Software Design Specification: “Unstable Bluff” Detection System

By Maron Vincent Ejanda, Janelle Kwofie, Chanine Malong, and Isaac Pompa

Unstable Bluff Detection System Requirement Specification

Prepared by: Maron Vincent Ejanda, Janelle Kwofie, Chanine Malong, and Isaac Pompa

October 15th, 2021

1 Introduction and Overview

The following will be used to describe The Unstable Bluff Detection System. The Unstable Bluff Detection System's purpose will be to detect bluffs in the Del Mar Area for their stability. Unstable bluffs are known to cause damages such as causing rocks to fall on cars, trains, or even people hence why this system running well and timely is so fundamental. Thus the main purpose of this system is safety. The system will work by tracking tourists in a 300-foot range as well as the bluffs. When the bluffs are recorded to been unstable authorities will be alerted. Ultimately, this system will blend in with the environment and will not disturb the tourists or tracks in the area of the bluffs.

2 User Requirements

The system's main functions will be executed through the cameras. In the 300-foot bluff zone, there will be cameras standing on metal pools capturing 50 by 50-foot partial images of the bluff. These 32-bit, timestamped images will then be sent in hourly increments to the off-sight local facility, the main user of this system. The Unstable Bluff Detection System should then allow the facility to send alerts and alarms to operators and lifeguards in the area when the bluffs hit level 4 or 5. These operators and lifeguards are the only other users of this system therefore, they should be able to receive these messages and alarms.

3 System Requirements

3.1 Functional Requirements

Now knowing the overview and the main users of this system, these functional requirements will list what this system needs to do. This system will monitor about 300-foot rangers of the bluffs by using 6 cameras placed onto poles along the area. These cameras will be able to capture a 50 50 foot area to fully capture the total 300 feet. The image captured will be sent to a local facility hourly where the images will be stored with time and level data for the bluff. This system also has a severity rating where 0 is safe, 1-3 is considered a potential rockslide, 4-5 is a significant change where alarms go off and the Amtrak rail operators and lifeguards are alerted to block the roads and evacuate the people. The facility will then check the footage to see if any people were left behind. In normal conditions, it is imperative that the system is still sending these images to the facility so they can be accessed and examined. The bluff must be monitored 24/7 by the system showing accurate readings of the levels as well as sending out alerts only when necessary.

Use Cases:

In a normal use case, the cameras and trackers record 24/7. If an unstable bluff of Level 4 or 5 occurs it will offset the alert and alert the authorities. Or in the case of no bluffs, no response will be given if it never hits level 4. If in the case the alarm goes off when it is unnecessary the authorities will confirm a false alarm and the system will be revised for the magnitude that alerts a level 4. In the cases, the camera fails the local facility will be alerted. Overall, in the case of false alarms, it is more important to be overly safe than sorry, and have the authorities confirm the safety of the bluff first.

3.2 Non-functional Requirements

The Cameras:

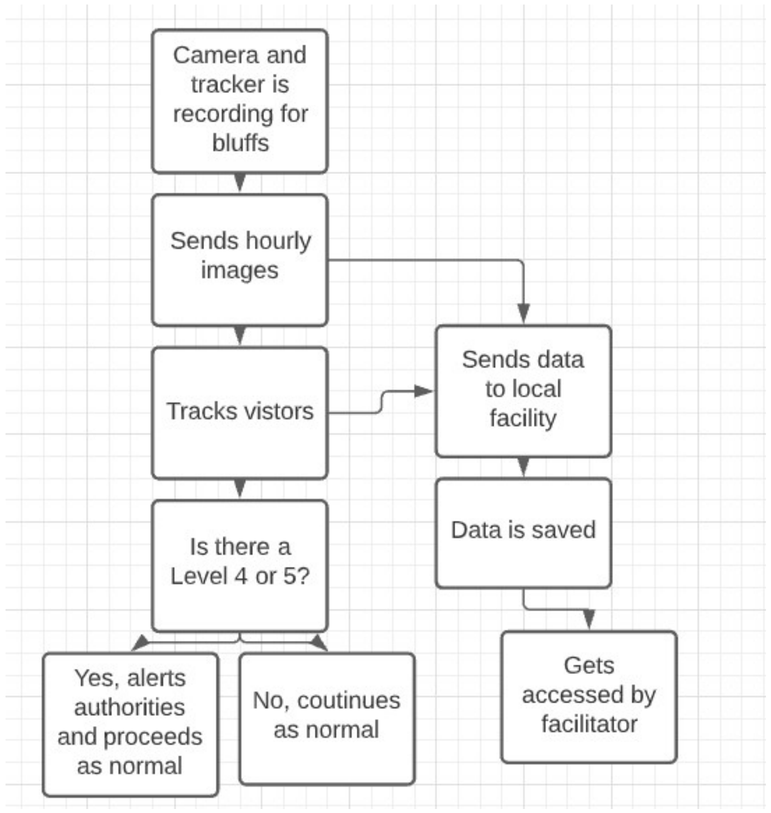
- The Cameras need to be 24/7 and hourly move 32 bit and time stamps will be sent from the cameras to an off-sight local facility.
- Cameras will sit on polls.
- These images need to be 4000 pixels per square bit and 8 bits per pixel. The images should be available in color and greyscale. Additionally, the bluff along with the edge and 300 ft of the surrounding area should be pictured.

Off Sight Facility:

- Needs to be an off-sight facility with lots of memory on hand to save all the information.

Alert:

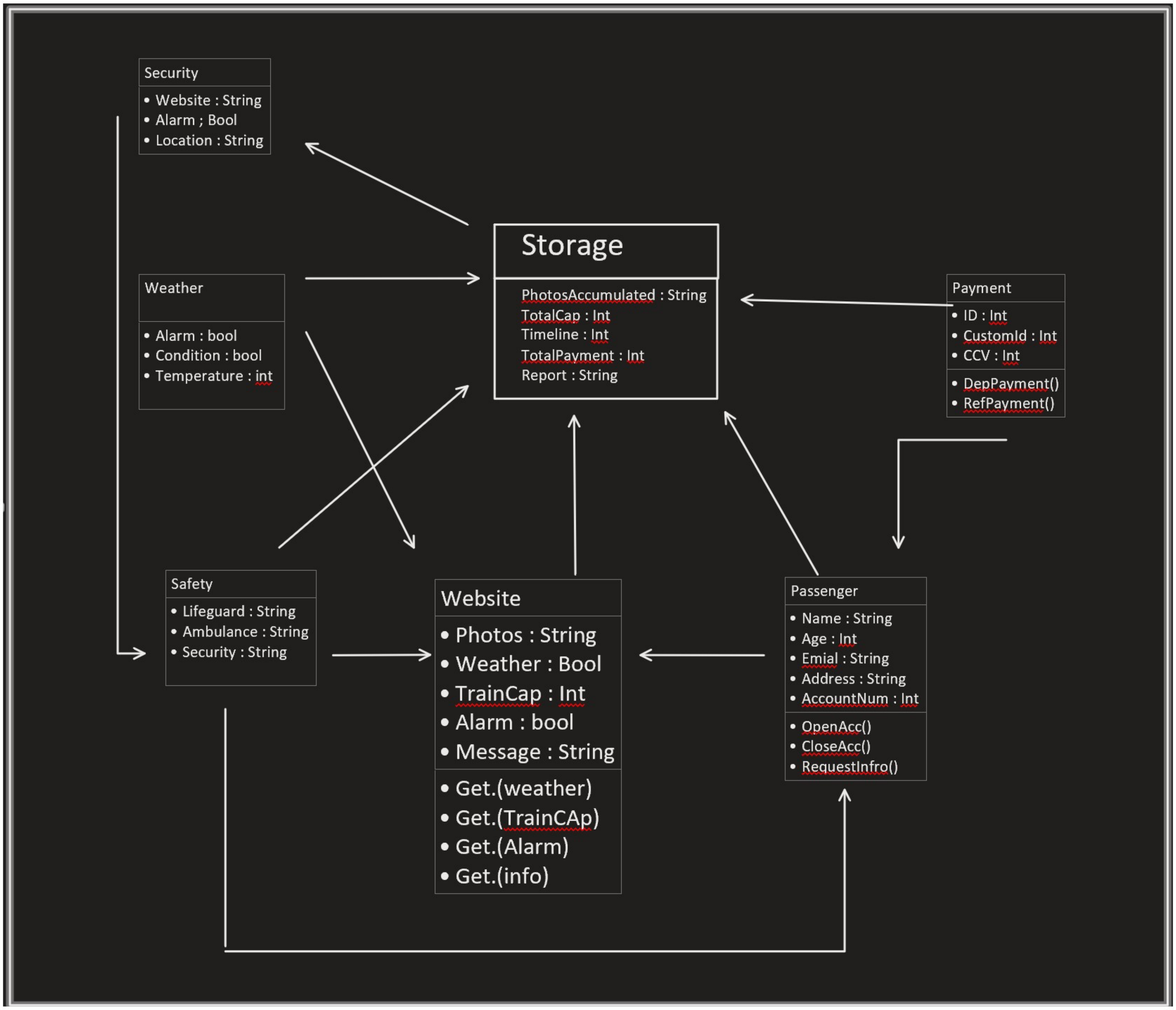
- Should send fast and correct alerts. ● Needs to be 24/7.


Camera and tracker is recording for bluffs
Sends hourly images
Tracks vistors
Is there a Level 4 or 5?
Yes, alerts authorities and proceeds as normal
No, coutinues as normal
Sends data to local facility
Data is saved
Gets accessed by facilitator

4 Other

This section (or following subsections) contains any relevant information not covered in any previous section. Examples include risks, constraints, assumptions (not already identified), and potential future changes.

- Alerts need to be correct and sent to everyone in the area, not just safety personal
- Images need to be adaptable to time of day, i.e. night mode versus day mode.
- Needs to not disturb visitors, tracks, or safety measures.
- Should ensure that alerts have the right information to set off for each level.


Security
• Website : String
• Alarm ; Bool
• Location : String
Storage
PhotosAccumulated : String
TotalCap : Int
Timeline : Int
TotalPayment : Int
Report : String
Weather
• Alarm : bool
• Condition : bool
• Temperature : int
Payment
• ID : Int
• CustomId : Int
• CCV : Int
• DepPayment()
• RefPayment()
Safety
• Lifeguard : String
• Ambulance : String
• Security : String
Website
• Photos : String
• Weather : Bool
• TrainCap : Int
• Alarm : bool
• Message : String
• Get.(weather)
• Get.(TrainCAp)
• Get.(Alarm)
• Get.(info)
Passenger
• Name : String
• Age : Int
• Emial : String
• Address : String
• AccountNum : Int
• OpenAcc()
• CloseAcc()
• RequestInfro()

Software Architecture Description

- The website class will contain the following attributes:
  - Photos of the bluffs for the system to catch any differences
  - Weather report because it can cause rockslides
  - Train capacity lets the operators know how many people there should be if an evacuation were to happen
  - The alarm alarms the lifeguards and operators by sending a message All of these attributes can be accessed by using the get() method for each.

- The weather class has the temperature and condition and will alarm if the condition is bad.

- In the passenger class, the attributes include:
  - All personal information such as name, age, email, address, and account number
  - Allows them to open an account by requesting the user's personal information - Can also close the account if they want

- After creating an account, they can purchase train tickets by providing their payment information in the payment class.
  - User must input their ID, CustomID, and CCV number of card - Can pay for a ticket or get a refund

- Under the security class, the attributes would be:
  - The website, alarm, and location of which area within the 300 feet bluffs have been affected
  - This will then be passed to the safety class

- The safety class is how the lifeguards, ambulance, and security will be alerted and is sent the location where the rocks are unstable. These people can access the main website and passenger data for safety.

- Meanwhile, all the data from the weather, safety, website, passenger, and payment classes are stored in the storage class.
  - Photos accumulated and timeline will be saved up in the storage for one year - Total capacity saves the total capacity of each train - The report is where the rating system occurs.
  - 0 = no noticeable changes
  - 1-3 = subtle changes, but not dangerous
  - 4-5 = significant changes and information is sent to the security and safety classes to alert the lifeguards and operators

Development Plan and Timeline:

As a team we would take 6 months to complete our software system. Nelle would be in charge of implementing the website class where it contains photos of bluffs, weather reports, train capacity, and the alarm that alerts the operators. Nelle would also be responsible for the weather class which includes temperatures, conditions, and the alarm that turns on if the condition is bad. Chanine will be in control of fulfilling the passenger and payment class. The passenger class has the passenger's personal information and permits a passenger to open and close an account by asking the user's personal information. The payment class would allow the user to input their debit/credit card information to purchase a ticket or request a refund. Vincent would be in charge of creating the security and safety class. The security class will have the location of the bluff that has been affected within 300 feet, the website, and alarms. The safety class will allow lifeguards, ambulance, and security to view the location of the unstable rocks. The safety class will also allow the lifeguards, ambulance, and security to access the passengers data. Finally, Isaac will be responsible for implementing the storage class which will let the system store all the classes information. For example, photos accumulated and timeline will be saved up in the storage for one year, the total capacity of each train, and reports of the rating system.

Passenger Functional Testing

Variables:
String: Name (first and last)
Integer: Age
String: EmailAddress
String: HomeAddress
Integer: AccountNum

Functions:
openAcc() will allow passengers to open an account on the website by calling the requestInfo() function. In this function, it will first prompt the user to enter their personal information and will set the user input to each of the variables (name, age, email, home, account number). They will also have the option to close their account through the closeAcc() function that would not need any information to be entered. When this is being tested the tester will only see the prompted messages and their inputs. If everything is successful the output will be a message and if it's not another message will say that it was unable to create an account.

```
public class Passenger {
        openAcc(); //Passenger class will call openAcc to open
        closeAcc(); //Passenger class will call closeAcc to close
}

public boolean openAcc(){
        requestInfo(); //calls requestInfo() to get all personal information
        if( //required info is entered correctly) {
                println("Your account has successfully opened!");
                //Shows the user that their account is opened
                }
        else(){
                println("Something went wrong! Please fill in all required fields.");
                //Shows the user that there is a problem opening their account
                }

public boolean requestInfo(){
        println("To open an account, please enter information in all required fields below: ");
        //Then will prompt each field for name, age, etc. and take the user input and use set()
        methods to store the information entered in the storage system.
        //All information for the will also be accessible to the website for the users themselves as
        well as the train operators for safety.
}
```

//When closeAcc() is called, the account under that email will be deleted from storage and will be prompted that the account has been deleted.

Payment Test Plan

Variables:
String: ID (Users Full Name)
Integer: cardNum (User Credit Card Number)
Integer: cardExp (User Credit Card Expiration)
Integer: CCV (User Card Code Verification)
Boolean: Transaction

Functions:
After the user creates an account they will be able to purchase train tickets through DepPayment() by calling the requestInfo() function. In the DepPayment() function it will allow the user to input their credit card information and will set the users input to each variable (full name, credit card number, credit card expiration, card code verification). The user will also have an option to receive a refund through the RefPayment(). In order for the user to receive a refund the system will go through storage(get.Info) to obtain the information needed to refund the user.

The expected output for this test will look like:

```
Payment() {

        DepPayment();//DepPayment function will call the requestInfo()
        RefPayment();// RefPayment function will call get.(Info) in storage
}

Public Boolean Transaction(String ID, int cardNum, int cardExp, int CCV) {
        DepPayment();// calls requestInfo() to get account information
        println("Please enter your card information to purchase ticket:")
        If(//required payment information is correct) {
        println("Payment Successful")
        return true;
        }
        else() {
        println("Payment Declined"
        return false;
        }
}
```

Name: "Isaac Pompa"
Card Number: 4400 3900 4000 1000
Card Expiration: 12/25
CCV: 500

The system will then return true if Transaction is successful if not it will return false, declining the payment. The system will also store the total purchase in storage under int TotalPayment; , To keep track of user(s) recent purchases. This would be helpful if the user wants a refund, we would pull up the user purchase from storage and refund the amount to the credit card used.

If DepPayment() returns true the system will print “Transaction Successful”
If Depayment() returns false the system will print “ Transaction Denied”

```
RefPayment() {

        get.(Info)//get.(Info) will get called to obtain the users past purchase through storage

}
```

When the user requests a refund to the system it will retrieve the users information by going

through storage(get.Info) to obtain the users credit card information to be able to refund the user.

Test-Plan(Weather):

Variables
Bool Alarm;
Bool Condition;
Int temperature;

Functionality:

- Alarm – The weather will contain the trigger on the safety alarm because it has the power to make the bluff unstable
    - This will go off if there is a heaver rain, thick fog, or high wind in by the train bluffs.
    - Alarm output will give the signal to the safety, will undergo to the security and will be posted on the website.
- Condition – This will give what kind of condition does the bluffs has
    - Condition such as the unstable bluff is visible and unsafe to be in it.
    - This will test and see if the train can go pass through the bluffs.
- Temperature – this will output a numerical temperature on the location of the bluffs.
    - Example: if temperature < 67F. meaning it has a cool temperature
    - If Temperature > 68. Meaning it is warm
    - Depending on the temperature, this can predict what kind of weather will be present in the day

Weather As a Whole:

- The weather will determine pre-condition of the bluff
- This has the power to trigger the alarm on the safety function
- All the data gathered and got on this function will be passed down to the different function to get checked and to be posted on the website and to be stored in the storage.

Pseudo Code for Weather Class

```
public class weather{
        //will get temperature from external local weather sites from a URL and making sure that
        //the wanted location and time is initialized and declared so that it will always show the
        //needed temperature correctly.
        int temperature;

        public void getTemp(){
                return temperature; //will return temperature when called.
        }

        public void setTemp(int t){
                temperature = t; //sets the temperature from website (int t) to int temperature.
        }

        public void condition(){
                getTemp(); //calls getTemp() to get the current temperature.
                if (temperature > 68) {
                        system.out.println( “Warm Forecast”);
                        //if the weather is over 68 degrees it’s warm and nothing to worry about.
                        }

                else() {
                        system.out.println( “Cold Forecast, be cautious”);
        //else, its cold and it might cause he bluffs to become unstable, especially if it rains.
                        }
        }
}
```

Test-plan (Website):

Variables:

String Photos;	Get.photos;
String Message;	Get.weather;
Bool Weather;	Get.trainCapacity;
Int trainCapacity;	Get.alarm;
Bool Alarm;	Get.information;

Functionality:

- The website will get photos that is stored in the storage using:
    - Get.photos – there will be a function on the website to grab the photo from the storage and that will be posted in the website.
- A message will be attached to the Photo taken from the storage.
    - This message will be made by the website developer.
    - This string of message will explain the current information on the bluffs taken from the picture. This will also give the safety information to the passengers.
- The train capacity will have 90 passenger per vehicle
    - The train capacity will contain the number of passengers in each vehicle and will be posted on the website beside the information given to the passenger.
    - Get.trainCapacity – the function will grab the capacity from the passenger function and will output the capacity in information of the website.
- Safety on the website – will give the information about the safety precautions and the alarm information
    - Get.alarm – this will grab the alarm "true or false" if it is safe to ride in the train. This information will be the first message on the website.

Website as a whole:

- The website will grab information on the storage mainly to post the photos
- Photos, messages, payment, capacity, and alarm will be posted.
- The website will check the other functions recursively to continuously give the information to the upcoming passengers.
- Website will be the main user interface to the people and these tests will mainly come from the other function which grabs the whole information.

Pseudo for Website:

- get.photo()
    - Returns photo path so it can be accessed
        - Example
            - Bluff A’s photo = “C://”
            - BluffA.get.photo() would return “C://”
            - It is a getter function thus it should not mutate.
- Get.weather
    - Returns string of current weather
        - Bluff A’s Weather = 70
        - Returns in the format “70F/21.11C” for the weather 70 degrees Fahrenheit and 21.11 Degrees Celcius. This function also translates it to celsius before returning.
- get.trainCapacity()
    - Returns train capacity as an Int
        - Example:
            - Train A has 78 passengers
            - TrainA.get.trainCapacity() would return 78 because it has a capacity of 78
- get.alarm()
    - Returns boolean if it is safe to ride
        - True being safe
        - False being dangerous
        - Example:
            - Train A’s safe = false;
                - The getter would return false because it is unsafe to ride.
                - It is a getter function thus it should not mutate.
- get.inforamtion()
    - Returns all of the information about the Bluff that is held by the constructor
        - Example:
            - Bluff A’s safe = false;
            - Bluff A’s Weather = 70
            - Bluff A’s photo = “C://”
                - Returns: “Is not safe to be by Bluff A, the current weather a bluff A is 70 degrees Fahrenheit and a current photo of Bluff A “C://””
                - It is a getter function thus it should not mutate.