Traffic Control System

A comprehensive Software Development Life Cycle report

Student:

Abhilash Reddy Peram (10532966)

Sabitha Maram (10533048)

Srikanth Shilesh Pasam (10387794)

Teacher:

Harnaik Dhoot

Course:

Software Engineering (B9IS119)

Table of Contents

Index of Figures

Index of Tables

Introduction

The project aims to build a Traffic Control System (TCS) for Dublin City Council (DCC). The TCS falls within the remit of Safety-Critical Systems. The project demands a detail-oriented approach.

The team developing the project consists of four developers, of which two are senior developers. One of the senior developers has prior experience working in Aircraft Control Systems, which is a heavily regulated industry very much similar to the TCS. This team is managed by a Project Manager with considerable experience in managing Software Development Projects and also a Scrum Master. The team works for a firm that has over three and a half decades of experience developing software systems using various process models such as Agile, Plan Driven, and Hybrid models. All this valuable experience of the team and the firm will play a critical role in the software development of this Safety-Critical System.

SDLC

The Chaos Report (2015) shows that only 29% of Software Projects completed successfully. The large number of failed projects forms a significant concern for any software development firm or a customer who requires services from the software industry (Hastie, 2015).

In order to be able to deal with such uncertainties, almost all major software development firms adopt SDLC - Software Development Life Cycle.

SDLC is the process of designing, developing, and testing high-quality software. There are four stages in this. Each stage focuses on a very critical element of the journey. Together, this cycle makes sure that the project is completed successfully to the user expectations, within the budget and delivered on time (*SDLC - Overview - Tutorialspoint*, no date).

Process Model

A process model forms the framework of the SDLC. There are various Process Models, each with its advantages and disadvantages. The DCC prefers the Waterfall Model as this model enables greater control by the management. This project makes it easy to track as the project moves from one stage to another. The project is only advanced to the next stage when all the goals of the previous stage have achieved and signed off. The flow is linear; hence, its also called as the 'Linear Sequential Model.'

The software firm has experience working with multiple different process models like Agile, Plan Driven, and Hybrid Model. For the TCS project, the firm will be using the Hybrid Model based on the paper published by Munassar and Govardhan in 2010. This specific Hybrid Model takes advantage of the below list of Process Models and integrates it into one:

- Waterfall Model
- Iteration Model
- V-Shaped Model
- Spiral Model
- Extreme Model

The flow of the Hybrid Model:

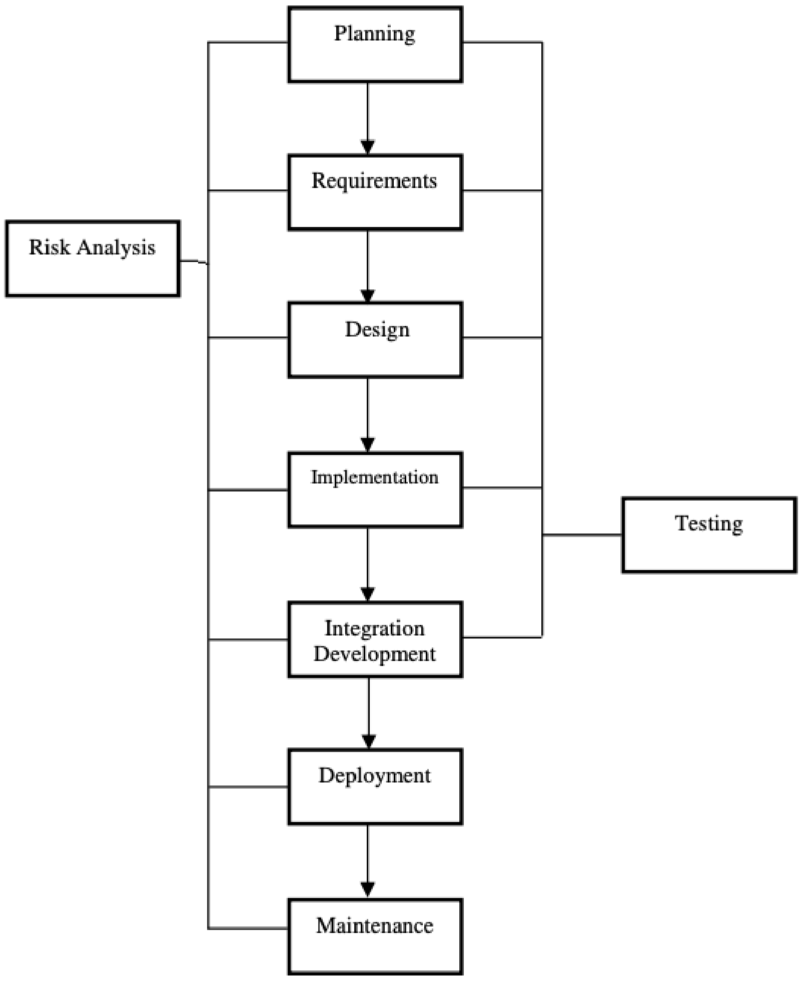

Fig. 1 – Hybrid Process Model (Munassar and Govardhan, 2010)

This model is structured similar to the Waterfall Model in the way it is linear and sequential, with each stage differentiated clearly. Along with the set goals for each stage, there is also emphasis on risk analysis and testing from the first stage itself. The Hybrid model helps address one of the biggest draw-back of the Waterfall model.

This Hybrid Model will give the TCS project the following advantages:

1. Specific milestones for each stage
2. Deliverables for each stage
3. Higher chance of success as the test plans are developed early on in the life cycle
4. High amount of risk analysis
5. Good for safety-critical projects
6. Test-based approach to gathering requirements and ensuring quality

(Munassar and Govardhan, 2010)

These advantages play a crucial role when it comes to designing and developing a safety-critical system such as the TCS. Moreover, as the Hybrid Model is primarily structured around the Waterfall Model itself with the bonus of various other models too like Risk Analysis and Testing phases, it will be easier to convince DCC to come on board with this software development approach.

The TCS project works on the Hybrid Process Model Framework. It has the following six stages:

- Requirements
- Design
- Implementation
- Integration Development
- Deployment
- Maintenance

Requirements

Requirements gathering is the first stage in any Software Development. Capers Jones Study shows the stages where software projects tend to fail. Of this, 12.50% was because of Requirement errors (Jones, 1986). The study shows an alarming number because gathering and analyzing requirements is the very first step towards developing the software. The study shows that 12.50% of the projects were faulty from the very beginning, even before the developers put in their first line of code. The process of gathering requirements is just as necessary, if not even more, than the designing or coding stage.

Stakeholders

Stakeholders include anyone who will be affected by the project or affect the outcome of the project. They could be internal or external stakeholders ('Software Engineering | Stakeholder,' 2019). Stakeholders for the TCS include the government - DCC, pedestrians, emergency services, the software development firm, and the development team – project manager and developers.

Types of Requirement

There are two types of Requirements, Functional and Non-functional Requirements.

Functional Requirements are specifications of the TCS system detailing what it should do in a particular situation like when an emergency vehicle is approaching or on getting specific input like when pedestrians click on the button to cross the road.

Non-functional requirements, on the other hand, talk about how the TCS should behave. It refers to the constraints placed on it like audio feedback on pressing the pedestrian button, the height of the traffic poles, and the brightness of the LED's used.

Requirements Engineering Processes

Four activities are designed to overcome the communication gap between stakeholders and to standardize the process of requirements collection. They are:

- Elicitation
- Analysis
- Validation
- Management

Elicitation

The first activity is the process of requirements elicitation or discovery. The Project Manager, along with a Senior Developer, shall discuss with the customer, DCC, in this case extensively regarding the services the system should provide and its operational constraints. Elicitation is further broken down into:

- Requirements Discovery
- Requirements Classification and Organization
- Requirements Prioritization and Negotiation
- Requirements Specification

The requirements for TCS are gathered and documented through interviews and scenarios (refer Appendix 1) during the Requirements Discovery level. User stories are a form of writing requirement.

User Stories

User Stories are short descriptions of the requirements of the system. User stories form from the perspective of an end-user of the system. Anyone else can also write it. Focus here is on the discussion of the stories itself rather than who has written it. User stories act as a pointer to the real requirements of the system and should ideally connect to some artifact (Cohn, no date).

For the TCS project, the functional and non-functional requirements written in the form of user stories are:

1. As part of the DCC Traffic Control Unit, we want the TCS to manage the traffic automatically and avoid traffic congestions. (Functional)
2. As Traffic Control Operators, we want the TCS also to have manual signal delay adjustments so that we can override during particular circumstances. (Functional)
3. As ambulance drivers, we want the TCS to automatically clear the traffic so that we can reach the destinations quickly and save lives. (Functional)
4. As pedestrians, we want the TCS to enable us to stop the flow of traffic so that we can cross the road safely. (Functional)
5. As pedestrians, we want the traffic signal to have a countdown timer so that we can know beforehand if we have enough time to cross the road. (Functional)
6. As part of the DCC Traffic Control Unit, we want the road sensors and traffic camera data to be stored in Dublin Central Traffic Control Database within DCC's secure datacenter so that it is safe for privacy and security reasons. (Functional)

7. As the Dublin Central Traffic Control Database, we want to use M programing language so that the MUMPS database system can work with the TCS. (Functional)
8. As part of DCC, we want the sensor data timestamped so that it adheres to legal requirements. (Functional)
9. As the DCC Traffic Control Unit, we want the programming language to be Python so that the sensor array of 16K or 32K gets used efficiently. (Functional)
10. As pedestrians with a disability, we want the pedestrian button to be big and low so that it is easily accessible. (Non-Functional)
11. Reliability – As Traffic Control Operators, we want the TCS to be reliable enough so that it does not crash unexpectedly. (Non-Functional)
12. Security – As part of the DCC, we want the TCS to be secure and encrypted so that it is safe from unauthorized access. (Non-Functional)
13. Maintainability – As part of the DCC, we want the TCS to be updated periodically so that the system stays up to date and free of bugs. (Non-Functional)

The next level is to organize these requirements into relevant groups, such as:

- Safety-Critical Systems requirements
- Pedestrian requirements
- Emergency vehicles requirements
- Programming language requirements
- Hardware requirements
- Database management requirement
- Legal requirements
- Infrastructure requirements
- Budget requirements

After organizing, it is followed by prioritizing the requirements like how EVAS will take higher priority than pedestrians or regular traffic and resolving any conflicts.

Finally, in the Requirement Specification stage, the requirements are documented in two categories:

- User Requirements
- System Requirements

User Requirements are those focused on end-users like DCC and the general public. These do not include any technical information. Whereas the System Requirements are for the developers and, as such, contain technical jargon. These are more detailed than the User Requirements.

Requirement Specification

Requirement Specification is the process of documenting user requirements and system requirements. There are two ways of doing this:

- Natural language Specification
- Structured Language Specification

The following are the Functional Requirements of the TCS written in structured language:

	Traffic Control System/Signal Interval Logic/SRS/1.1
Function	Calculate signal change frequency
Description	Calculates the time interval between changing signals for each lane based on the flow of traffic
Inputs	Road sensors and traffic cameras
Processing	Calculate traffic density based on the inputs and process them using the SCOOT algorithm (Skabardonis *et al.*, 2013) to decide the signal change frequency
Outputs	Change in the interval between signal changes
Pre-Condition	The TCS sensor data and video data gets relayed to the processor continuously
Post-Condition	Post analysis, the wait time reduces during low traffic and increases during high traffic

Table 1 – SRS for Signal Interval

	Traffic Control System/EVAS Override/SRS/1.2
Function	Clear traffic for emergency vehicles
Description	The TCS prioritizes approaching emergency vehicles and clears traffic ahead of it
Inputs	Data from EVAS
Processing	Determine the direction an emergency vehicle is approaching and override all the traffic signals along that path to keep the traffic to a minimum
Outputs	Signals change to green in the lane an emergency vehicle is approaching
Pre-Condition	TCS synced with EVAS
Post-Condition	Once the emergency vehicle passes, the TCS operations resume to normal state removing the override

Table 2 – SRS for EVAS Behavior

Traffic Control System/Pedestrian Control/SRS/1.3

Function	Stop traffic to enable pedestrian to cross the road
Description	When a pedestrian presses the traffic button, the TCS will need to determine when to stop the traffic flow so that the pedestrian can cross the road safely
Inputs	Data from the pedestrian button
Processing	On receiving a signal from the pedestrian button, the TCS will calculate based on the traffic density when to change the lane to red and pedestrian to green
Outputs	Lane signal turns red, and the pedestrian signal turns green
Pre-Condition	The EVAS should not have overridden the TCS
Post-Condition	After a fixed time, interval changes the pedestrian signal to red and resume the normal flow of traffic

Table 3 – SRS for Pedestrian Control

The following are the Functional Requirements Specification of the TCS written in natural language:

2.1 The TCS shall store the traffic data from the road sensors and traffic cameras in the MUMPS database by using M language so that it can be stored safely in DCC's secure data center.

2.2 The TCS shall timestamp all the data collected in order to abide by the legal regulations.

3.1 The road sensors shall be programmed in Python so that the array memories between 16K and 32K size gets used efficiently.

Analysis

Requirement analysis is the stage where the software team determines if the stated requirements for the TCS are incomplete, ambiguous, or contradictory. If any identified, then these are clarified before proceeding further (*Requirement Analysis Techniques*, no date).

Validation

Validation forms a crucial part of the Requirements Engineering Process. This stage is used to confirm that the TCS requirements gathered so far and analyzed is what DCC wants. If the project gets delivered without passing through the validation stage, and it contained any requirement errors, then the cost of fixing this maybe 100 times more than fixing any implementation errors (Sommerville, 2016). Validations gets done by checking the following parameters of the TCS:

- Consistency
- Completeness
- Realism
- Verifiability
- Validity

The requirements of the TCS get validated through requirement reviews, which is a systemic manual analysis of the requirements. The reviews will include both the software development team and DCC.

Management

Changes in requirements get accommodated into the process throughout the SDLC. Changes initiated during the early stages of the SDLC are always easier to implement than those who come up towards the end. This point is where Requirements Management plays a crucial role. Having a set of procedures to tackle changes in requirements will make it easier to implement the changes.

The software development team will keep track of all the requirements put forward by DCC and its dependencies. Keeping track will help understand the relationships between the various requirements and processes. In case DCC updates its requirements or modifies any existing ones during the SDLC, then the team will be able to assess the cost estimate and impact quickly.

On the whole, the requirements gathered will help get a clear understanding of the system developed and what is expected out of it before we start to build it.

Design

Once the software requirements specifications document for the TCS is verified with the DCC and approved, the team will move to the design phase. The design stage uses the inputs from the SRS documents to design the software architecture of the TCS. The figure 2 shows the overview of the TCS architecture.

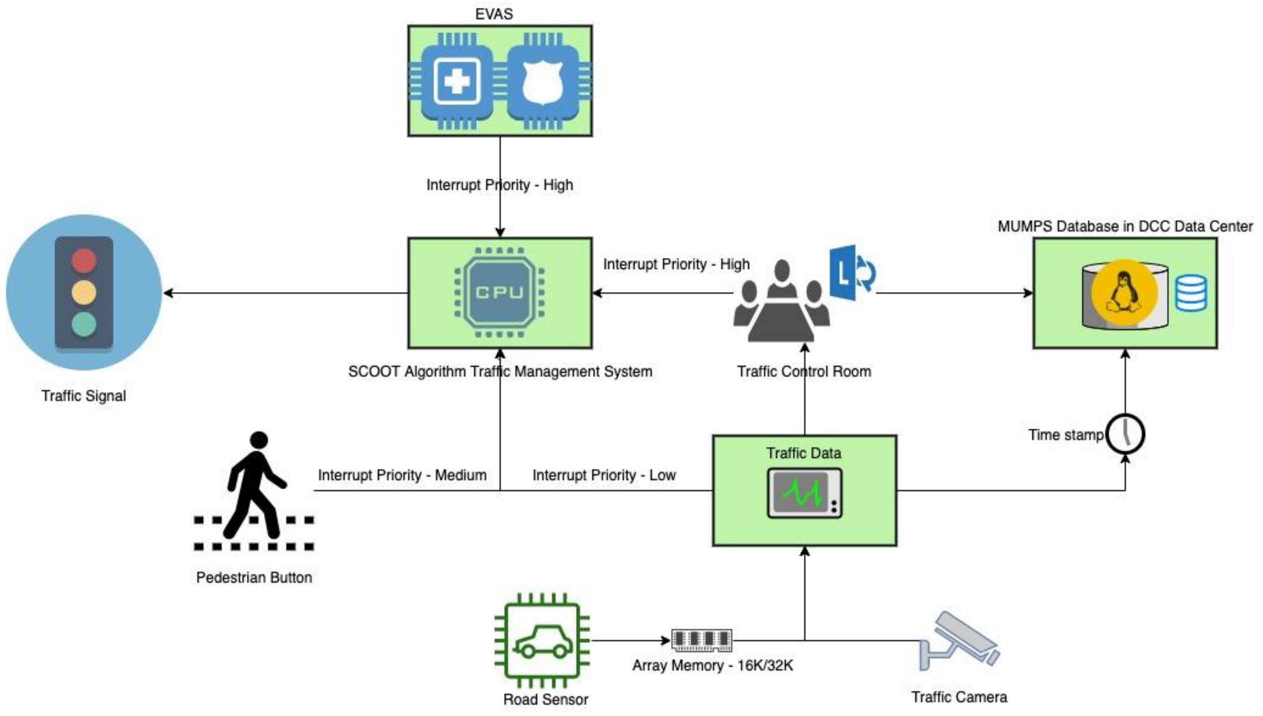
EVAS
Interrupt Priority - High
Traffic Signal
SCOOT Algorithm Traffic Management System
Interrupt Priority - High
Traffic Control Room
MUMPS Database in DCC Data Center
Time stamp
Traffic Data
Pedestrian Button
Interrupt Priority - Medium
Interrupt Priority - Low
Road Sensor
Array Memory - 16K/32K
Traffic Camera

Fig. 2 – TCS Design Architecture

The figure 2 shows a centralized TCS. Data from various sources are processed, and the traffic managed based on the SCOOT algorithm. The entire systems monitored from the Traffic Control Room with live traffic data flowing in. The control room has access to override the TCS if needed manually. They can also access historical traffic data from the database by providing valid credentials. All the traffic data collected is time-stamped and stored within the DCC's Data Center directly from the sensors. This method will avoid any tampering with the data. The EVAS is given the highest interrupt priority so that the TCS prioritizes them before other sources.

UML – Class Diagram

The class diagram is used to represent the attributes or classes of a system. Each class also shows the various methods used within it. The TCS comprises attributes like the 'DCC Traffic Control,' 'Permissions,' 'Routes,' 'Diversions' among others.

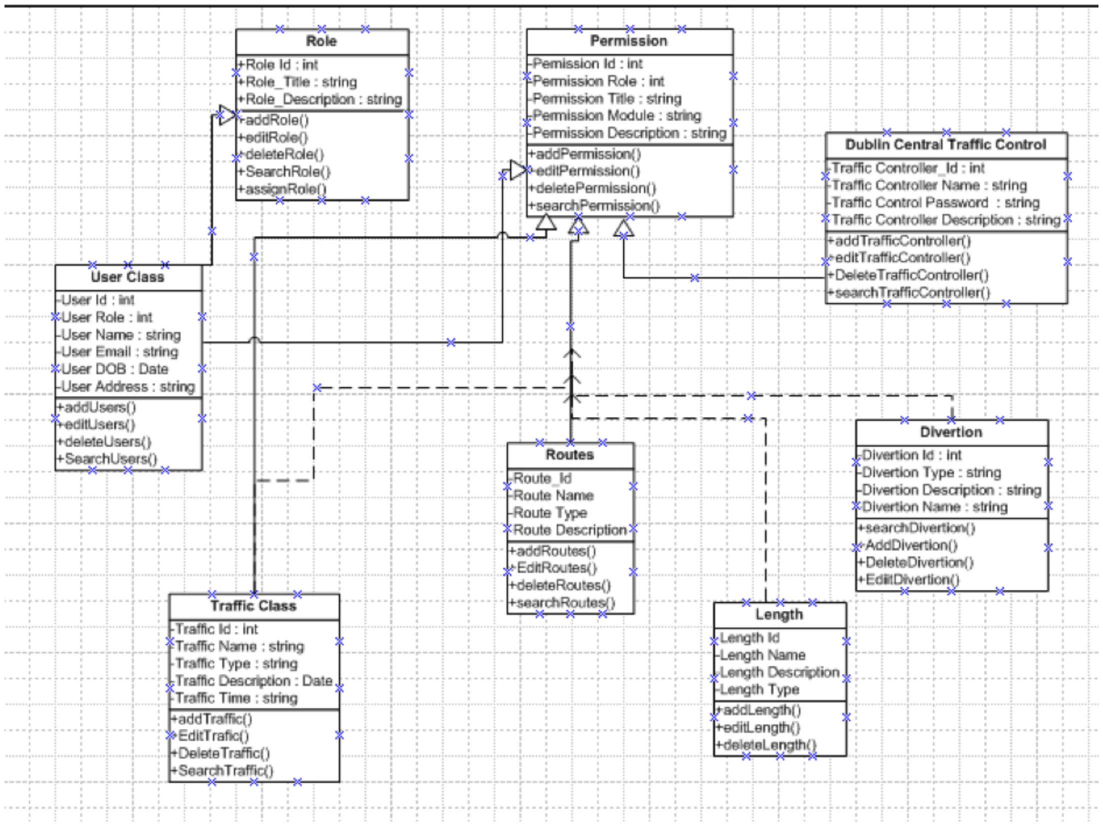

Fig. 3 – Class Diagram (refer Appendix 2 for UML diagram link)

Figure 3 shows all the classes used in the TCS and their associations with each other. The Dublin Central Traffic Control class consists of attributes like the Traffic Controller ID, name, description, and password. This class has methods that enable us to add, edit, delete, or search for a traffic controller in the system. The Permissions class manages access to this class. The Permissions class manages methods that can add, edit, delete, or search for users with permissions access. Each user is given an ID based on their role for managing permissions. Role class takes care of this. A new user is registered into the system by the User class. This class collects the user's information to be stored in the database. All these classes together manage the user end of the TCS.

The next set of classes is used to manage the traffic parameters. These include the Routes, Traffic, Length, and Diversion classes. The Routes class manages all the route maps of the TCS. Traffic class manages the real-time traffic data collected from the sensors and cameras. The Length class manages the duration of the traffic signal depending on the flow of the traffic. Finally, the Diversions class can be used to manage diversions in traffic based on which the flow of the traffic can be adjusted.

UML – Use Case Diagram

The Use Case diagram shows how different types of users can access the TCS. There are primarily two types of users: admin and controller. Each kind of user has its own sets of permissions associated with the system. The admin is the top-level user who can manage everything, including adding new controllers to the system, managing traffic routes, and diversions. The controller, on the other hand, has limited access to the system, like managing signal timings and searching the routes to view the density of the traffic. Its shown in figure 4.

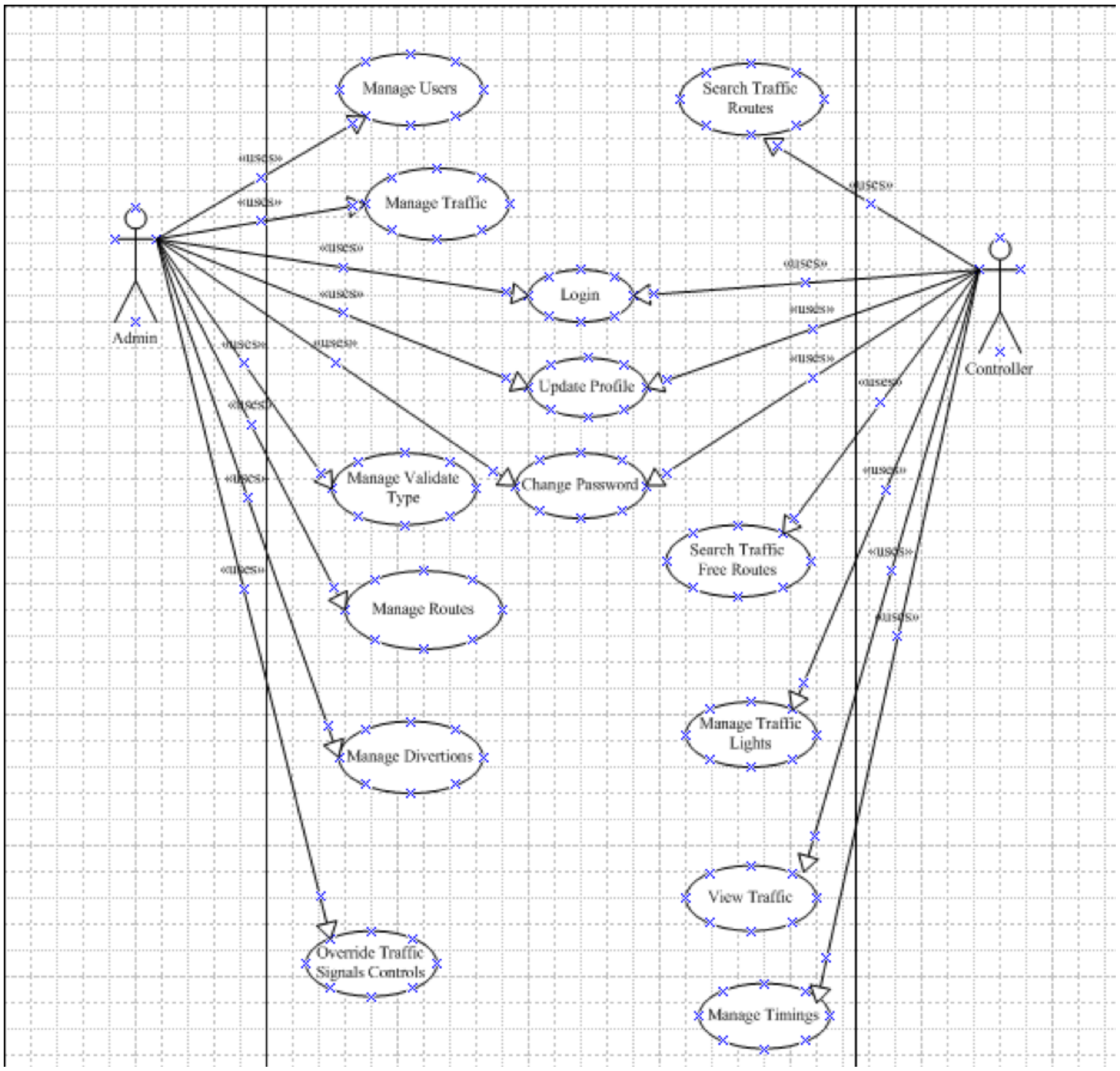

Fig. 4 – Use Case Diagram (refer Appendix 2 for UML diagram link)

UML – Activity Diagram

The Activity diagram shows the flow of various activities within the system. The flow can be sequential, branched, or concurrent. The figure 5 shows the Activity diagram for the TCS.

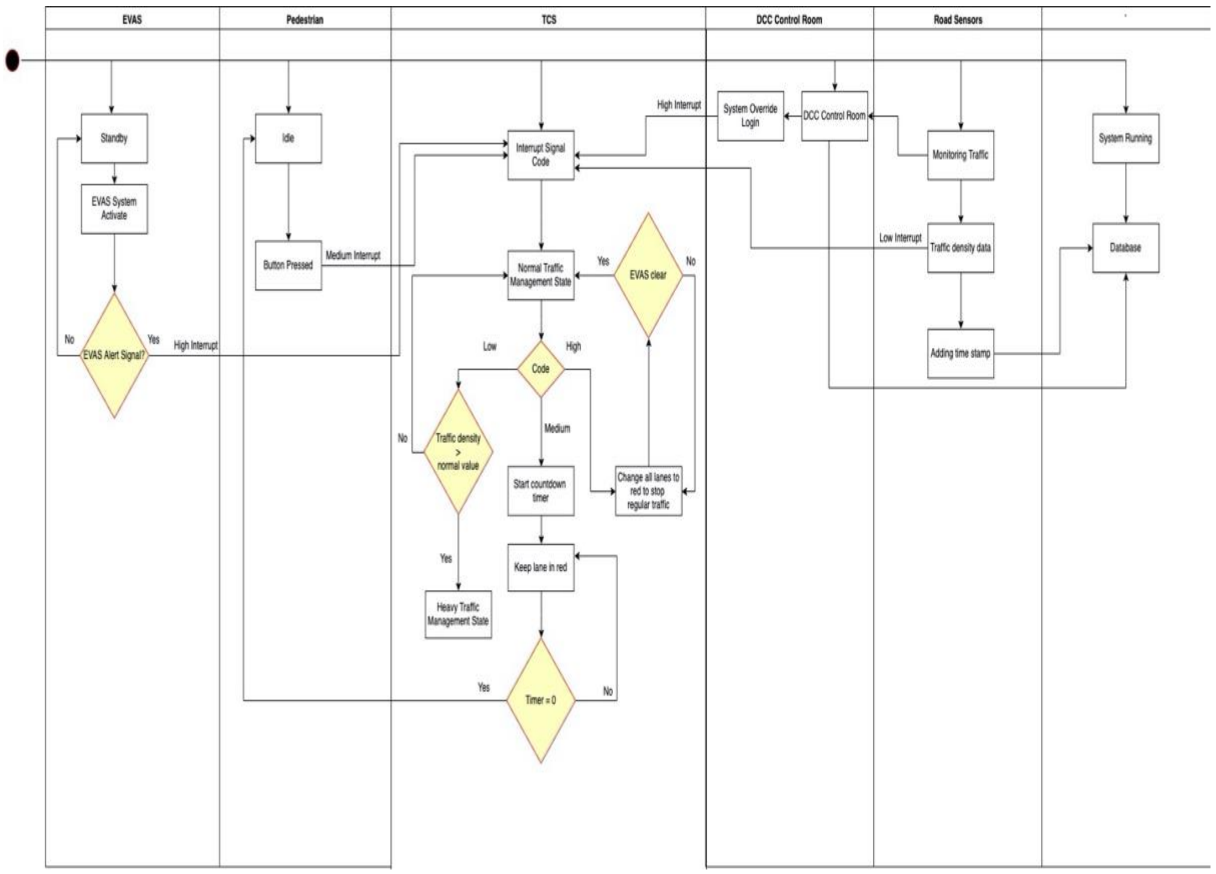

Fig. 5 – Activity Diagram (refer Appendix 2 for UML diagram link)

The activity diagram uses a swim lane approach with all the actors in different lanes. The TCS lane is the central aspect of the entire system. It collects inputs from EVAS, pedestrian control, road sensors, and DCC control room. The TCS system receives three levels of interrupt signal codes. The regular traffic management gets the least priority, which is the default state of the system indicated in the 'Normal Traffic Management State' block. The pedestrian block gets medium priority. Whenever a pedestrian presses the button, a medium interrupt signal gets sent to the TCS. The signal starts the countdown timer for the pedestrians to cross the road. Once the timer expires, the pedestrian system is set back to the idle state. Finally, the EVAS system gets the highest interrupt priority. The interrupt signal is managed by the emergency services directly and can override the TCS. As long as the EVAS alert signal is active, the TCS control gets overridden by the EVAS system. Once the alert gets cleared, the EVAS switches back to standby state allowing the TCS to return to standard functionality.

The DCC Control Room has access to the TCS system directly in case manual control is required. The manual control requires exclusive access and authorizations through a login page. They can access the DCC Datacenter, which stores the timestamped traffic data from the sensors.

UML – Sequence Diagram

The sequence diagram is an interaction diagram that shows the interactive behavior of the system in a sequential order. Figure 6 shows the Sequence Diagram of the TCS.

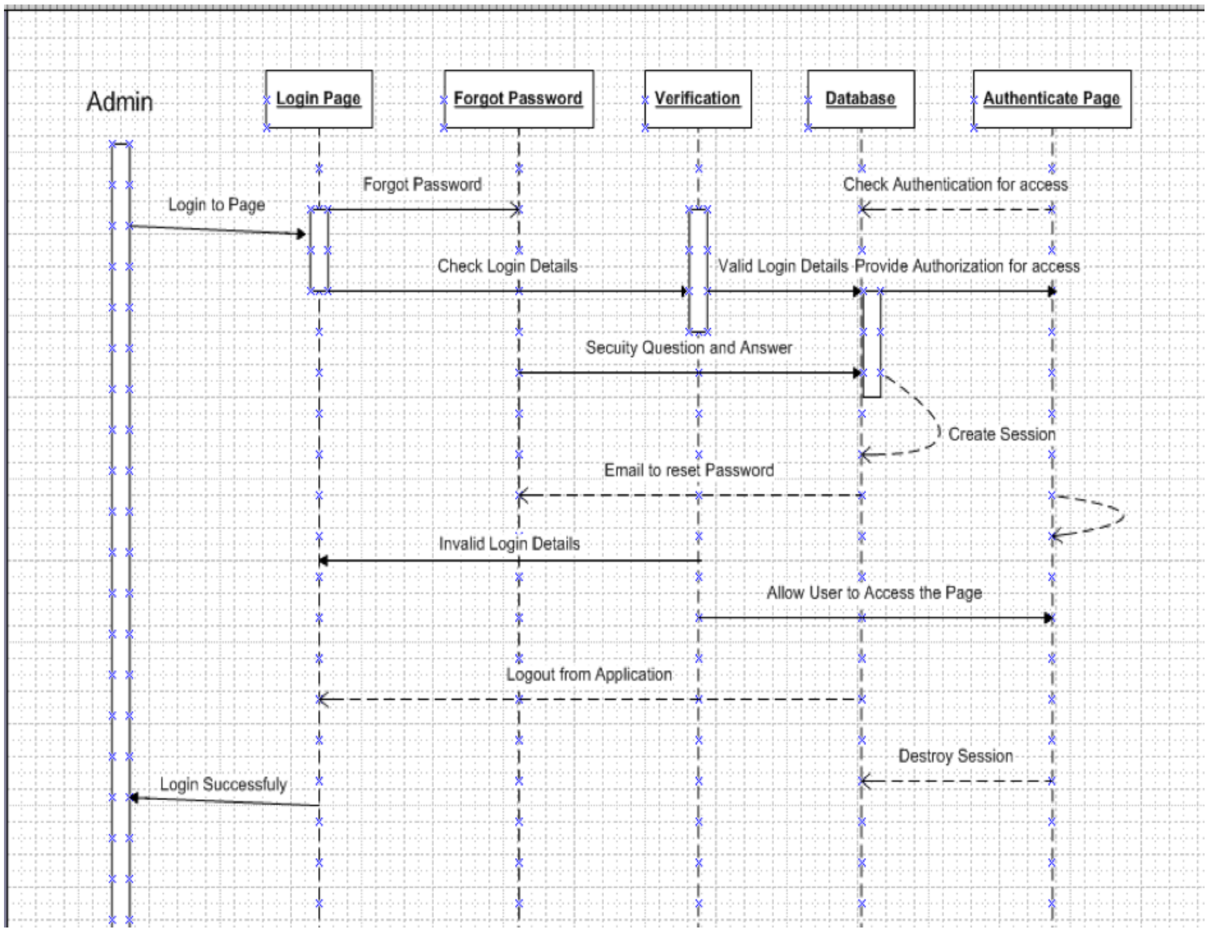

Fig. 6 – Sequence Diagram (refer Appendix 2 for UML diagram link)

The Sequence diagram shows how the Admin accesses the system. The user lands on the login page. On successfully logging in, the admins verified across the DCC database. Based on the admin privileges, a session gets created for the user. In case the user is unable to login to the system, they will get directed to the forgot password page where they will need to answer the security question. On successfully validating the security questions, an email will be sent to reset the password.

Justification

The architecture forms a bridge between the requirements stage and the design/coding stage. It helps transition the project from a concept or idea into something tangible. The architecture plays a crucial role in shaping the outcome of the final product.

The system architecture is the blueprint for the TCS. Figure 2 uses box and line diagrams to show the outline of the system architecture. This design is simple and easy to understand, which comes in handy when explaining to the DCC. This system architecture portrays the logical view of the TCS. It explains how the different components within the TCS are interconnected and highlights the logical flow of signals within the system. The TCS system architecture in figure 2 uses the MVC – Model View Controller pattern for highlighting the flow of signals and data within the system.

The UML diagrams follow the architecture. Figures 3 to 6 are 4 UML diagrams that show the design of the internal system elements. The class diagram lists all the classes and their corresponding methods, which get used in the code.

The design architectures simple so that DCC can find it easy to implement the entire functionalities into the system. The Emergency Vehicle Approach system is used to override the entire control of the traffic system and making it more flexible. The architecture will also be useful in reducing the amount or percentage of the congestion among the vehicles, and thus the problems can be reduced to some extent.

The traffic system also includes the timing control features that are helpful in varying the duration of each signal in the traffic control. The architecture designed supports the feature of varying the time sequence and the phase with which the lights change from one signal color to another. The incorporation of the various sequence with which the lights change gets included in the system design. The design also defines the coordination of the signals and shows how they would operate with each control frequency. These features and the control specifications in the design of the "Traffic Control System" are efficient in handling the traffic flow in urban traffic. Hence, the system design for the TCS is a highly effective and standard process for handling the traffic.

The architecture is designed using necessary features like continuous traffic visualization, which will help in easy detection of the location for the vehicles. The system also provides flexibility in providing the correct signal or the traffic light by analyzing the situation or the condition on hand. This design used will provide accurate results and less complexity in the operations of the system.

Test Cases

A Test Plan is used to test the TCS in various scenarios. These test cases help analyze the performance of the TCS before it gets implemented. The test plan helps provide a glimpse into the functionality of the system in live conditions. Table 4 shows five different test case scenarios for the TCS.

No	Test Case	Scenario	Expected Result	End Result	Status
1	EVAS	The system will detect the approaching emergency vehicle and would instantly change the light to green, or it can show a separate indicator light for the emergency vehicle to pass.	The red traffic signal needs to change to green as the emergency vehicle approaches.	The red signal changed to green.	Pass
2	Signal change	Lanes with higher traffic density are set to green longer while the other lanes are in red for that duration.	The green signal for the most congested path needs to remain On. The signals on the other two opposite lanes need to be red to stop the flow of traffic.	The lights remained red for two opposite lanes and green for the other two opposite lanes.	Pass
3	Timing variation	The time for which the signal remains red or green varies with the level of congestion.	The time for which the light remains green in the busy path needs to be increased and	The green light remained on for the set duration when the flow of the traffic increased and	Pass

			reduced again when the flow reduces.	then subsequently decreased to the minimum duration when the number of vehicles reduced.	
4	Displaying countdown	The countdown will be displayed whenever the signal goes red.	As the red light turns on, the countdown for crossing the road also needs to start.	The light changed from green to red, and the countdown display started.	Pass
5	Displaying Error Message	The user of the TCS will get the error message when the user fails to authenticate.	Error needs to get filed and message to be displayed when the user puts the wrong password.	Error Message displayed when the user entered the wrong password.	Pass
6	Interrupt Priority Test	The TCS will receive all the interrupt signals (EVAS, Pedestrian, and High Traffic) simultaneously.	TCS should prioritize them based on the interrupt levels from high to low.	The TCS prioritized EVAS first. After the vehicle passed, normal working resumed. The next priority was the pedestrian interrupt. Finally, the lanes with high traffic are in green for longer.	Pass

Table 4 – Test Cases

Risk Analysis

Risks	Details	Mitigation
Technical Risks	The system may fail to operate. The system may not be able to communicate with the other field elements. The data transmission may get interrupted. The monitoring system can fail due to server problems. The lights may become inoperative. The timing delay may be wrongly programmed, which can lead to false phase detection.	The admin needs to check the system periodically to identify any faults. The admin should also incorporate proper maintenance of the system. Cloud computing encryption features should be included to enable a safer communication system within the field elements. The lights and the electrical system should get periodically checked. The programming should be checked and tested before deployment to avoid this.
Security Risks	The TCS could get hacked due to security breaches.	Ensuring the credentials of the users monitored closely.
Operational Risks	Operational risk may arise when there is involvement of multiple parties handling TCS. The operator may not be trained to handle TCS.	The operation of the TCS should be centralized to avoid any conflicts. The operator should get trained for the correct implementation and operation of the TCS.

Table 5 – Risk Analysis

First Release Plan

The planning for the project launch will include taking into account the pre-requisites for the task. These include sharing information with the stakeholders of the project. The resources that are available for the project need to get identified when including data and charts.

The first Release plan for the software aims at addressing the project objectives and checking whether the system can perform its designated functional and non-functional tasks. The steps that will be taken for this review is as provided below:

Product Vision: To demonstrate the working of TCS and its corresponding functional and non-functional requirements.

Release Schedule Review: The schedule for the release will be decided based on the outcomes of the test case scenarios and risk case analysis by all the stakeholders.

Plan Schedule: The different milestones within each stage of the project get identified, and the corresponding data presented regarding its functionality progress. Based on this data, a schedule gets planned for the next stages.

Map the expected User story scenarios to Test Results: From the user stories collected so far, the functional and non-functional requirements get mapped to the results obtained for ease of tracking.

Communication Plan: The communication plan will help maintain a clear and consistent channel between all the stakeholders during the development of the TCS. The communication medium will enable the DCC to provide the team with any changes in their requirements, if any necessary. These get implemented before releasing the final product. The first release should have a log of all these communications with their status in the project.

Test Case: The functionality of the TCS gets demonstrated to the DCC through various test case scenarios. The scenarios will try to address the functional and non-functional requirements.

Re-Release: Schedule for future releases, if any.

Code

The TCS code enables automatic traffic control based on the traffic density of individual lanes. Data from traffic sensors continuously monitor the rate of traffic flow per second and use it in the code to alter the traffic signal timings accordingly.

The highest priority is given to the EVAS system, followed by pedestrians. When EVAS turns 'True,' the entire TCS is overridden to prioritize access to emergency vehicles.

Due to deployment limitations and unavailability of live sensor data, we have assumed random data and user inputs for testing (refer Appendix 2 and Appendix 3).

Conclusion

Once the TCS project achieves all the milestones for each of the stages post the first release, it will be time to deploy the project on a large scale. As the infrastructure is being laid out by the DCC, the team will be working on setting up the DCC Traffic Control center and configuring their systems. The User Manual of the system gets handed over once all this is complete. The manual will contain every single detail of the TCS, UML diagrams, EVAS integration, troubleshooting scenarios, hardware specification, and configuration details.

Appendix 1

1) What is the project?

   To build a Traffic Control System, TCS, for Dublin City Council, DCC.

2) What does DCC expect of the TCS?

   The system must access and use data from road sensors and traffic cameras that indicate the flow of vehicular traffic and the extent of congestion in order to determine optimal light changes alongside pedestrian users' requirements too.

3) What are the specifications of the road sensors?

   The sensor arrays have between 16K and 32K of memory.

4) Is there any requirement for the programming language used?

   The system should use either C or C++ as the programming language.

5) Where is this data stored?

   All sensor data and video data must be collected and stored in the Dublin Central Traffic Control Database, located in DCC's secure data center.

6) What is the database used to store this data?

   The data must get stored in the MUMPS database system. The DB should be written in the M language, hosted on Linux, and timestamped for legal reasons.

7) Are there any other specific requirements expected of the TCS?

   The TCS should come within the remit of safety-critical systems. The system should also interface with the Emergency Vehicle Approach System, EVAS.

8) How exactly does DCC see this working? Provide a scenario.

   When an emergency vehicle is approaching, then the TCS should override traffic signal controls to prevent traffic flow ahead of the approaching emergency vehicles (fire engines, ambulance, police, and security forces). The EVAS signal change requests require priority processing ahead of the pedestrian button presses.

9) Does DCC have any preferences in-process model?

   Yes. Waterfall software development model.

Appendix 2

GitHub repository link for UML diagrams, Code and Report:

https://github.com/srikanthshileshpasam/SE_B9IS119_CA1

Appendix 3

```
#!/usr/bin/env python
# coding: utf-8

# In[119]:


import random
import time

#Using boolean to test different scenarios. On deployment, these values
will be taken from pedestrian sensors and EVAS respectively
pedestrian_click = False
emergency = False

#This class is used to collect data from external sources which will be
used by the TCS
class TrafficData:
    def __init__(self):
        pass

    #Data from road sensors and traffic cameras
    def road_sensor():
        traffic_lane = []
        #Random generator for dummy traffic data
        for k in range(0,4):
            traffic_lane.append(random.randint(1,10))
        return traffic_lane

    #Data from EVAS. Currently using user input. On deployment will be
using data from the EVAS system
    def evas():
        ev = 5
        while ev <=0 or ev > 4:
            ev = int(input('Which lane is the emergency vehicle
approaching on? Choose between 1 to 4.\n'))
        return ev

    #Data from pedestrian button. Currently using user input. on
deployment will be using data from the pedestrian button
    def pedestrian():
        ped = 5
        while ped <=0 or ped > 4:
            ped = int(input('Which lane is the pedestrian crossing? Choose
between 1 to 4.\n'))
        return ped

#This class uses data from the TrafficData() class to manage traffic
class TrafficControl:
    def __init__(self, traffic_flow, pedestrian_alert=0):
        self.traffic_flow = traffic_flow
        self.pedestrian_alert = pedestrian_alert
    #Logic during regular traffic flow and pedestrian scenarios
    def normal_traffic(self):
        #Loop to cycle between the 4 lanes
        for j in range(0,4):
            signal = ['Red', 'Red', 'Red', 'Red']
            #Time for pedestrians to cross the road
            timer_ped = 3

            #Logic for calculating green signal interval for each lane
based on traffic density of those respective lanes
            timer_lane = self.traffic_flow[j] + 1

            #Checking to see if pedestrians need to cross the lane.
            if j != (self.pedestrian_alert - 1):
                signal[j] = 'Green'
                print(f'\nLane {j+1} changes to green while other lanes
are in red\n{signal}\n')

                #Timer for traffic signal
                for i in range(0, timer_lane):
                    print(f'Green for {timer_lane - i} seconds')
                    time.sleep(1)
            #In case pedestrian needs to cross the lane
            else:
                print(f'\nPedestrian crossing lane
{self.pedestrian_alert}\n')
                #Timer for pedestrian crossing
                for i in range(0, timer_ped):
                    print(f'Pedestrian green for {timer_ped - i} seconds')
                    time.sleep(1)

    #Logic during EVAS scenarios
    def evas_alert(self, incoming_lane):
        self.incoming_lane = incoming_lane
        signal = ['Red', 'Red', 'Red', 'Red']
        signal[incoming_lane-1] = 'Green'
        return signal


traffic_data = TrafficData
road_sensor_data = traffic_data.road_sensor()

#For displaying the traffic density data and status of the system
for k in range(0,4):
        print(f'Traffic density in lane {k+1} is: {road_sensor_data[k]}
vehicles/sec.')
print(f'\nEmergency status: {emergency}\nPedestrian status:
{pedestrian_click}\n')

#Highest priority given to EVAS system by checking it first
if emergency == True:
    #Emergency vehicle operation mode:
    emergency_lane = traffic_data.evas()
    traffic = tcs.evas_alert(emergency_lane)
    print(f'\nAlert!\nEVAS override of traffic signals!\nChanging all
lanes to RED except for oncoming emergency vehicle in lane
{emergency_lane}\n{traffic}')
```

```
#Second highest priority given to pedestrians
elif pedestrian_click == True:
    #Pedestrian operation mode:
    pedestrian_lane = traffic_data.pedestrian()
    tcs = TrafficControl(road_sensor_data, pedestrian_lane)
    tcs.normal_traffic()

#Lowest priority for regular traffic flow
else:
    #Normal traffic operation mode:
    tcs = TrafficControl(road_sensor_data)
    tcs.normal_traffic()
```

Bibliography

Cohn, M. (no date) *User Stories and User Story Examples by Mike Cohn*, *Mountain Goat Software*. Available at: https://www.mountaingoatsoftware.com/agile/user-stories (Accessed: 22 November 2019).

Hastie, S. (2015) *Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch*, *InfoQ*. Available at: https://www.infoq.com/articles/standish-chaos-2015/ (Accessed: 22 November 2019).

Jones, C. (1986) *Programming Productivity*. McGraw-Hill.

Munassar, N. M. A. and Govardhan, A. (2010) 'HYBRID MODEL FOR SOFTWARE DEVELOPMENT PROCESSES', in.

Requirement Analysis Techniques (no date). Available at: https://www.visual-paradigm.com/guide/requirements-gathering/requirement-analysis-techniques/ (Accessed: 22 November 2019).

SDLC - Overview - Tutorialspoint (no date). Available at: https://www.tutorialspoint.com/sdlc/sdlc_overview.htm (Accessed: 22 November 2019).

Skabardonis, A. *et al.* (2013) *Advanced Traffic Signal Control Algorithms.pdf*. California PATH Research Report UCB-ITS-PRR-2014-04, p. 123. Available at: https://merritt.cdlib.org/d/ark:%252F13030%252Fm5nc7fc1/2/producer%252F884613548.pdf (Accessed: 24 November 2019).

'Software Engineering | Stakeholder' (2019) *GeeksforGeeks*, 12 March. Available at https://www.geeksforgeeks.org/software-engineering-stakeholder/ (Accessed: 22 November 2019).

Sommerville, I. (2016) *Software Engineering, Global Edition*. Pearson Higher Ed.