VDM SPECIFICATION DOCUMENT

Formal Methods in Software Engineering (SE-313)

Department of Software
Engineering, NED University
of Engineering &
Technology, Karachi

Roll No	SE-21084
Name	Syed Arsalan
Batch	2021
Year	2023-2024
Department	Software Engineering

Fire Alarm System

Scope:

The Fire Alarm System project is a comprehensive initiative aimed at developing a sophisticated and reliable system for detecting and responding to changes in environmental temperature, with a primary focus on identifying potential fire hazards. At its core is the FireAlarmSystem class, encapsulating critical state variables, including threshold temperature, environment temperature, alarm status, heat detection, and sound intensity. The project seeks to ensure the system's robustness by mandating that the environment temperature, threshold temperature, and sound intensity are greater than zero during the initialization phase.

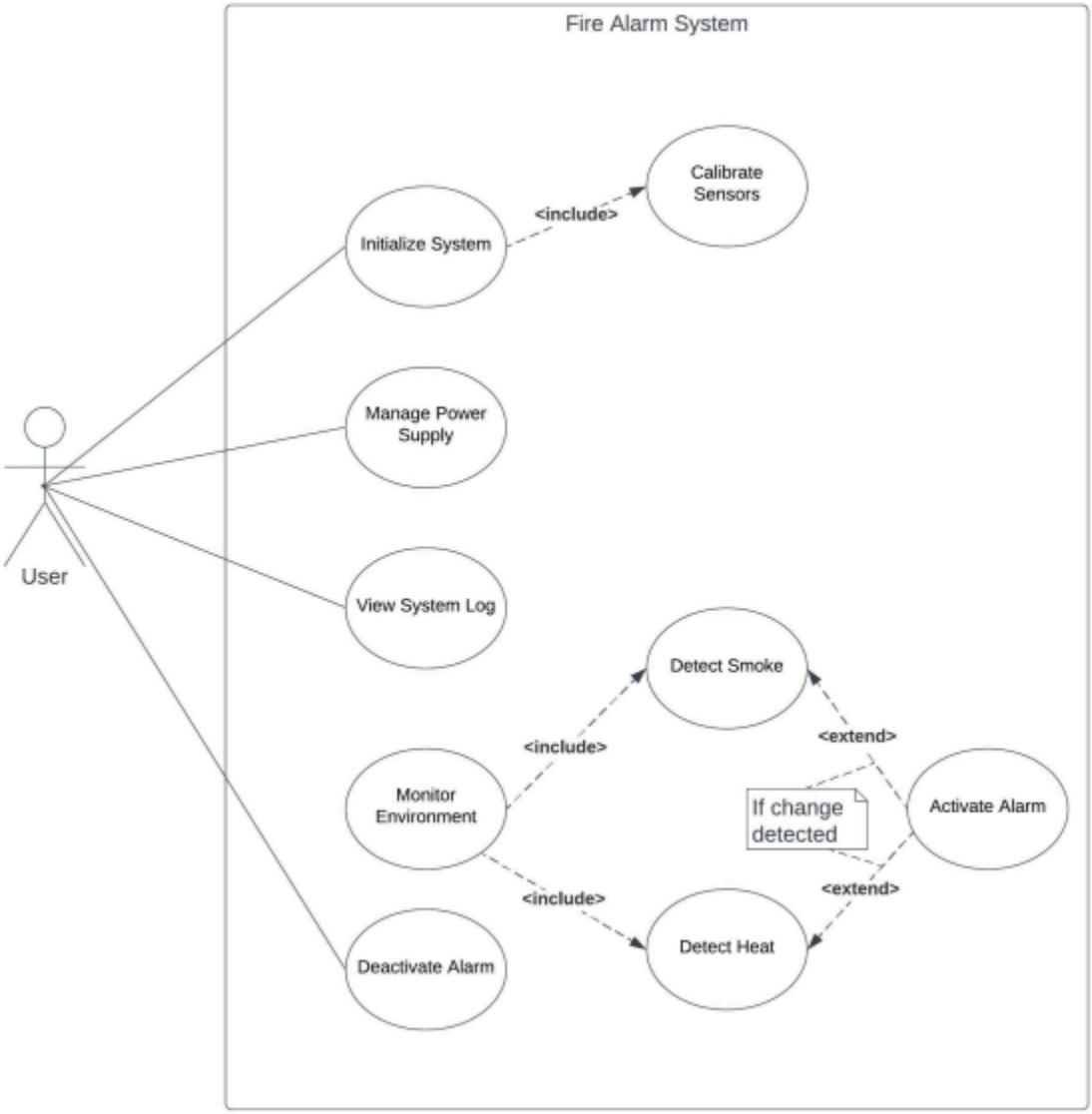
The system's functionality encompasses various operations designed to provide a seamless and effective fire detection and alarm mechanism. These operations include setting the threshold temperature, reading the environment temperature, adjusting the alarm volume, detecting changes in temperature, triggering and deactivating the alarm, and retrieving system statistics. The setThresholdTemp operation allows users to define the threshold temperature, a pivotal parameter influencing the system's response. Similarly, the getEnvironmentTemp operation facilitates the continuous monitoring of the ambient temperature, updating the system state accordingly.

The adjustVolume operation caters to user preferences by enabling customization of the alarm sound intensity. The detectChange operation, a key feature, signals a change in temperature and instructs the alarm to sound if the temperature surpasses the specified threshold. Additionally, the triggerAlarm and deactivateAlarm operations facilitate the activation and deactivation of the alarm, respectively, based on the detected heat and the established threshold.

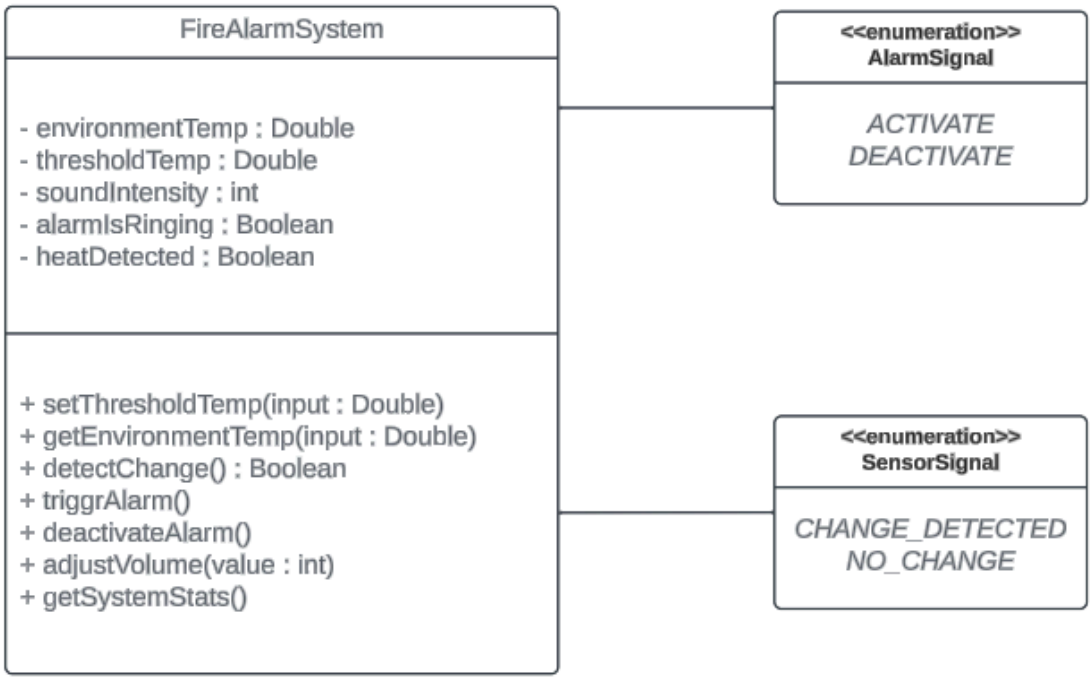
The project's overarching objective is to provide a versatile and user-friendly solution for early fire detection, ensuring safety in diverse environments. The getSystemStats operation serves as a means to retrieve vital system statistics, consolidating information such as environment temperature, threshold temperature, sound intensity, alarm status, and heat detection. Through a careful integration of these operations and functionalities, the Fire Alarm System project aspires to contribute significantly to fire safety measures, offering a robust and effective solution for various settings.

4 + 1 Architecture:

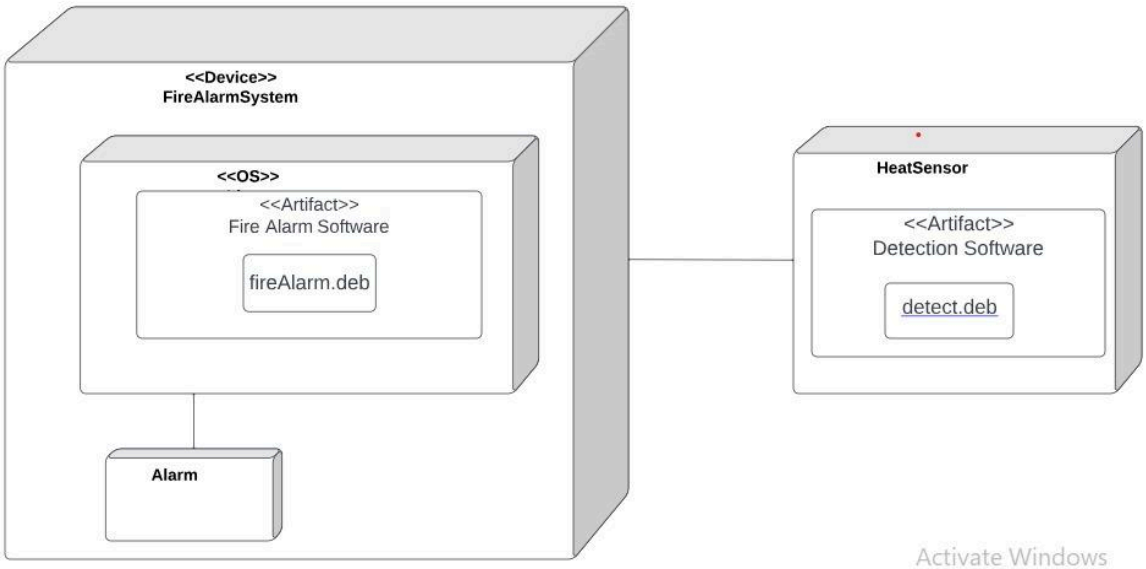
1) Scenario View



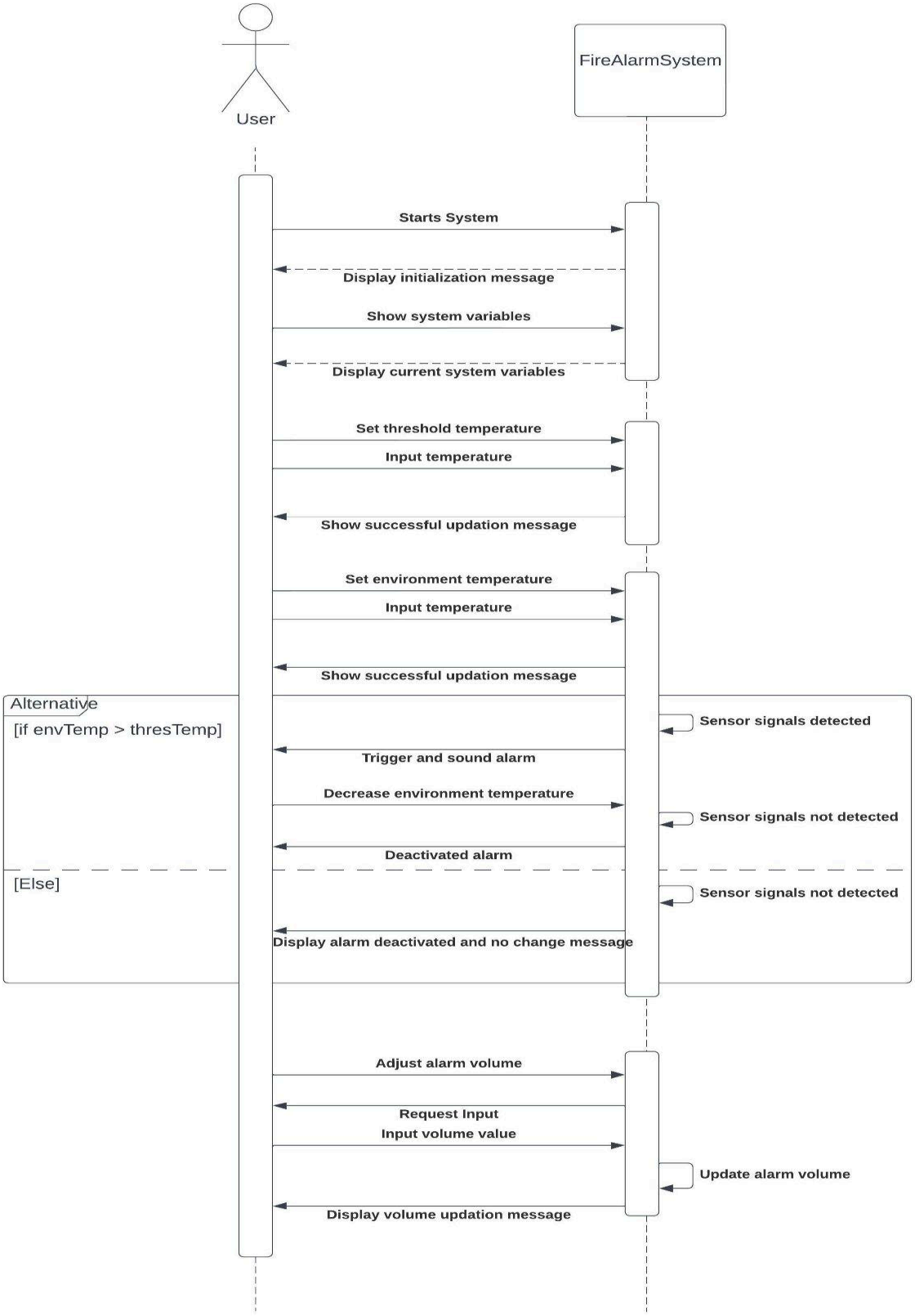
2) Logical View



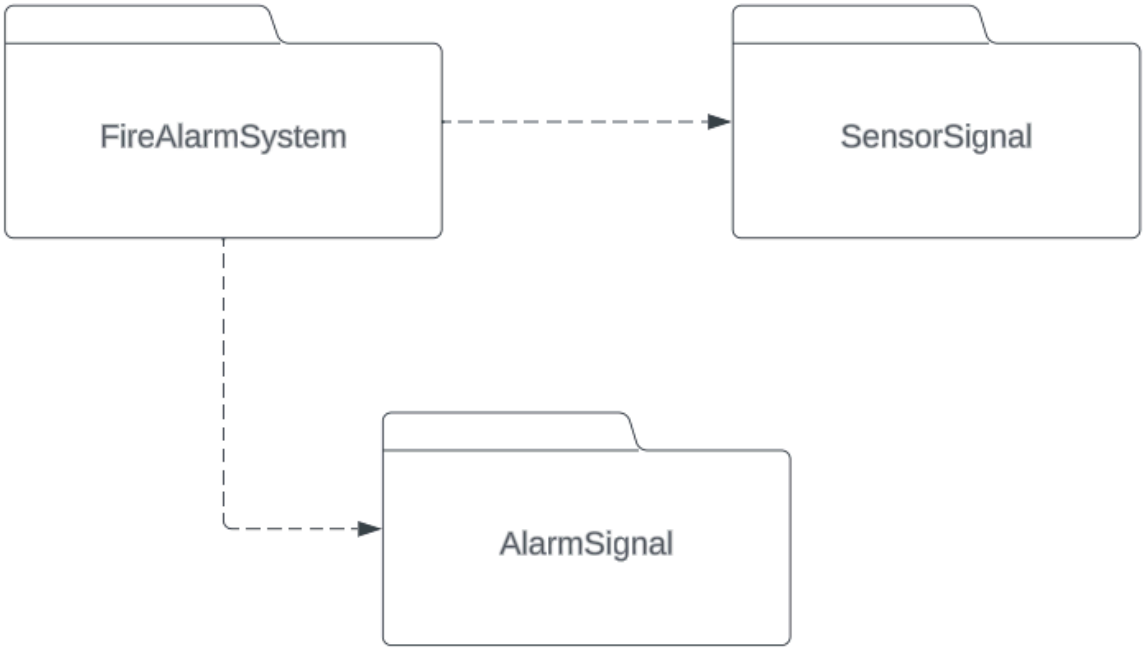
3) Physical View



4) Process View



5) Development View



VDM Specification:

FireAlarmSystem Class:

types
AlarmSignal = <*ACTIVATE*> | <*DEACTIVATE*>
SensorSignal = <*CHANGE_DETECTED*> | <*NO_CHANGE*>

values
MIN : $\mathbb{R}$ = 0

state *FireAlarmSystem* **of**
thresholdTemp : [$\mathbb{R}$]
environmentTemp : [$\mathbb{R}$]
alarmIsRinging : [$\mathbb{B}$]
heatDetected: [$\mathbb{B}$]
soundIntensity: [$\mathbb{R}$]
-- the environment temperature, threshold temperature and sound intensity should be greater than zero
inv-mk-*FireAlarmSystem* (*e, t, s, a, h*) $\triangleq$ (*isAboveZero*(*e*) $\vee$ *e* = **nil**) $\wedge$
(*isAboveZero*(*t*) $\vee$ *t* = **nil**) $\wedge$
(*isAboveZero*(*s*) $\vee$ *s* = **nil**) $\wedge$
s = FALSE $\wedge$ *h* = FALSE
– all the states are assigned default values when the system is initialized
init-mk-*FireAlarmSystem* (*e, t, s, a, h*) $\triangleq$ *e* = -1 $\wedge$ *t* = -1 $\wedge$ *a* = 0 $\wedge$ *h* = FALSE $\wedge$
s = FALSE
end

functions
isAboveZero(*val* : $\mathbb{R}$) *outcome* : $\mathbb{B}$
pre TRUE
post *outcome* $\Leftrightarrow$ *val* $\geq$ *minTemp*

operations
– this operation assigns the input value to the threshold temperature of the system
setThresholdTemp (*inputTemp* : $\mathbb{R}$)
ext wr *thresholdTemp* : [$\mathbb{R}$]
pre *isAboveZero*(*inputTemp*)
post *thresholdTemp* = *inputTemp*

– this operation reads the environment temperature and assigns it to the state variable

getEnvironmentTemp (*inputTemp* : [ℝ])
ext wr *environmentTemp* : [ℝ]
    rd *thresholdTemp* : [ℝ]
pre *isAboveZero*(*inputTemp*) ∧ *thresholdTemp* ≠ **nil**
post *environmentTemp* = *inputTemp*

– this operation records the sound intensity of the alarm which shall ring on heat detection

adjustVolume (*volume* : [ℝ])
ext wr *soundIntensity*: [ℝ]
pre *isAboveZero*(*volume*)
post *soundIntensity* = *volume*

detectChange () *signal* : *SensorSignal*
ext wr *heatDetected* : [𝔹]
    rd *environmentTemp* : [ℝ]
    rd *thresholdTemp* : [ℝ]
pre *thresholdTemp* ≠ **nil** ∧ *environmentTemp* ≠ **nil**
post (*environmentTemp* ≥ *thresholdTemp* ∧ *heatDetected* = TRUE ∧ *signal* = <*CHANGE_DETECTED*>) ∨ (*environmentTemp* < *thresholdTemp* ∧ *heatDetected* = FALSE ∧ *signal* = <*NO_CHANGE*>)

– this operation detects change in temperature and instructs the alarm to sound if the temperature is higher than a certain value

triggerAlarm () *signal* : *AlarmSignal*
ext wr *alarmIsRinging* : [𝔹]
    rd *environmentTemp* : [ℝ]
    rd *thresholdTemp* : [ℝ]
    rd *heatDetected*: [𝔹]
pre *heatDetected* = TRUE ∧ *environmentTemp* ≥ *thresholdTemp*
post *signal* = <*ACTIVATE*> ∧ *alarmIsRinging* = TRUE

deactivateAlarm () *signal* : *AlarmSignal*
ext wr *alarmIsRinging* : [𝔹]
    rd *environmentTemp* : [ℝ]
    rd *thresholdTemp* : [ℝ]
    rd *heatDetected*: [𝔹]
pre *alarmIsRinging* = TRUE ∧ *environmentTemp* < *thresholdTemp*
post *signal* = <*DEACTIVATE*> ∧ *alarmIsRinging* = FALSE

getSystemStats () *envTemp* : [ℝ] *threshTemp* : [ℝ] *soundInt* : [ℝ] *alarmStatus* : [𝔹] *heatStatus* : [𝔹]

ext rd *alarmIsRinging* : [𝔹]
    rd *environmentTemp* : [ℝ]
    rd *thresholdTemp* : [ℝ]
    rd *heatDetected*: [𝔹]
    rd *soundIntensity* : [ℝ]

pre TRUE

post *envTemp* = *environmentTemp* ∧ *thresholdTemp* = *threshTemp* ∧ *soundIntensity* = *soundInt* ∧ *alarmIsRinging* = *alarmStatus* ∧ *heatDetected* = *heatStatus*

Implementing the FireAlarmSystem Class Specifications

1. Translating the values clause into C++

VDM- SL	C++
values *MIN* : ℝ = 0	public: const double MIN = 0;

2. Translating A State Clause Into C++

VDM- SL	C++
state *FireAlarmSystem* **of** *thresholdTemp* : [ℝ] *environmentTemp* : [ℝ] *alarmIsRinging* : [𝔹] *heatDetected*: [𝔹] *soundIntensity*: [ℝ]	public: double thresholdTemp; double environmentTemp; bool alarmIsRinging; bool heatDetected; double soundIntensity;

3. Translating An Invariant Into C++

VDM- SL	C++
inv-mk-*FireAlarmSystem* (*e, t, s, a, h*) ≜ (*isAboveZero*(*e*) ∨ *e* = **nil**) ∧ (*isAboveZero*(*t*) ∨ *t* = **nil**) ∧ (*isAboveZero*(*s*) ∨ *s* = **nil**) ∧ = FALSE ∧ *h* = FALSE	bool invCheck() { bool expression = ((this->thresholdTemp >= MIN || this->thresholdTemp == -1) && (this->environmentTemp >= MIN || this->environmentTemp == -1) && (this->soundIntensity > MIN || this->soundIntensity == 0) && (this->heatDetected == false)

	&& (this->alarmIsRinging == false));
	if (invTest(expression))
	{
	return true;
	}
	return false;
	}

4. The Invariant Check Interface

```
class Invariant
{
public:
   virtual bool invCheck() = 0;
};
```

5. The Vdm Class

```
class VDM
{
public:
   bool invTest(bool expression)
   {
       if (!expression)
       {
           throw runtime_error("VDMException: State Invariant not
satisfied");
       }
       else
       {
           cout << "System successfully initialized" << endl;
           return true;
       }
   }

   bool preTest(bool expression)
   {
```

```
        if (!expression)
        {
            throw runtime_error("VDMException: Pre-condition not
satisfied");
        }
        return true;
    }
};
```

6. Translating The Initialization Clause Into C++

VDM- SL	C++

init-mk-*FireAlarmSystem* (*e, t, s, a, h*) $\triangleq$ *e* = -1 $\wedge$ *t* = -1 $\wedge$ *a* = 0 $\wedge$ *h* = FALSE $\wedge$ *s* = FALSE	FireAlarmSystem(int t, int e, int a, int h, int s)
	{
	thresholdTemp = t;
	environmentTemp = e;
	alarmIsRinging = a;
	heatDetected = h;
	soundIntensity = s;
	invCheck();
	}

7. Translating Operation Specifications Into C++

VDM- SL	C++

setThresholdTemp (*inputTemp* : $\mathbb{R}$)
ext wr *thresholdTemp* : [$\mathbb{R}$]
pre *isAboveZero*(*inputTemp*)
post *thresholdTemp* = *inputTemp*
	void setThresholdTemp(double inputTemp)
	{
	bool check = preTest(inputTemp > MIN);
	if (check)
	{
	this->thresholdTemp = inputTemp;
	cout << endl

	<< "Threshold temperature successfully updated!" << endl; } }

VDM- SL	**C++**

getEnvironmentTemp (*inputTemp* : [ℝ])
ext wr *environmentTemp* : [ℝ]
rd *thresholdTemp* : [ℝ]
pre *isAboveZero*(*inputTemp*) ∧ *thresholdTemp* ≠ **nil**
post *environmentTemp* = *inputTemp*
	void getEnvironmentTemp(double inputTemp)
	{
	bool check = preTest(inputTemp > MIN && thresholdTemp != -1);
	if (check)
	{
	environmentTemp = inputTemp;
	cout << endl
	<< "Environment temperature successfully updated!" << endl;
	}
	}

VDM- SL	**C++**

adjustVolume (*volume* : [ℝ])
ext wr *soundIntensity*: [ℝ]
pre *isAboveZero*(*volume*)
post *soundIntensity* = *volume*
	void adjustVolume(double sound)
	{
	bool check = preTest(sound > 0);
	if (check)
	{
	soundIntensity = sound;
	cout << endl
	<< "Alarm Sound Intensity

	successfully updated!" << endl << endl; } }

VDM- SL	**C++**

detectChange () *signal* : *SensorSignal*
ext wr *heatDetected* : [$\mathbb{B}$]
  rd *environmentTemp* : [$\mathbb{R}$]
  rd *thresholdTemp* : [$\mathbb{R}$]
pre *thresholdTemp* ≠ **nil** ∧ *environmentTemp* ≠ **nil**
post (*environmentTemp* ≥ *thresholdTemp* ∧ *heatDetected* = TRUE ∧ *signal* =
  <*CHANGE_DETECTED*>)
∨(*environmentTemp* < *thresholdTemp* ∧ *heatDetected* =
  FALSE ∧ *signal* = <*NO_CHANGE*>)
	bool detectChange()
	 {
	  bool check = preTest(thresholdTemp != -1 && environmentTemp != -1);
	  if (check)
	  {
	   if (environmentTemp >= thresholdTemp)
	   {
	    heatDetected = true;
	    return true;
	   }
	   else
	   {
	    heatDetected = false;
	    return false;
	   }
	  }
	 }

VDM- SL	**C++**

triggerAlarm () *signal* : *AlarmSignal*
ext wr *alarmIsRinging* : [$\mathbb{B}$]
rd *environmentTemp* : [$\mathbb{R}$]
rd *thresholdTemp* : [$\mathbb{R}$]
rd *heatDetected*: [$\mathbb{B}$]
pre *heatDetected* = TRUE $\wedge$
environmentTemp $\geq$ *thresholdTemp*
post *signal* = <*ACTIVATE*> $\wedge$
alarmIsRinging = TRUE
	void triggerAlarm()
	{
	bool check = preTest(heatDetected == true && (environmentTemp >= thresholdTemp));
	if (check)
	{
	asignal = ACTIVATE;
	alarmIsRinging = true;
	cout << "////////////// ALARM IS RINGING at Sound Intensity " << soundIntensity << " //////////// " << endl;
	}
	}

VDM- SL	**C++**

deactivateAlarm () *signal* : *AlarmSignal*
ext wr *alarmIsRinging* : [$\mathbb{B}$]
rd *environmentTemp* : [$\mathbb{R}$]
rd *thresholdTemp* : [$\mathbb{R}$]
rd *heatDetected*: [$\mathbb{B}$]
pre *alarmIsRinging* = TRUE $\wedge$
environmentTemp < *thresholdTemp*
post *signal* = <*DEACTIVATE*> $\wedge$
alarmIsRinging = FALSE
	void deactivateAlarm()
	{
	bool check = preTest(alarmIsRinging == true && (environmentTemp < thresholdTemp));
	if (check)
	{
	asignal = DEACTIVATE;
	alarmIsRinging = false;
	environmentTemp = thresholdTemp - 10;
	cout << "////////////// ALARM DEACTIVATED //////////// " << endl;

	} }

VDM- SL	**C++**

getSystemStats () *envTemp* : [ℝ] *threshTemp* : [ℝ] *soundInt* : [ℝ] *alarmStatus* : [𝔹] *heatStatus* :[𝔹]
ext rd *alarmIsRinging* : [𝔹]
rd *environmentTemp* : [ℝ]
rd *thresholdTemp* : [ℝ]
rd *heatDetected*: [𝔹]
rd *soundIntensity* : [ℝ]
pre TRUE
post *envTemp* = *environmentTemp* ∧ *thresholdTemp* = *threshTemp* ∧ *soundIntensity* = *soundInt*
∧ *alarmIsRinging* = *alarmStatus* ∧ *heatDetected* = *heatStatus*
	void getSystemStats()
	{
	cout << endl
	<< "System Threshold Temperature: " << thresholdTemp << endl;
	cout << "Current Environment Temperature: " << environmentTemp << endl;
	cout << "System sound intensity: " << soundIntensity << endl;
	cout << "Alarm ringing status: " << alarmIsRinging << endl;
	cout << "Heat Detection Status : " << heatDetected << endl
	<< endl;
	}

8. Translating The Signal Type Into C++

AlarmSignal:

AlarmSignal

```
enum alarmSignal
{
   ACTIVATE,
   DEACTIVATE
} asignal;
```

SensorSignal:

SensorSignal

```
enum sensorSignal
{
   CHANGE_DETECTED,
   NO_CHANGE
} signal;
```

9. Driver Code

```
#include <iostream>
#include <string>
#include <thread>
#include <chrono>
using namespace std;


enum alarmSignal
{
   ACTIVATE,
   DEACTIVATE
} asignal;
```

```
enum sensorSignal
{
   CHANGE_DETECTED,
   NO_CHANGE
} signal;

class Invariant
{
public:
   virtual bool invCheck() = 0;
};

class VDM
{
public:
   bool invTest(bool expression)
   {
       if (!expression)
       {
           throw runtime_error("VDMException: State Invariant not
satisfied");
       }
       else
       {
           cout << "System successfully initialized" << endl;
           return true;
       }
   }

   bool preTest(bool expression)
   {
       if (!expression)
       {
           throw runtime_error("VDMException: Pre-condition not
satisfied");
       }
       return true;
   }
};
```

```
class FireAlarmSystem : public Invariant, public VDM
{
public:
   double thresholdTemp;
   double environmentTemp;
   bool alarmIsRinging;
   bool heatDetected;
   double soundIntensity;

public:

   const double MIN = 0;
   FireAlarmSystem(int t, int e, int a, int h, int s)
   {

       thresholdTemp = t;
       environmentTemp = e;
       alarmIsRinging = a;
       heatDetected = h;
       soundIntensity = s;
       invCheck();
   }


   bool invCheck()
   {
       bool expression = ((this->thresholdTemp >= MIN ||
this->thresholdTemp == -1) &&
                          (this->environmentTemp >= MIN ||
this->environmentTemp == -1) &&
                          (this->soundIntensity > MIN ||
this->soundIntensity == 0) &&
                          (this->heatDetected == false) &&
(this->alarmIsRinging == false));
       if (invTest(expression))
       {
           return true;
       }
       return false;
   }
```

```
    void setThresholdTemp(double inputTemp)
    {

        bool check = preTest(inputTemp > MIN);

        if (check)
        {
            this->thresholdTemp = inputTemp;
            cout << endl
                 << "Threshold temperature successfully updated!" << endl;
        }
    }

    void getEnvironmentTemp(double inputTemp)
    {
        bool check = preTest(inputTemp > MIN && thresholdTemp != -1);

        if (check)
        {
            environmentTemp = inputTemp;
            cout << endl
                 << "Environment temperature successfully updated!" <<
endl;
        }
    }

    bool detectChange()
    {
        bool check = preTest(thresholdTemp != -1 && environmentTemp != -1);

        if (check)
        {
            if (environmentTemp >= thresholdTemp)
            {
                heatDetected = true;
                return true;
            }
            else
            {
```

```
                heatDetected = false;
                return false;
            }
        }
    }

    void triggerAlarm()
    {
        bool check = preTest(heatDetected == true && (environmentTemp >=
thresholdTemp));

        if (check)
        {
            asignal = ACTIVATE;
            alarmIsRinging = true;
            cout << "////////////// ALARM IS RINGING at Sound Intensity " <<
soundIntensity << " //////////////" << endl;
        }
    }

    void deactivateAlarm()
    {
        bool check = preTest(alarmIsRinging == true && (environmentTemp <
thresholdTemp));

        if (check)
        {
            asignal = DEACTIVATE;
            alarmIsRinging = false;
            environmentTemp = thresholdTemp - 10;
            cout << "////////////// ALARM DEACTIVATED //////////////" << endl;
        }
    }

    void adjustVolume(double sound)
    {
        bool check = preTest(sound > 0);

        if (check)
        {
```

```
            soundIntensity = sound;
            cout << endl
                 << "Alarm Sound Intensity successfully updated!" << endl
                 << endl;
        }
    }

    void getSystemStats()
    {
        cout << endl
             << "System Threshold Temperature: " << thresholdTemp << endl;
        cout << "Current Environment Temperature: " << environmentTemp <<
endl;
        cout << "System sound intensity: " << soundIntensity << endl;
        cout << "Alarm ringing status: " << alarmIsRinging << endl;
        cout << "Heat Detection Status : " << heatDetected << endl
             << endl;
    }
};
class FireAlarmSystemTester
{
public:
    void startSystem()
    {
        try
        {
            cout << '\t' << '\t' << "FIRE ALARM SYSTEM" << endl
                 << endl;
            FireAlarmSystem system(-1, -1, false, false, 0);
            int choice;
            double temp;
            double sound;
            do
            {
                try
                {
                    cout << endl
                         << endl
                         << "DISPLAYING MENU" << endl
                         << endl;
```

```
        cout << "1: Show System Variables " << endl
             << "2: Specify Threshold Temperature" << endl
             << "3: Get Environment Temperature " << endl
             << "4: Adjust Sound Intensity " << endl
             << "5: EXIT" << endl
             << endl;
        cout << "Enter Choice: ";
        cin >> choice;
        switch (choice)
        {

        case 1:
            cout << endl
                 << "SHOWING SYSTEM CONDITIONS" << endl;
            option1(system);
            break;
        case 2:
            cout << endl
                 << "Enter desired Threshold Temperature: ";
            cin >> temp;
            option2(system, temp);
            break;
        case 3:
            cout << endl
                 << "Enter current Enivronment Temperature: ";
            cin >> temp;
            option3(system, temp);
            break;
        case 4:
            cout << endl
                 << "Enter Required Sound Intensity: ";
            cin >> sound;
            option4(system, sound);
            break;
        default:
            break;
        }
    }
    catch (const std::exception &e)
    {
```

```
                std::cerr << '\n'
                          << e.what() << '\n'
                          << '\n';
            }

        } while (choice != 5);
    }
    catch (const std::exception &e)
    {
        std::cerr << e.what() << '\n';
        cout << "Initialization failed " << endl
             << "Enter valid input fields" << endl
             << endl;
    }
}
void option1(FireAlarmSystem &system)
{
    system.getSystemStats();
}
void option2(FireAlarmSystem &system, double inputTemp)
{
    system.setThresholdTemp(inputTemp);
}
void option3(FireAlarmSystem &system, double inputTemp)
{
    system.getEnvironmentTemp(inputTemp);
    bool check = system.detectChange();
    double temp;
    if (check)
    {
        signal = CHANGE_DETECTED;
        cout << "SENSOR SIGNAL: CHANGE_DETECTED" << endl;
        char deactivate;
        while (deactivate != 'd')
        {
            asignal = ACTIVATE;
            cout << endl
                 << "ALARM SIGNAL: ACTIVATE" << endl;
            system.triggerAlarm();
            cout << "Enter d to deactivate: ";
```

```
                cin >> deactivate;
                if (deactivate == 'd')
                {
                    do
                    {
                        cout << endl
                             << "Decrease Environment Temperature to
deactivate: ";
                        cin >> temp;
                        if (temp < system.thresholdTemp)
                        {
                            system.environmentTemp = temp;
                            asignal = DEACTIVATE;
                            cout << endl
                                 << "ALARM SIGNAL: DEACTIVATE" << endl;
                            system.deactivateAlarm();
                            system.detectChange();
                        }
                        else
                        {
                            cout << "Temperature still too high, decrease
it further" << endl
                                 << endl;
                        }
                    } while (temp >= system.thresholdTemp);
                }
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
        }
        else
        {
            signal = NO_CHANGE;
            cout << "SENSOR SIGNAL: NO_CHANGE" << endl
                 << "Alarm in DEACTIVATE Status" << endl
                 << endl;
        }
    }
    void option4(FireAlarmSystem &system, double sound)
    {
        system.adjustVolume(sound);
```

```
    }
};

int main()
{
    FireAlarmSystemTester fireSystem;

    fireSystem.startSystem();

    return 0;
}
```

Testing the Java Class

The FireAlarmSystem tester class

```
class FireAlarmSystemTester
{
public:
    void startSystem()
    {
        try
        {
            cout << '\t' << '\t' << "FIRE ALARM SYSTEM" << endl
                 << endl;
            FireAlarmSystem system(-1, -1, false, false, 0);
            int choice;
            double temp;
            double sound;

            do
            {
                try
                {
                    cout << endl
                         << endl
                         << "DISPLAYING MENU" << endl
                         << endl;
```

```
                cout << "1: Show System Variables " << endl
                     << "2: Specify Threshold Temperature" << endl
                     << "3: Get Environment Temperature " << endl
                     << "4: Adjust Sound Intensity " << endl
                     << "5: EXIT" << endl
                     << endl;

                cout << "Enter Choice: ";
                cin >> choice;

                switch (choice)
                {

                case 1:
                    cout << endl
                         << "SHOWING SYSTEM CONDITIONS" << endl;
                    option1(system);
                    break;
                case 2:
                    cout << endl
                         << "Enter desired Threshold Temperature: ";
                    cin >> temp;
                    option2(system, temp);
                    break;
                case 3:
                    cout << endl
                         << "Enter current Enivronment Temperature:
";
                    cin >> temp;
                    option3(system, temp);
                    break;
                case 4:

                    cout << endl
                         << "Enter Required Sound Intensity: ";
                    cin >> sound;
                    option4(system, sound);
                    break;
```

```
                default:
                    break;
                }
            }
            catch (const std::exception &e)
            {
                std::cerr << '\n'
                          << e.what() << '\n'
                          << '\n';
            }

        } while (choice != 5);
    }
    catch (const std::exception &e)
    {
        std::cerr << e.what() << '\n';
        cout << "Initialization failed " << endl
             << "Enter valid input fields" << endl
             << endl;
    }
}

void option1(FireAlarmSystem &system)
{
    system.getSystemStats();
}

void option2(FireAlarmSystem &system, double inputTemp)
{
    system.setThresholdTemp(inputTemp);
}
void option3(FireAlarmSystem &system, double inputTemp)
{
    system.getEnvironmentTemp(inputTemp);
    bool check = system.detectChange();
    double temp;
    if (check)
    {
```

```
            signal = CHANGE_DETECTED;
            cout << "SENSOR SIGNAL: CHANGE_DETECTED" << endl;
            char deactivate;

            while (deactivate != 'd')
            {
                asignal = ACTIVATE;
                cout << endl
                     << "ALARM SIGNAL: ACTIVATE" << endl;
                system.triggerAlarm();

                cout << "Enter d to deactivate: ";
                cin >> deactivate;

                if (deactivate == 'd')
                {
                    do
                    {
                        cout << endl
                             << "Decrease Environment Temperature to
deactivate: ";
                        cin >> temp;
                        if (temp < system.thresholdTemp)
                        {
                            system.environmentTemp = temp;
                            asignal = DEACTIVATE;
                            cout << endl
                                 << "ALARM SIGNAL: DEACTIVATE" << endl;
                            system.deactivateAlarm();
                            system.detectChange();
                        }
                        else
                        {
                            cout << "Temperature still too high, decrease
it further" << endl
                                 << endl;
                        }
                    } while (temp >= system.thresholdTemp);
                }
```

```
                std::this_thread::sleep_for(std::chrono::seconds(1));
            }
        }
        else
        {
            signal = NO_CHANGE;
            cout << "SENSOR SIGNAL: NO_CHANGE" << endl
                 << "Alarm in DEACTIVATE Status" << endl
                 << endl;
        }
    }
    void option4(FireAlarmSystem &system, double sound)
    {
        system.adjustVolume(sound);
    }
};
```

Boundary Value Analysis:

Test Cases (Environment Temperature)	Expected Output	Actual Output
-1	Invalid Temperature	VDMException: State Invariant not satisfied
0	Valid Temperature	System successfully initialized
1	Valid Temperature	System successfully initialized