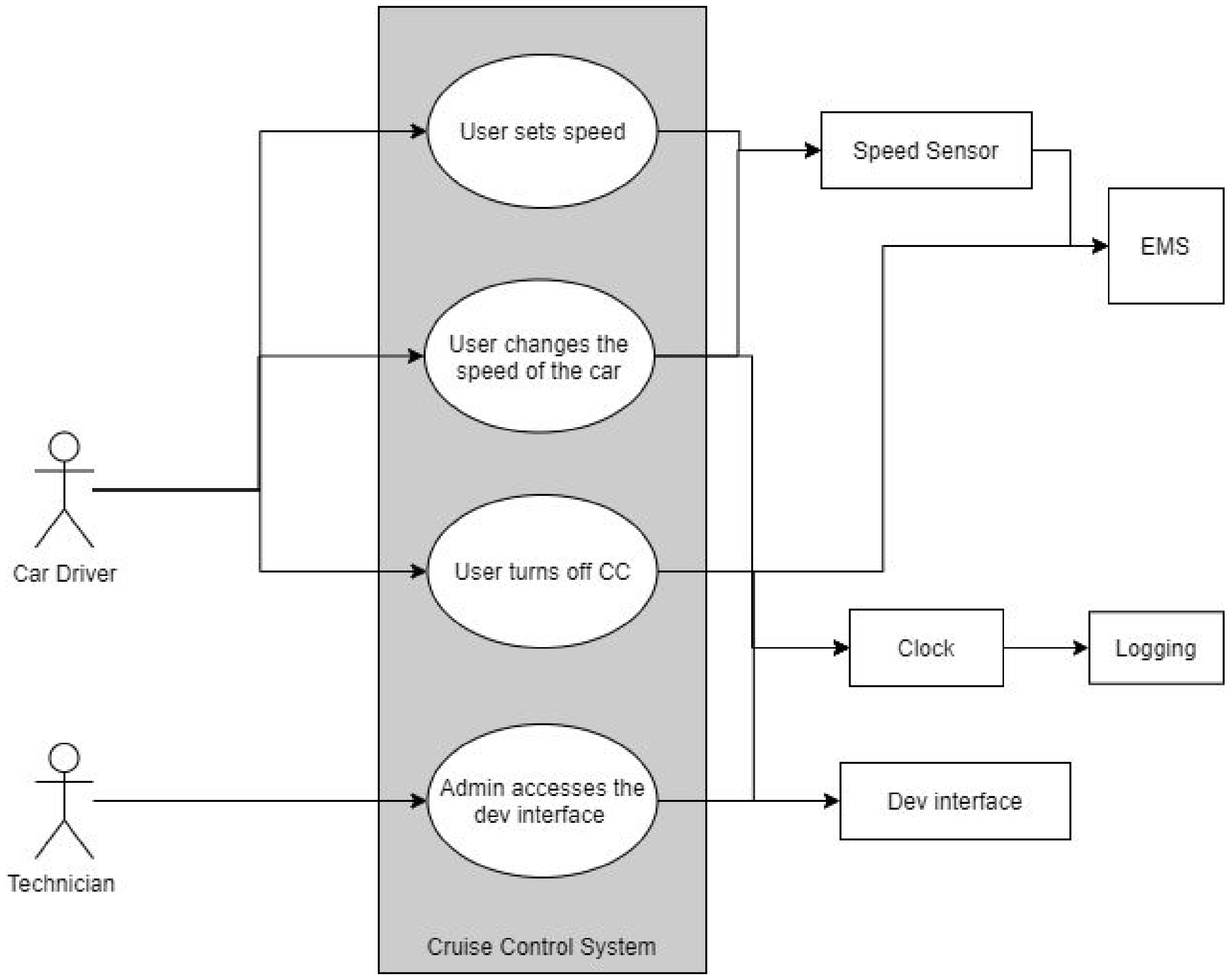
Iteration: 1, Created 02/12/20
Primary Actor: Driver
Goal in Context: To let the driver change the speed of the car while the cruise control is turned on.
Preconditions: Car must be turned on and cruise control must be turned on.
Trigger: The driver wants to slow down the vehicle.

1. The user's vehicle is at a constant speed, and cruise control is on.
2. The user presses on the brake pedal, causing the vehicle to decelerate
3. The cruise control temporarily pauses.
4. The Engine Management System detects the change in speed and changes the throttle.
5. Sensors queue speed data to cruise control.
6. After the driver releases the gas/brake pedal and the vehicle stops decelerating, the cruise control re-enables itself with the vehicle's current speed.
7. Cruise Control continuously queues sensor data of vehicle's speed.

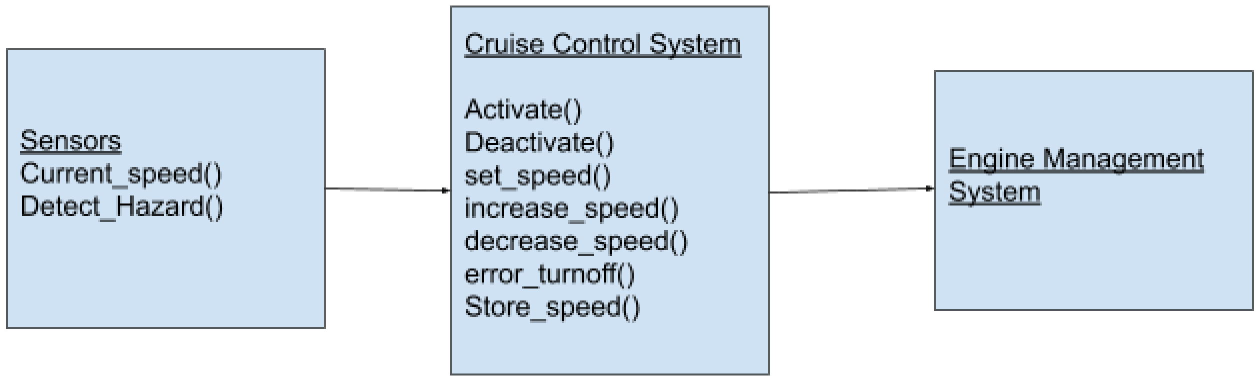
Exceptions:

4. Vehicle slows to a halt, but the engine is still running -Cruise control re-enables itself when the vehicle begins moving again
5. Vehicle slows to a halt, and driver turns off the engine -Cruise control automatically turns off as if the driver manually deactivated it
6. Cruise control does not resume - Cruise Control does not activate, alert user.

Use Case Diagram



UML Class-Based Modeling



UML CRC Model Index Card

Class : Cruise Control System On/Off	
Responsibility:	**Collaborator:**
Turn on the CC system	Command Control Class
Turn off the CC system	Command Control Class

Class : Command Control	
Responsibility:	**Collaborator:**
Turn on indicator light	Display Interface
Activate CC system	Sensors
Display error/hazards	Display Interface
Increase speed	EMS, Sensors
Decrease Speed	EMS, Sensors
Maintain Speed	EMS, Sensors

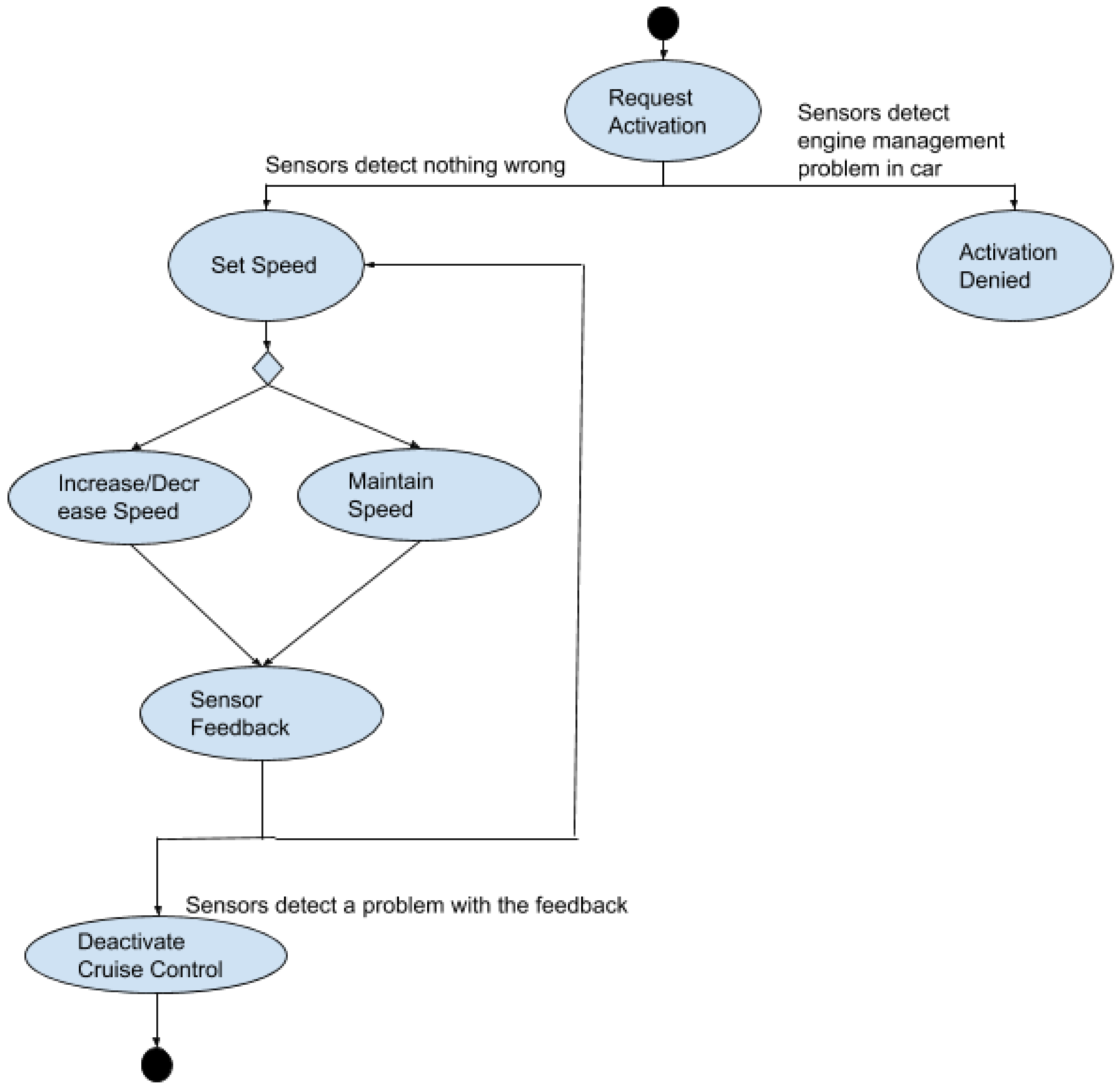
Class : Sensors	
Responsibility:	**Collaborator:**
Calculates current speed	
Detects Hazards	Engine Management System

Class : Data	

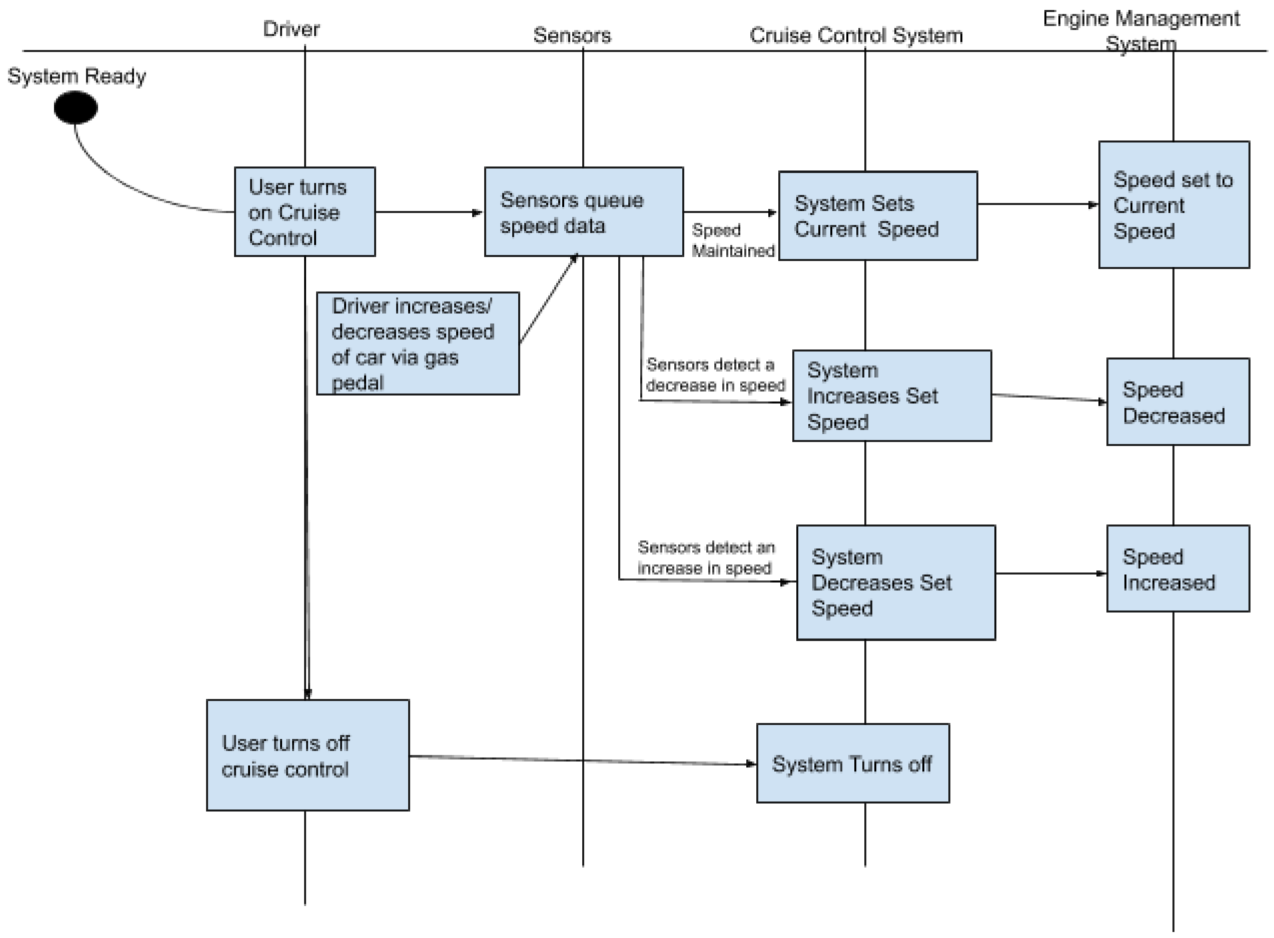
Responsibility:	**Collaborator:**
Get time	
Get current Speed	EMS

Class : Log	
Responsibility:	**Collaborator:**
Write to log	Command Control, Sensors

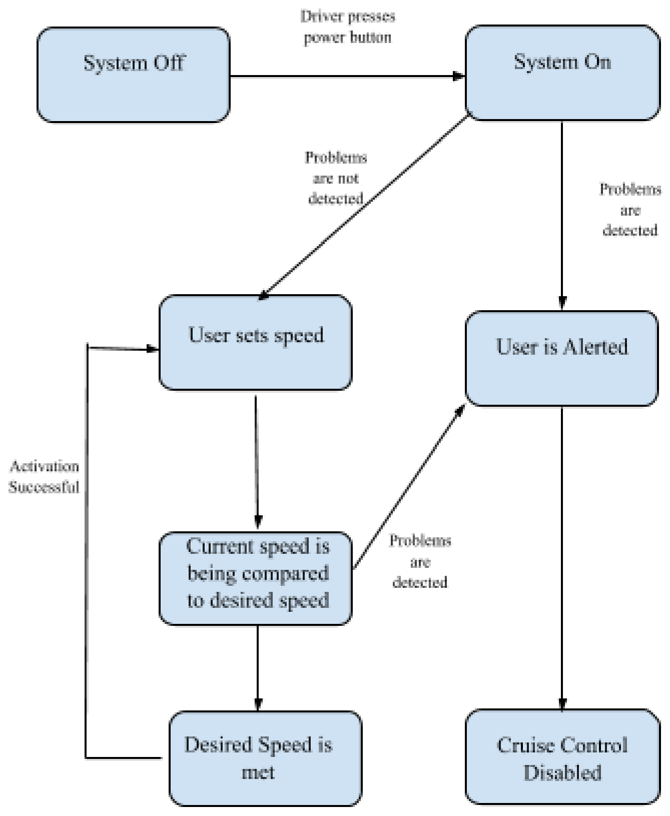
UML Activity Diagram


Request Activation
Sensors detect nothing wrong
Sensors detect engine management problem in car
Set Speed
Activation Denied
Increase/Decrease Speed
Maintain Speed
Sensor Feedback
Sensors detect a problem with the feedback
Deactivate Cruise Control

UML Sequence Diagram


Driver
Sensors
Cruise Control System
Engine Management System
System Ready
User turns on Cruise Control
Sensors queue speed data
Speed Maintained
System Sets Current Speed
Speed set to Current Speed
Driver increases/ decreases speed of car via gas pedal
Sensors detect a decrease in speed
System Increases Set Speed
Speed Decreased
Sensors detect an increase in speed
System Decreases Set Speed
Speed Increased
User turns off cruise control
System Turns off

UML State Diagram



Section 5: Cruise Control Software Architecture

CC's Architecture Style

The architecture style used in the CC will be Object Oriented Architecture and Layered Architecture implemented into a Call and Return Architecture.

It is possible to include a data repository for the CC's client software to access due to the CC constantly receiving data from various facets of the vehicle, like the Data-Centered style. This shall provide a single area in which the data the CC logs can be stored and modified, as it is necessary because it is heavily reliant on the data it receives from its sensors. The data it will be storing is the speed of the car and time when the system is active. The advantage of storing data in one spot is that it will be easy to go through iterations of logs if the cruise control is to be updated. The drawback of storing data in one file is that the file might get big and could potentially slow down, or the file exceeds the system's memory and could impair the CC's ability to function. To remedy this issue, the system will give the user the ability to access and/or delete the data stored in the CC. Additionally, it will be hard to model the cruise control since information needs to be passed to different components and information will be coming in from other components.

Elements of Object-Oriented-Programming will also be utilized in this architectural style, as the system relies upon methods used by several components to control the vehicle's speed and utilize and/or manipulate data obtained from the sensors. This shall allow the system to easily handle the data, and simplifies the components to be more manageable by the system. However, Object-Oriented-Programming is inherently slower than other architecture styles, as functions must also perform other tasks such as store data and shuffle around programs aside from their intended use.

The programs and components could be organized in a fashion similar to that of the Layered Architecture style, as they can easily be divided into those involving the user interface, and those that actually manipulate or interact with the vehicle's inner components, which the user cannot directly interface with the CC. Similar to the Call-and-Return aspects of this CC's architectural style, this shall organize the system's components in a way that is easy to modify and understand how said components affect and depend on each other. The core layer will be software that is closest to interacting with the engine management system, such as those which control speed The outer layer will be closest to interacting with the user, such as those responsible for turning the system on/off. The advantage of this is that it will provide greater ease of use by isolating programs that the user needs to know while keeping them away from programs that they do not necessarily need to understand and/or may cause them confusion. The drawback to this is that it will be much more difficult for the user to resolve issues with their CC and they must consult someone familiar with the system. Another drawback is that input from the user and sensors are both inputs into the program and that is harder to model in a Layered Architecture style.

Call and Return style can be integrated because it will allow for a hierarchy to exist in the program. The hierarchy can provide organization while also minimizing the risk of commands to go out of order. The flow of data can be consistent to subprograms and not interfere with other parts of the cruise control. The smaller subprograms will also be easier to create. The drawbacks of using call and return is that if there are any complex data structures we may want to implement on a new iteration of the cruise control, the call and return will be hard to support the data structures.

Finally, Object oriented implementation of call and return will be the architectural style of the CC. Object oriented style will make it easy to create objects for each component of the cruise control software and make it easy to manipulate them or store information in them. Call and Return will help organize the data in a hierarchical fashion that can also have each subcomponent act as states. This will be easier to implement and safer since two things at once can not occur. For example, speeding up and braking; these will not occur at the same time in an object oriented implementation of call and return. Constraints are the CC cannot add another interface because the architectural style will not have another port for it to connect to. The connectors that exist between the car and the CC software are: the Engine management system and the CC software, sensors and the CC software, driver and CC software, and the button interface and the CC software. The connectors that exist in the software are between: the on and off switch to Command control, command control to sensors. Sensors to data, data to log, data to Command control.

Components of the Software

In the outermost layer of the software's architecture, there are components that revolve around detecting user input, such as whether the user powers on/off the system, presses a button, or manually influences the vehicle's speed. The next layer of components include those that perform the functions linked to the various user inputs it detects, those that receive and manipulate the data stored in the system and/or received through its sensors, the sensors themselves, and others which provide notifications (such as system errors) to the user through the interface. The deepest and final layer is composed of the Engine Management System and affects the engine itself, which is mainly responsible for maintaining the vehicle's speed when the system is on. Most of these components are dependent on other components in different layers, hence why they are connected. One possible limitation with the software is that if one of these components fails, then other components that are dependent on it will also fail, which could result in the entire system being unusable if a key component is malfunctioning.

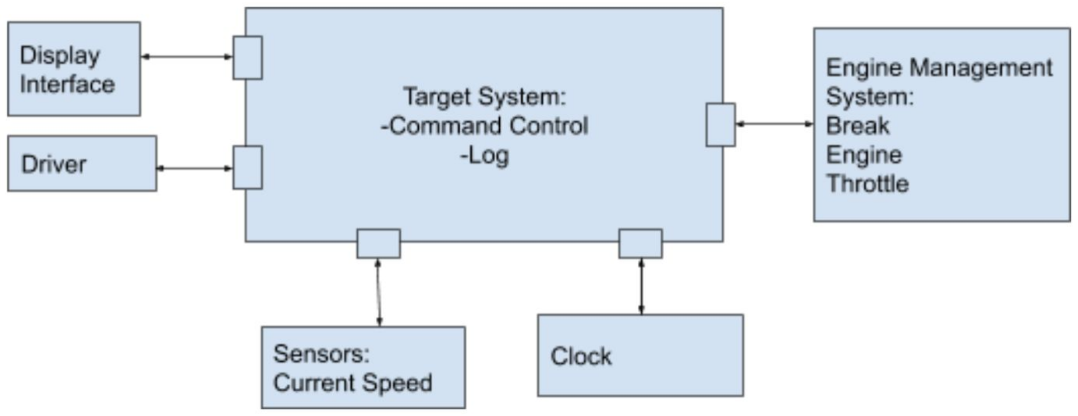
Control Management

The control is managed in two parts. The first program of control is turning the cruise control on and off. The signal to turn the cruise control on and off is managed by the Activator control which is in charge of turning the cruise control on and off. The Control Activator will check for errors and if it detects one, the cruise control will not activate, and the system will notify the driver. Then control is passed to the second program. The second program of the control is utilizing the input from the user and directing it to where it needs to be applied. It will control the speed of the acceleration, deceleration, and constant speed of the car. The user directs the control and the control directs the speed of the car. If the user chooses to temporarily deactivate the CC, the program will cease all influence from the system to the vehicle's engine, but will not shut down the software. Likewise, the program will also detect when the user provides the signal to reactivate the CC, which will resume control over the engine and maintain the vehicle's desired speed.

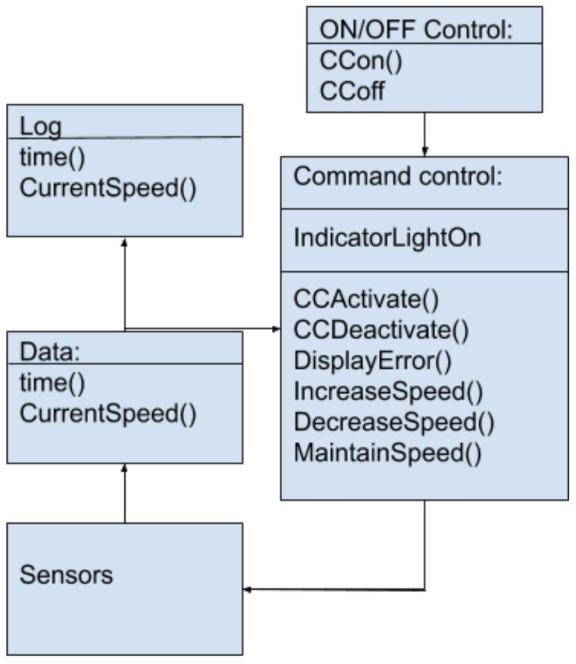
Data Architecture

Overall, the data architecture of the CC consists of layers of components that are interconnected. Data is communicated through the use of classes, and data objects will be passed to the system sporadically when they are required. For example, when the current speed needs to be set, that data will be requested and passed by the proper sensor class. The call and return elements ensure the most current data is passed. A small memory and storage unit will be required to store the data of the logs made by the system. When any functional components perform an action, they will send their data to the memory and storage for the purpose of logging. The data components will act in a passive manner. Control will request data from the sensors, which will be passed in the form of class methods using the call and return method.

Architectural Design



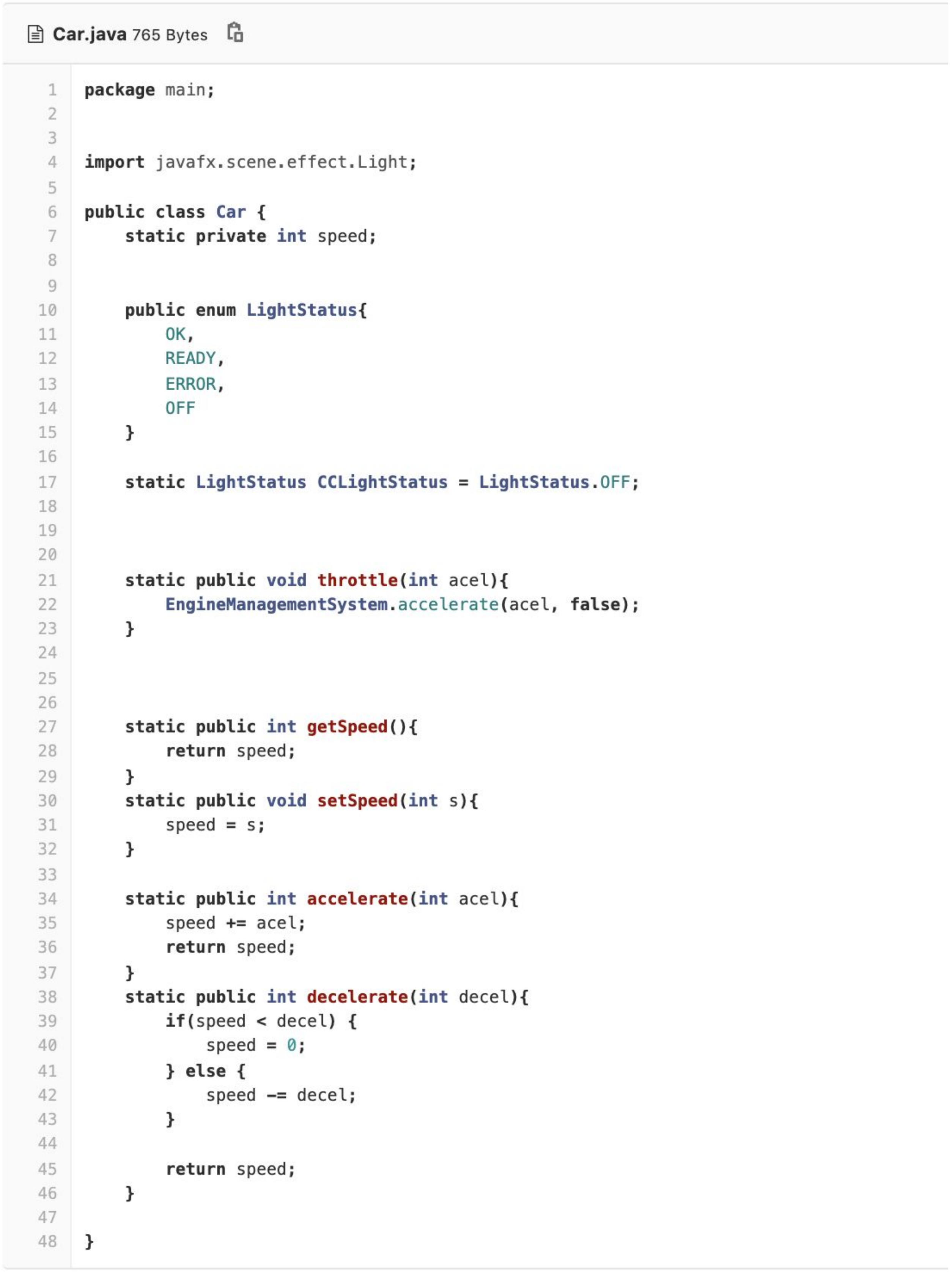
Architectural Style



Issues

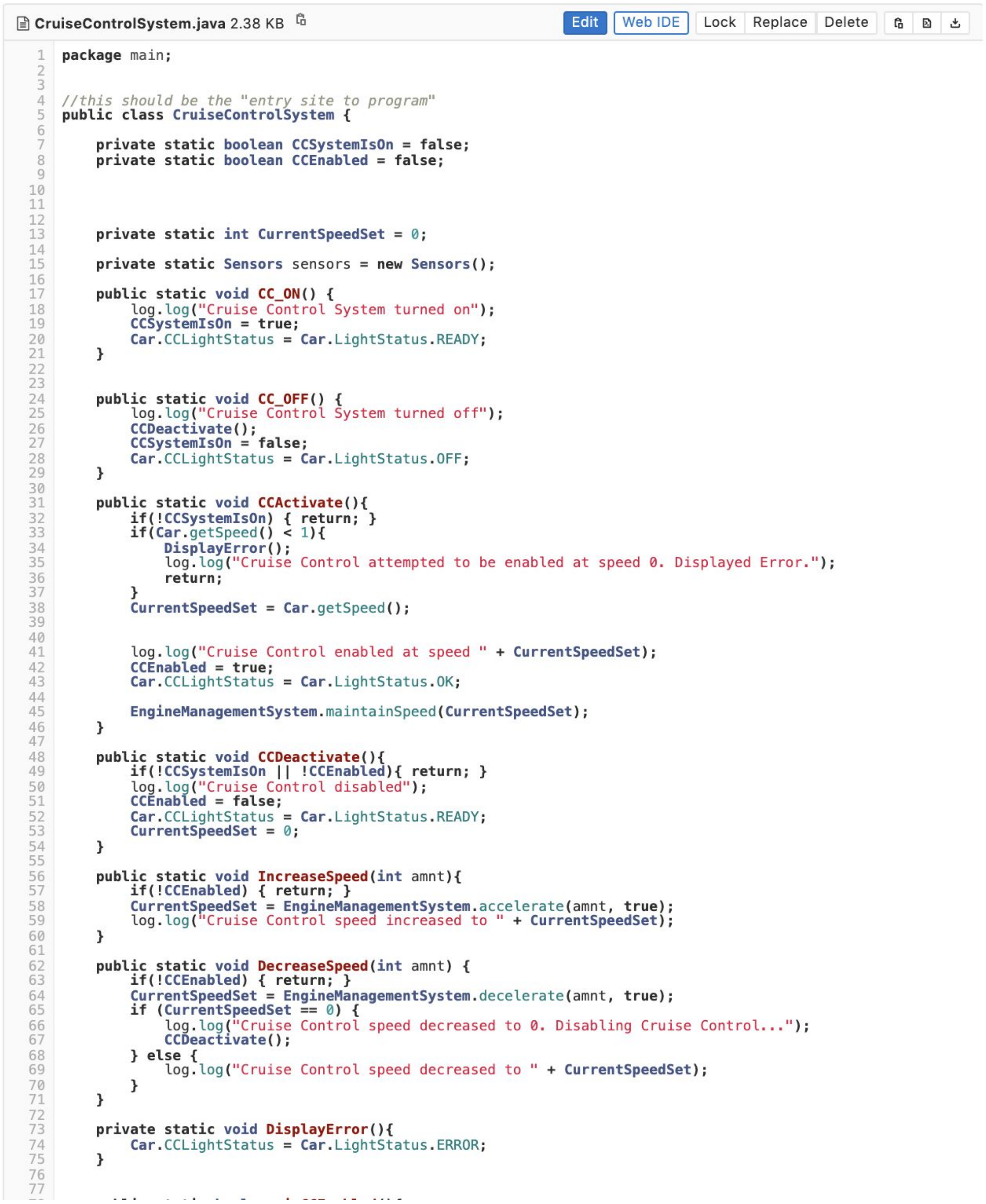
1)
    a) Reliability - Mission critical, Should have a 99.999 success rate for software and a 99.99 success rate for hardware.
    b) Useability - simplicity, buttons clearly labeled with functionality
    c) Maintainability - depends on the function and maintenance of the car.
    d) Functionality - needs to work in real time.
    e) Performance - just needs to be in real time, the car needs to be responsive to user commands.
2) The diagrams are made to support the requirements. If there are any issues that occur while developing the CC, the diagrams might have to be slightly modified to meet the requirements.
3) The systems structure and pattern structure both use the sensors to make sure the car is at the correct speed and what speed the car is currently at. The user is what activates the structures and pattern.
4) The pattern has a positive effect on the quality attributes because it makes sure all the quality attributes are being met.
5) Issues could arise of commands being out of order and subsequently data being corrupted but this is resolved through the use of call and return architecture.
6) Issues uncovered while making the diagrams was how to make sure that commands from the user were being processed in a certain order so issues would not occur. Another issue was how the cruise control would interact with the interfaces of the car.

Section 6: Code



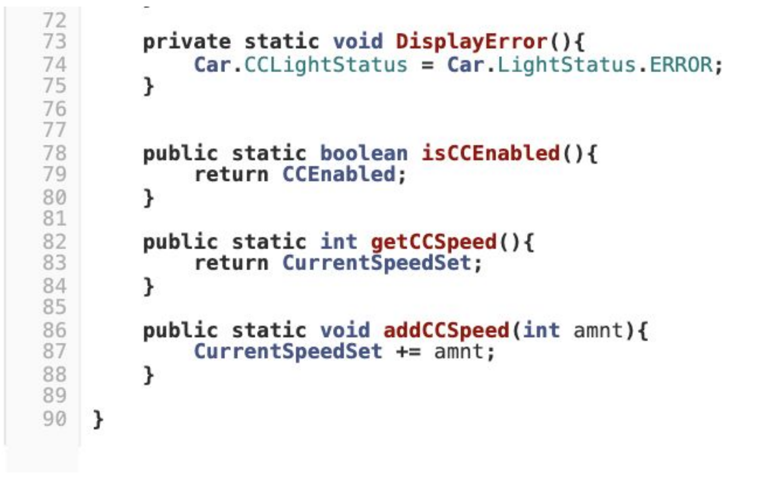
Car.java 765 Bytes

```java
package main;


import javafx.scene.effect.Light;

public class Car {
    static private int speed;


    public enum LightStatus{
        OK,
        READY,
        ERROR,
        OFF
    }

    static LightStatus CCLightStatus = LightStatus.OFF;



    static public void throttle(int acel){
        EngineManagementSystem.accelerate(acel, false);
    }



    static public int getSpeed(){
        return speed;
    }
    static public void setSpeed(int s){
        speed = s;
    }

    static public int accelerate(int acel){
        speed += acel;
        return speed;
    }
    static public int decelerate(int decel){
        if(speed < decel) {
            speed = 0;
        } else {
            speed -= decel;
        }

        return speed;
    }

}
```

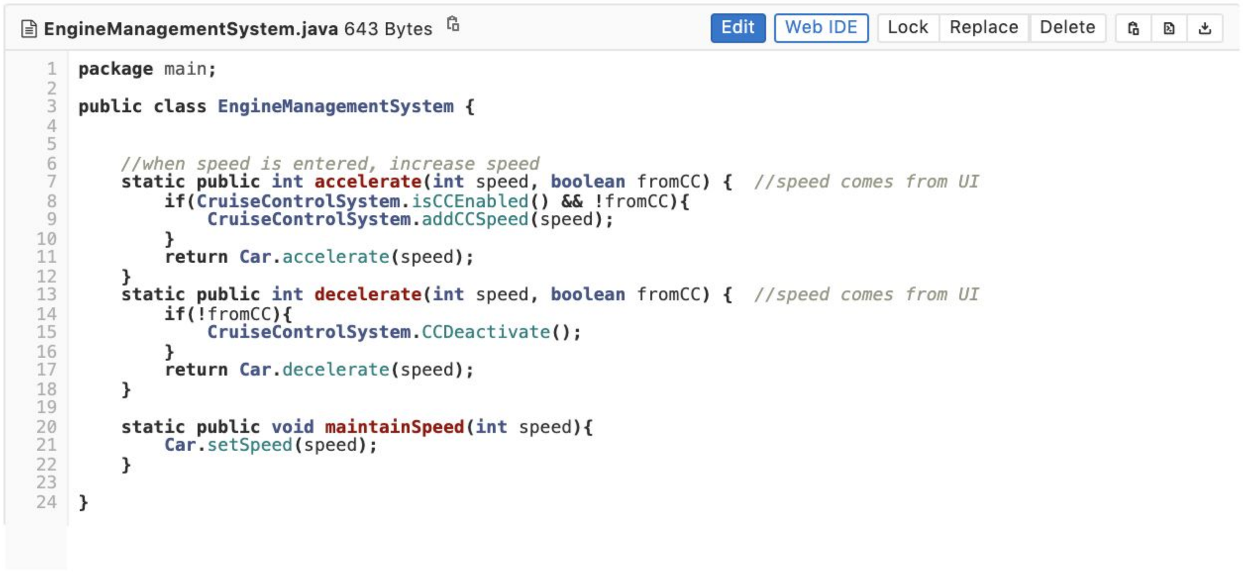


CruiseControlSystem.java 2.38 KB

```java
package main;


//this should be the "entry site to program"
public class CruiseControlSystem {

    private static boolean CCSystemIsOn = false;
    private static boolean CCEnabled = false;




    private static int CurrentSpeedSet = 0;

    private static Sensors sensors = new Sensors();

    public static void CC_ON() {
        log.log("Cruise Control System turned on");
        CCSystemIsOn = true;
        Car.CCLightStatus = Car.LightStatus.READY;
    }


    public static void CC_OFF() {
        log.log("Cruise Control System turned off");
        CCDeactivate();
        CCSystemIsOn = false;
        Car.CCLightStatus = Car.LightStatus.OFF;
    }

    public static void CCActivate(){
        if(!CCSystemIsOn) { return; }
        if(Car.getSpeed() < 1){
            DisplayError();
            log.log("Cruise Control attempted to be enabled at speed 0. Displayed Error.");
            return;
        }
        CurrentSpeedSet = Car.getSpeed();


        log.log("Cruise Control enabled at speed " + CurrentSpeedSet);
        CCEnabled = true;
        Car.CCLightStatus = Car.LightStatus.OK;

        EngineManagementSystem.maintainSpeed(CurrentSpeedSet);
    }

    public static void CCDeactivate(){
        if(!CCSystemIsOn || !CCEnabled){ return; }
        log.log("Cruise Control disabled");
        CCEnabled = false;
        Car.CCLightStatus = Car.LightStatus.READY;
        CurrentSpeedSet = 0;
    }

    public static void IncreaseSpeed(int amnt){
        if(!CCEnabled) { return; }
        CurrentSpeedSet = EngineManagementSystem.accelerate(amnt, true);
        log.log("Cruise Control speed increased to " + CurrentSpeedSet);
    }

    public static void DecreaseSpeed(int amnt) {
        if(!CCEnabled) { return; }
        CurrentSpeedSet = EngineManagementSystem.decelerate(amnt, true);
        if (CurrentSpeedSet == 0) {
            log.log("Cruise Control speed decreased to 0. Disabling Cruise Control...");
            CCDeactivate();
        } else {
            log.log("Cruise Control speed decreased to " + CurrentSpeedSet);
        }
    }

    private static void DisplayError(){
        Car.CCLightStatus = Car.LightStatus.ERROR;
    }


```



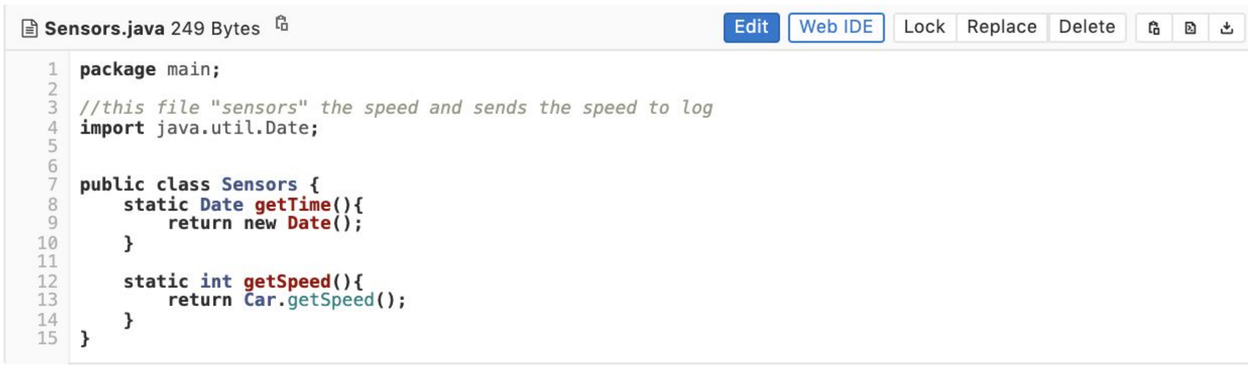
```java
    private static void DisplayError(){
        Car.CCLightStatus = Car.LightStatus.ERROR;
    }


    public static boolean isCCEnabled(){
        return CCEnabled;
    }

    public static int getCCSpeed(){
        return CurrentSpeedSet;
    }

    public static void addCCSpeed(int amnt){
        CurrentSpeedSet += amnt;
    }

}
```



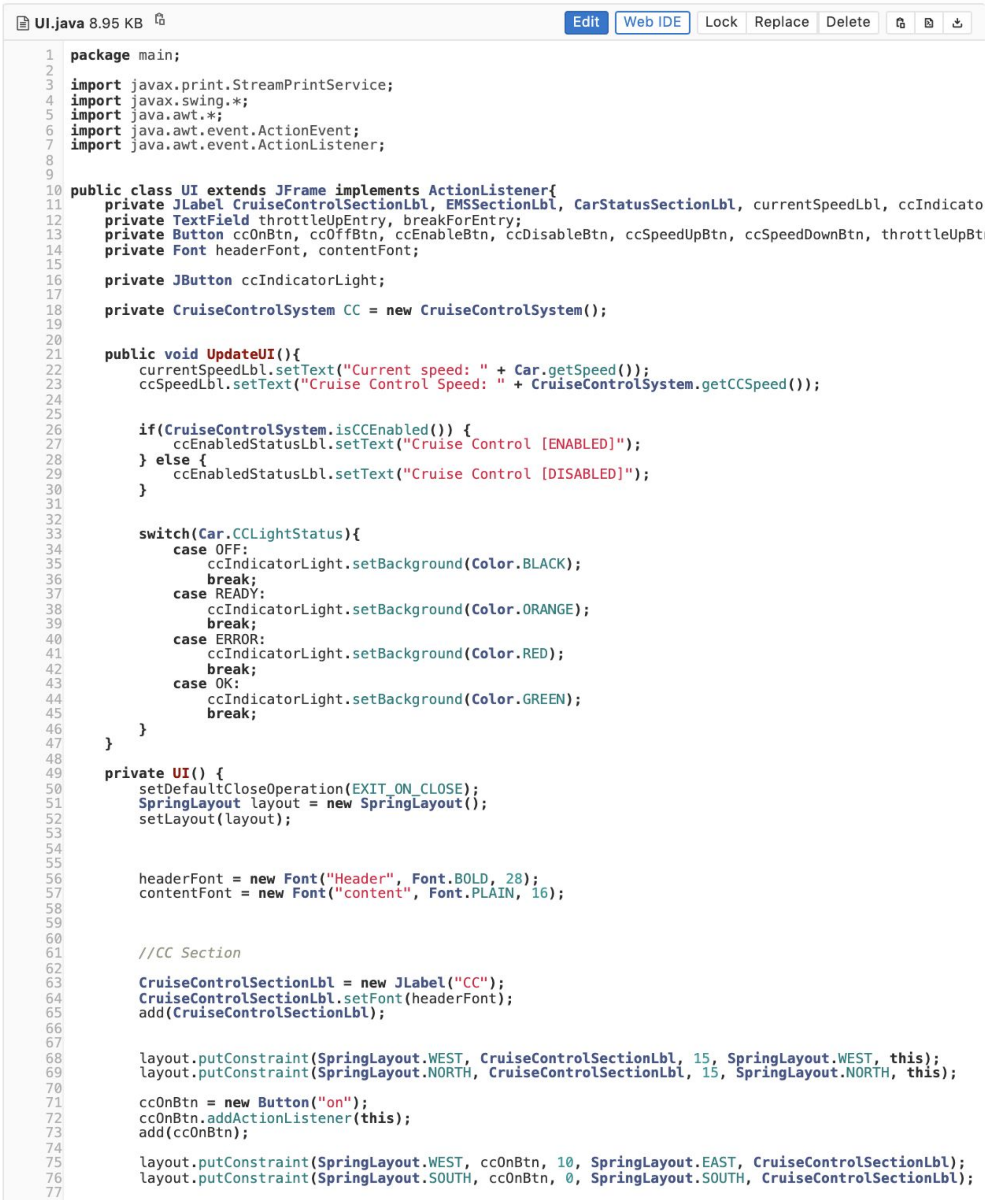
EngineManagementSystem.java 643 Bytes

Edit | Web IDE | Lock | Replace | Delete

```java
package main;

public class EngineManagementSystem {


    //when speed is entered, increase speed
    static public int accelerate(int speed, boolean fromCC) {  //speed comes from UI
        if(CruiseControlSystem.isCCEnabled() && !fromCC){
            CruiseControlSystem.addCCSpeed(speed);
        }
        return Car.accelerate(speed);
    }
    static public int decelerate(int speed, boolean fromCC) {  //speed comes from UI
        if(!fromCC){
            CruiseControlSystem.CCDeactivate();
        }
        return Car.decelerate(speed);
    }

    static public void maintainSpeed(int speed){
        Car.setSpeed(speed);
    }

}
```



```java
package main;

//this file "sensors" the speed and sends the speed to log
import java.util.Date;


public class Sensors {
    static Date getTime(){
        return new Date();
    }

    static int getSpeed(){
        return Car.getSpeed();
    }
}
```

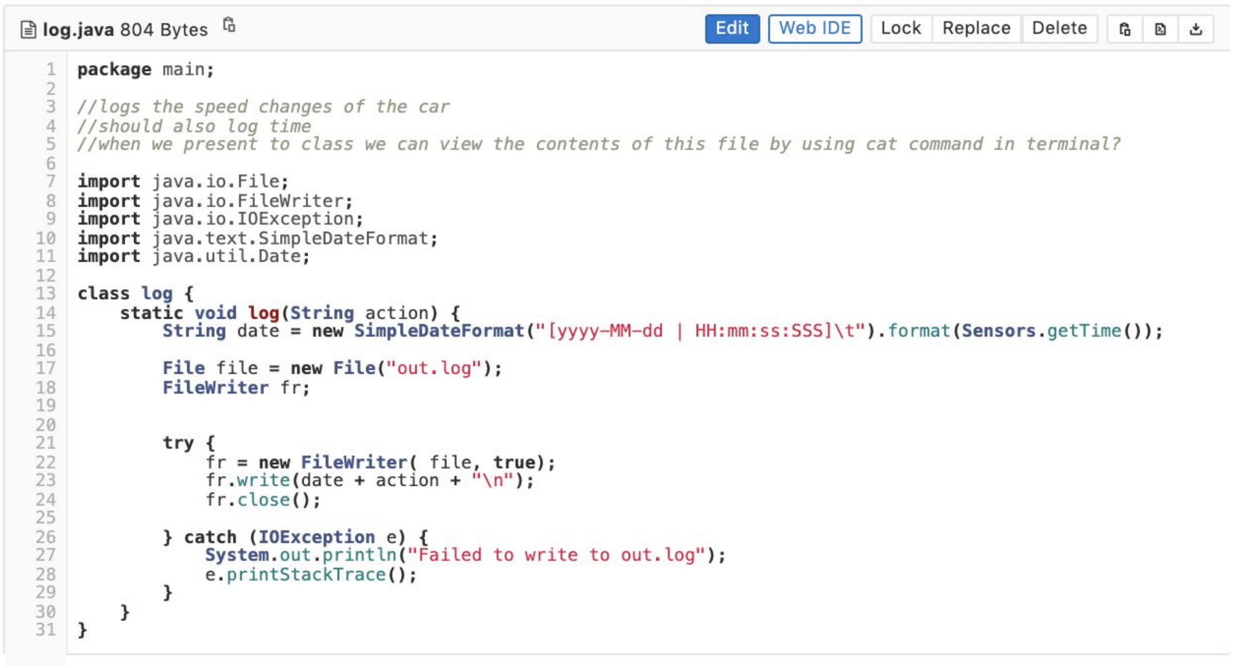


UI.java 8.95 KB

Edit Web IDE Lock Replace Delete

```java
package main;

import javax.print.StreamPrintService;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class UI extends JFrame implements ActionListener{
    private JLabel CruiseControlSectionLbl, EMSSectionLbl, CarStatusSectionLbl, currentSpeedLbl, ccIndicato
    private TextField throttleUpEntry, breakForEntry;
    private Button ccOnBtn, ccOffBtn, ccEnableBtn, ccDisableBtn, ccSpeedUpBtn, ccSpeedDownBtn, throttleUpBt
    private Font headerFont, contentFont;

    private JButton ccIndicatorLight;

    private CruiseControlSystem CC = new CruiseControlSystem();


    public void UpdateUI(){
        currentSpeedLbl.setText("Current speed: " + Car.getSpeed());
        ccSpeedLbl.setText("Cruise Control Speed: " + CruiseControlSystem.getCCSpeed());


        if(CruiseControlSystem.isCCEnabled()) {
            ccEnabledStatusLbl.setText("Cruise Control [ENABLED]");
        } else {
            ccEnabledStatusLbl.setText("Cruise Control [DISABLED]");
        }


        switch(Car.CCLightStatus){
            case OFF:
                ccIndicatorLight.setBackground(Color.BLACK);
                break;
            case READY:
                ccIndicatorLight.setBackground(Color.ORANGE);
                break;
            case ERROR:
                ccIndicatorLight.setBackground(Color.RED);
                break;
            case OK:
                ccIndicatorLight.setBackground(Color.GREEN);
                break;
        }
    }

    private UI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        SpringLayout layout = new SpringLayout();
        setLayout(layout);



        headerFont = new Font("Header", Font.BOLD, 28);
        contentFont = new Font("content", Font.PLAIN, 16);



        //CC Section

        CruiseControlSectionLbl = new JLabel("CC");
        CruiseControlSectionLbl.setFont(headerFont);
        add(CruiseControlSectionLbl);


        layout.putConstraint(SpringLayout.WEST, CruiseControlSectionLbl, 15, SpringLayout.WEST, this);
        layout.putConstraint(SpringLayout.NORTH, CruiseControlSectionLbl, 15, SpringLayout.NORTH, this);

        ccOnBtn = new Button("on");
        ccOnBtn.addActionListener(this);
        add(ccOnBtn);

        layout.putConstraint(SpringLayout.WEST, ccOnBtn, 10, SpringLayout.EAST, CruiseControlSectionLbl);
        layout.putConstraint(SpringLayout.SOUTH, ccOnBtn, 0, SpringLayout.SOUTH, CruiseControlSectionLbl);

        add(ccOnBtn);

        layout.putConstraint(SpringLayout.WEST, ccOnBtn, 10, SpringLayout.EAST, CruiseControlSectionLbl);
        layout.putConstraint(SpringLayout.SOUTH, ccOnBtn, 0, SpringLayout.SOUTH, CruiseControlSectionLbl);

        ccOffBtn = new Button("off");
        ccOffBtn.addActionListener(this);
        ccOffBtn.setEnabled(false);
        add(ccOffBtn);

        layout.putConstraint(SpringLayout.WEST, ccOffBtn, 10, SpringLayout.EAST, ccOnBtn);
        layout.putConstraint(SpringLayout.SOUTH, ccOffBtn, 0, SpringLayout.SOUTH, CruiseControlSectionLbl);

        ccEnableBtn = new Button("enable");
        ccEnableBtn.addActionListener(this);
        add(ccEnableBtn);

        layout.putConstraint(SpringLayout.WEST, ccEnableBtn, 0, SpringLayout.WEST, CruiseControlSectionLbl)
        layout.putConstraint(SpringLayout.NORTH, ccEnableBtn, 15, SpringLayout.SOUTH, CruiseControlSectionL


        ccDisableBtn = new Button("disable");
        ccDisableBtn.addActionListener(this);
        add(ccDisableBtn);

        layout.putConstraint(SpringLayout.WEST, ccDisableBtn, 0, SpringLayout.WEST, CruiseControlSectionLbl
        layout.putConstraint(SpringLayout.NORTH, ccDisableBtn, 15, SpringLayout.SOUTH, ccEnableBtn);


        ccSpeedUpBtn = new Button("Increase CC Speed");
        ccSpeedUpBtn.addActionListener(this);
        add(ccSpeedUpBtn);

        layout.putConstraint(SpringLayout.WEST, ccSpeedUpBtn, 0, SpringLayout.WEST, CruiseControlSectionLbl
        layout.putConstraint(SpringLayout.NORTH, ccSpeedUpBtn, 15, SpringLayout.SOUTH, ccDisableBtn);


        ccSpeedDownBtn = new Button("Decrease CC Speed");
        ccSpeedDownBtn.addActionListener(this);
        add(ccSpeedDownBtn);

        layout.putConstraint(SpringLayout.WEST, ccSpeedDownBtn, 0, SpringLayout.WEST, CruiseControlSectionL
        layout.putConstraint(SpringLayout.NORTH, ccSpeedDownBtn, 15, SpringLayout.SOUTH, ccSpeedUpBtn);




        //EMS section

        EMSSectionLbl = new JLabel("EMS");
        EMSSectionLbl.setFont(headerFont);
        add(EMSSectionLbl);


        layout.putConstraint(SpringLayout.WEST, EMSSectionLbl, 15, SpringLayout.WEST, this);
        layout.putConstraint(SpringLayout.NORTH, EMSSectionLbl, 450, SpringLayout.NORTH, CruiseControlSecti


        throttleUpBtn = new Button("Throttle up by");
        throttleUpBtn.addActionListener(this);
        add(throttleUpBtn);

        layout.putConstraint(SpringLayout.WEST, throttleUpBtn, 0, SpringLayout.WEST, EMSSectionLbl);
        layout.putConstraint(SpringLayout.NORTH, throttleUpBtn, 15, SpringLayout.SOUTH, EMSSectionLbl);

        throttleUpEntry = new TextField("0");
        add(throttleUpEntry);

        layout.putConstraint(SpringLayout.WEST, throttleUpEntry, 5, SpringLayout.EAST, throttleUpBtn);
        layout.putConstraint(SpringLayout.NORTH, throttleUpEntry, 0, SpringLayout.NORTH, throttleUpBtn);


        breakForBtn = new Button("Break for");
        breakForBtn.addActionListener(this);
        add(breakForBtn);

        layout.putConstraint(SpringLayout.WEST, breakForBtn, 0, SpringLayout.WEST, EMSSectionLbl);
        layout.putConstraint(SpringLayout.NORTH, breakForBtn, 15, SpringLayout.SOUTH, throttleUpBtn);

        breakForEntry = new TextField("0");
        add(breakForEntry);
```

```java
        breakForBtn = new Button("Break for");
        breakForBtn.addActionListener(this);
        add(breakForBtn);

        layout.putConstraint(SpringLayout.WEST, breakForBtn, 0, SpringLayout.WEST, EMSSectionLbl);
        layout.putConstraint(SpringLayout.NORTH, breakForBtn, 15, SpringLayout.SOUTH, throttleUpBtn);

        breakForEntry = new TextField("0");
        add(breakForEntry);

        layout.putConstraint(SpringLayout.WEST, breakForEntry, 5, SpringLayout.EAST, breakForBtn);
        layout.putConstraint(SpringLayout.NORTH, breakForEntry, 0, SpringLayout.NORTH, breakForBtn);



        //Car Status Section

        CarStatusSectionLbl = new JLabel("Car Status");
        CarStatusSectionLbl.setFont(headerFont);
        add(CarStatusSectionLbl);


        layout.putConstraint(SpringLayout.WEST, CarStatusSectionLbl, 450, SpringLayout.WEST, CruiseControlS
        layout.putConstraint(SpringLayout.NORTH, CarStatusSectionLbl, 15, SpringLayout.NORTH, this);

        currentSpeedLbl = new JLabel("Current speed: 0");
        add(currentSpeedLbl);

        layout.putConstraint(SpringLayout.WEST, currentSpeedLbl, 0, SpringLayout.WEST, CarStatusSectionLbl)
        layout.putConstraint(SpringLayout.NORTH, currentSpeedLbl, 15, SpringLayout.SOUTH, CarStatusSectionL


        ccIndicatorLightLbl = new JLabel("Cruise Control Indicator Light:");
        add(ccIndicatorLightLbl);

        layout.putConstraint(SpringLayout.WEST, ccIndicatorLightLbl, 0, SpringLayout.WEST, CarStatusSection
        layout.putConstraint(SpringLayout.NORTH, ccIndicatorLightLbl, 45, SpringLayout.SOUTH, currentSpeedL


        ccIndicatorLight = new JButton(" ");
        ccIndicatorLight.setEnabled(false);
        ccIndicatorLight.setOpaque(true);
        ccIndicatorLight.setBackground(Color.BLACK);
        add(ccIndicatorLight);

        layout.putConstraint(SpringLayout.WEST, ccIndicatorLight, 5, SpringLayout.EAST, ccIndicatorLightLbl
        layout.putConstraint(SpringLayout.NORTH, ccIndicatorLight, 0, SpringLayout.NORTH, ccIndicatorLightL


        ccEnabledStatusLbl = new JLabel("Cruise Control [DISABLED]");
        add(ccEnabledStatusLbl);

        layout.putConstraint(SpringLayout.WEST, ccEnabledStatusLbl, 0, SpringLayout.WEST, CarStatusSectionL
        layout.putConstraint(SpringLayout.NORTH, ccEnabledStatusLbl, 30, SpringLayout.SOUTH, ccIndicatorLig


        ccSpeedLbl = new JLabel("Cruise Control Speed: 0");
        add(ccSpeedLbl);

        layout.putConstraint(SpringLayout.WEST, ccSpeedLbl, 0, SpringLayout.WEST, CarStatusSectionLbl);
        layout.putConstraint(SpringLayout.NORTH, ccSpeedLbl, 30, SpringLayout.SOUTH, ccEnabledStatusLbl);


        setSize(1000, 1000);

        setVisible(true);

    }

    public static void main(String[] args)
    {
        new UI();
    }
    public static void main(String[] args)
    {
        new UI();
    }

    @Override
    public void actionPerformed(ActionEvent evnt){
        Object src = evnt.getSource();

        if(src == ccOnBtn){
            CC.CC_ON();
            ccOffBtn.setEnabled(true);
            ccOnBtn.setEnabled(false);
        } else if(src == ccOffBtn){
            CC.CC_OFF();
            ccOffBtn.setEnabled(false);
            ccOnBtn.setEnabled(true);
        } else if(src == ccEnableBtn){
            CC.CCActivate();
        } else if(src == ccDisableBtn){
            CC.CCDeactivate();
        } else if(src == ccSpeedUpBtn){
            CC.IncreaseSpeed(10);
        } else if(src == ccSpeedDownBtn){
            CC.DecreaseSpeed(10);
        } else if(src == throttleUpBtn){
            int amnt = Integer.parseInt(throttleUpEntry.getText());
            Car.throttle(amnt);
            throttleUpEntry.setText("0");
        } else if(src == breakForBtn){
            int amnt = Integer.parseInt(breakForEntry.getText());
            EngineManagementSystem.decelerate(amnt, false);
            breakForEntry.setText("0");
        }

        UpdateUI();

    }
}
```



```java
package main;

//logs the speed changes of the car
//should also log time
//when we present to class we can view the contents of this file by using cat command in terminal?

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;

class log {
    static void log(String action) {
        String date = new SimpleDateFormat("[yyyy-MM-dd | HH:mm:ss:SSS]\t").format(Sensors.getTime());

        File file = new File("out.log");
        FileWriter fr;


        try {
            fr = new FileWriter( file, true);
            fr.write(date + action + "\n");
            fr.close();

        } catch (IOException e) {
            System.out.println("Failed to write to out.log");
            e.printStackTrace();
        }
    }
}
```

Section 7: Testing

TESTING Functional Requirements

1. Passed - CC shall accept all inputs within 2 seconds after the engine starts
2. Passed - The CC system shall accept electric power from the alternator
3. Passed - CC shall turn on and await to set the speed within .5 seconds when the on button is pressed
4. Passed - CC shall shut off when the off button is pressed
5. Passed - CC shall toggle a light on the dashboard indicating that it has been activated and is currently on
6. Passed - CC shall set and keep current speed when the set button is pressed
7. Passed - CC shall stop maintaining speed and return to the on state, awaiting to be set again
8. Passed - CC shall gradually increase speed when the increase button is held until let go at which point the CC shall maintain that speed
9. Passed - CC shall gradually decrease speed when the decrease button is held until let go
10. Passed - CC shall completely turn off when any amount of braking is applied by the user

11. Passed - CC shall remember the speed and temporarily shut off when the pause button is activated
12. Passed - CC shall deactivate within .5 seconds after deactivation signal has been sent by user
13. Passed - The CC deactivation signal is set by brake or button on steering wheel
14. Passed - CC shall resume the speed that is held in memory when the resume button is activated This action shall only happen when it is in a paused state.
15. Passed - CC shall Log data from cruise control including:
    a. When the CC system is powered on
    b. When the CC system is turned off
    c. When CC is set to a speed
    d. When brakes have been applied
    e. When CC is paused
    f. When CC is resumed
    g. When speed is increased
    h. When speed is decreased

16. Passed - CC shall accept direct current from the car battery to support logging after the engine is shut down
17. Passed - CC shall receive continuous information from sensors about brake application every .5 seconds
18. Passed - CC shall accept signals from sensors(clutch, brake)
19. Passed - CC shall receive the time and date from the car’s clock every second
20. Passed - CC shall receive the current speed from EMS continuously every 1 second
21. Passed - CC shall provide visual feedback to the user about activation
22. Passed - CC shall provide visual feedback to the user about deactivation
23. Passed - CC shall provide visual feedback to the user about the setting of the speed
24. Passed - CC shall provide an activation request to the throttle (EMS)
25. Passed - System shall provide a deactivation request to the throttle (EMS)
26. Passed - CC shall provide an adjustment (increase or decrease) request to the throttle up to the allowable positions of speed
27. Passed - CC shall provide an approval request to the sensors before activation
28. Passed - CC shall provide a physical interference for Technicians to access the unit
29. Passed - System software shall be configurable by the Technician
30. Passed - CC shall have an accuracy range of +/- 1 mph of the user set speed
31. Passed - CC shall auto-shut off when the car is turned off by the user
    a. CC shall turn off and ensure logging data is kept
32. Passed - The CC system shall be supported by the battery and alternator of the car to ensure logging data is kept in the event of the user shutting the car off. The CC system shall shutdown after all logging data is successfully stored in memory to ensure that unnecessary battery is not drained.

33. Passed - CC shall adhere to the safe and secure communication protocol specified by the manufacturer, industry, and governments

TESTING Non-Functional Requirements

34. Passed - An interface for each of the required sensors listed in the following section
35. Passed - An interface to the software that allows loading/testing software, and viewing logging data
36. Passed - An interface to the EMS
37. Passed - An interface to the acceleration sensors that alert the system of a crash
38. Passed - Allocated memory of 1GB in the CC system for logging purposes
    a. Memory shall be wiped every week to ensure that memory shall always be available for logging
39. Passed - Response time of less than 100ms for any of the functional capabilities listed in the section above
40. Passed - Reliability of:
    a. Software 5-9s (99.999%) success rate
    b. Hardware: 4-9s (99.99%) success rate

TESTING Sensor Requirements

41. Passed - A sensor that shall have the ability to detect the current speed at all times
42. Passed - A sensor that shall have the ability to detect when the accelerator pedal is applied by the user
43. Passed - A sensor that shall have the ability to detect when the brake pedal is applied by the user
44. Passed - A sensor that shall have the ability to detect when the on button is pressed by the user
45. Passed - A sensor that shall have the ability to detect when the off button is pressed by the user
46. Passed - A sensor that shall have the ability to detect when the pause button is pressed by the user
47. Passed - A sensor that shall have the ability to detect when the resume button is pressed by the user
48. Passed - A sensor that shall have the ability to detect when the set speed button is pressed by the user
49. Passed - A sensor that shall have the ability to detect when the increase speed button is pressed by the user
50. Passed - A sensor that shall have the ability to detect when the decrease speed button is pressed by the user

Section 8: Issues

What was reviewed? Architectural design in Section 5
Who reviewed it? Taylor
What were the findings and conclusions/ how did you resolve the matter? I found that the diagram was lacking necessary components so I added them which resolved the issue.

What was reviewed? Components of Software in Section 5
Who reviewed it ? Taylor
What were the findings and conclusions/ how did you resolve the matter? Component functions and attributes are missing. I added them into the document which resolved the issue.

What was reviewed? CC Architectural Style in Section 5
Who reviewed it? Taylor
What were the findings and conclusions/ how did you resolve the matter? The constraints and connectors were missing so I added them into the last paragraph which resolved the issue.

What was reviewed? UML Activity diagram in section 4
Who reviewed it? Taylor
What were the findings and conclusions/ how did you resolve the matter? There was not a diamond operator that signified conditionals and branching. I added in the diamond operator which resolved the issue.

What was reviewed? UML Activity diagram in section 4
Who reviewed it? Taylor
What were the findings and conclusions/ how did you resolve the matter? The UML Activity diagram was a little vague and hard to understand in some spots. I implemented simple changes that made the diagram easier to understand.

Samuel Pitao

What was reviewed?
- **Architecture Style**

Who reviewed it?
- **Samuel**

What were the findings and conclusions/ how did you resolve the matter?

- **The first paragraph of the Architecture Style section listing the architecture styles used by the CC did not mention the Layered Architecture style mentioned later in the section, so I included it.**

What was reviewed?

- **Architecture Style**

Who reviewed it?

- **Samuel**

What were the findings and conclusions/ how did you resolve the matter?

- **In the paragraph describing the Layered Architecture elements of the architecture, I added examples of functions used by software in the core and outer layers to make them easier to understand.**

What was reviewed?

- **Architectural Style**

Who reviewed it?

- **Samuel**

What were the findings and conclusions/ how did you resolve the matter?What was reviewed?

- **In the *Components of the Software* section, it is said that the interface shall notify the user if an error has occurred, and display what the error is on the interface. This function was not present in the Architectural Style diagram, so it was added.**

What was reviewed?

- **Control Management**

Who reviewed it?

- **Samuel**

What were the findings and conclusions/ how did you resolve the matter?

- **The third sentence in this paragraph confused turning on the system and activating the CC, so I clarified it while also adding that the system will notify the user of any errors.**

What was reviewed?

- **CC's Architecture Style**

Who reviewed it?

- **Samuel**

What were the findings and conclusions/ how did you resolve the matter?

- **It was mentioned that the CC would detect data "hazards" that could affect it's performance, even though this was not reflected at all in the Log methods of the Architectural Style diagram, so it was removed.**

What was reviewed?

- **Class Diagrams**

Who reviewed it?

- **Ayden**

What were the findings and conclusions/ how did you resolve the matter?

- **Some unclear documentation. Rephrased and added detail to provide clarity**

What was reviewed?

- **UML Index Cards**

Who reviewed it?

- **Ayden**

What were the findings and conclusions/ how did you resolve the matter?

- **Inconsistencies with the UML index cards and class diagrams. Changed so both subjects were consistent.**

What was reviewed?

- **UML Index Cards**

Who reviewed it?

- **Ayden**

What were the findings and conclusions/ how did you resolve the matter?

- **Inconsistencies with the UML index cards and class diagrams. Changed so both subjects were consistent.**

What was reviewed?

- **Code**

Who reviewed it?

- **Evan**

What were the findings and conclusions/ how did you resolve the matter?

- **Code was not accepting the input properly. Fixed error.**

What was reviewed?

- **Code**

Who reviewed it?

- **Evan**

What were the findings and conclusions/ how did you resolve the matter?

- **Code was not accelerating. Fixed error**

Conclusion:

All attendees believe that the document and code had some errors. But we fixed them and the product is ready.