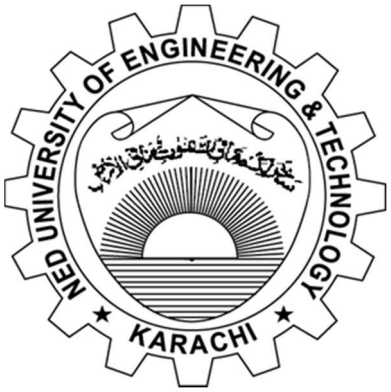


Department of Software Engineering, NEDUET

Formal Methods in Software Engineering **(SE-313)**

Smart Grid Control System
Formal Specification (VDM)

Muhammad Zain Ul Abedin SE – 21081

Submitted to: -

Dr. Mustafa Latif

1. PROJECT SCOPE

The **scope** of the *Smart Grid Control System* revolves around creating a versatile and efficient platform for managing devices within a smart grid infrastructure. The system enables users to add and remove *devices*. The functionalities include adding devices and removing device as per power requirements.The system aims to provide a robust and adaptable solution for effective smart grid management.

SmartGridControlSystem Class

The SmartGridControlSystem class serves as the ***central orchestrator***, managing multiple Device instances. It includes functionalities for adding new Devices, checking the existence of device IDs, removing devices and displaying information about the connected devices.

CRITICAL NATURE

The critical nature of the Smart Grid Control System is inherently tied to the fundamental requirement of maintaining the power balance invariant. This invariant is not merely a design constraint but a *foundational principle* that directly impacts the stability, reliability, and safety of the entire smart grid infrastructure

Let:

C_t *be total consumption of electricity at time t*

G_t *be total generation of electricity at time t*

P_t *be the net power balance at time t* $(Pt = Gt - Ct)$

The Power Balance Invariant can be formally expressed as

$$\forall t, Pt \geq 0$$

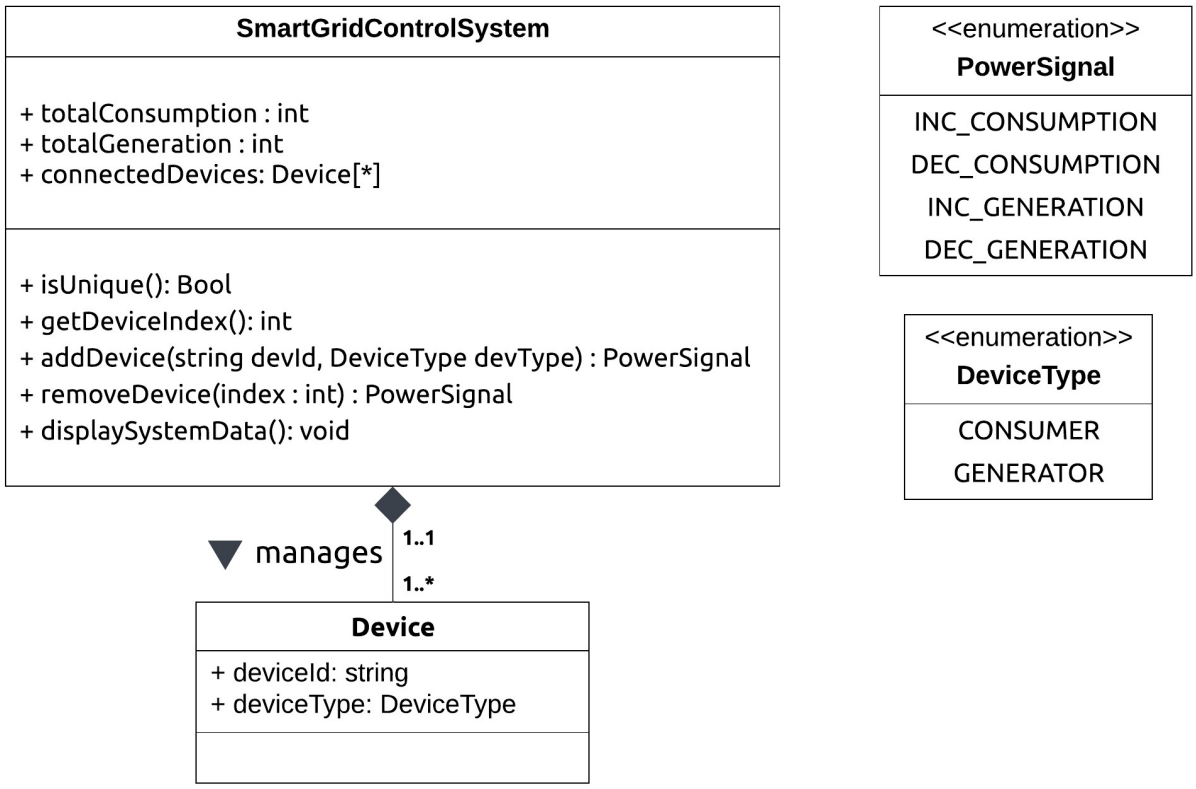
This equation states that at any given time t, the net power balance (Pt) must be greater than or equal to zero

2. 4+1 ARCHITECTURE

The 4+1 architecture of Smart Grid Control System can be illustrated as follows

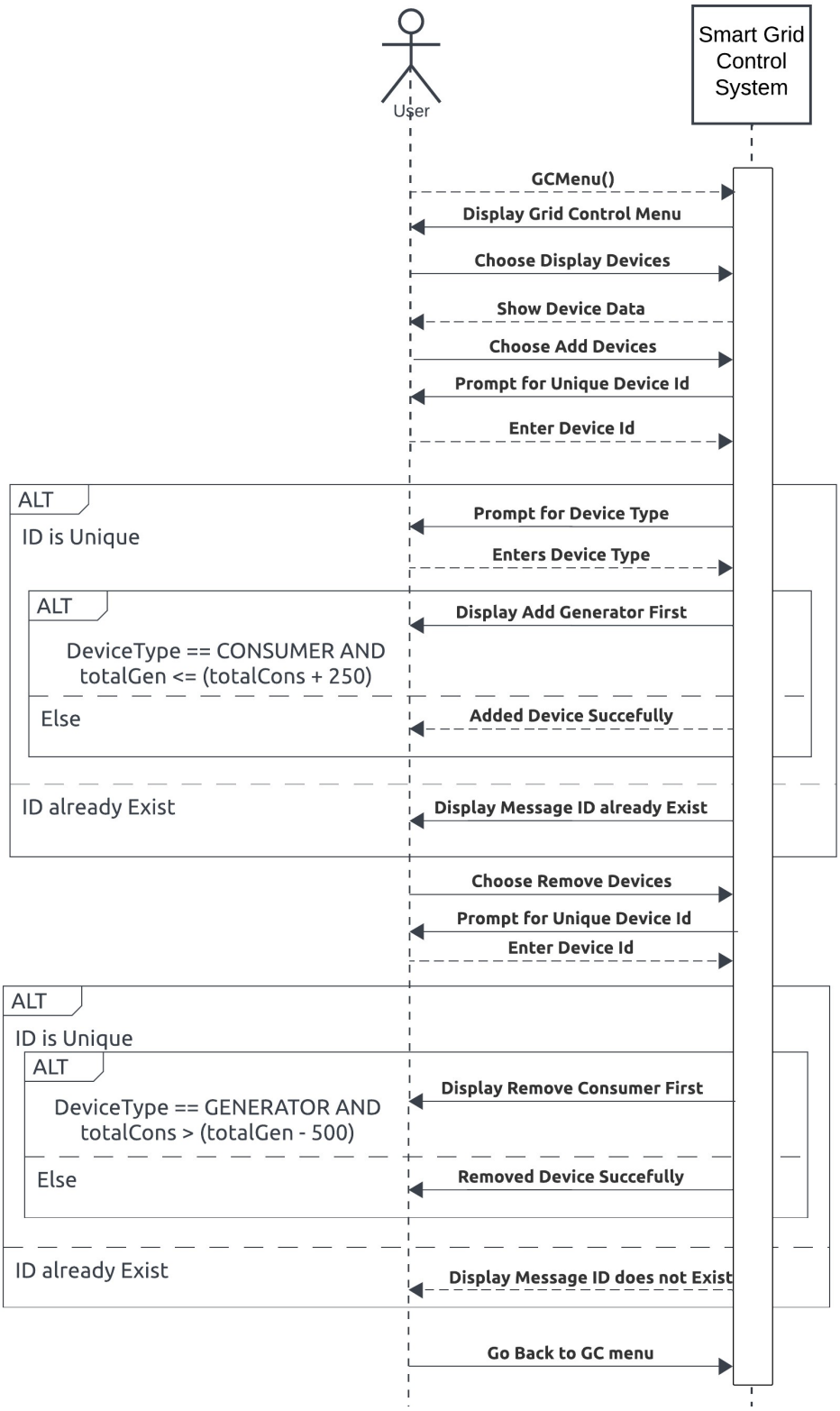
2.1. LOGICAL

Logical View of SGCS demonstrated through Class Diagram



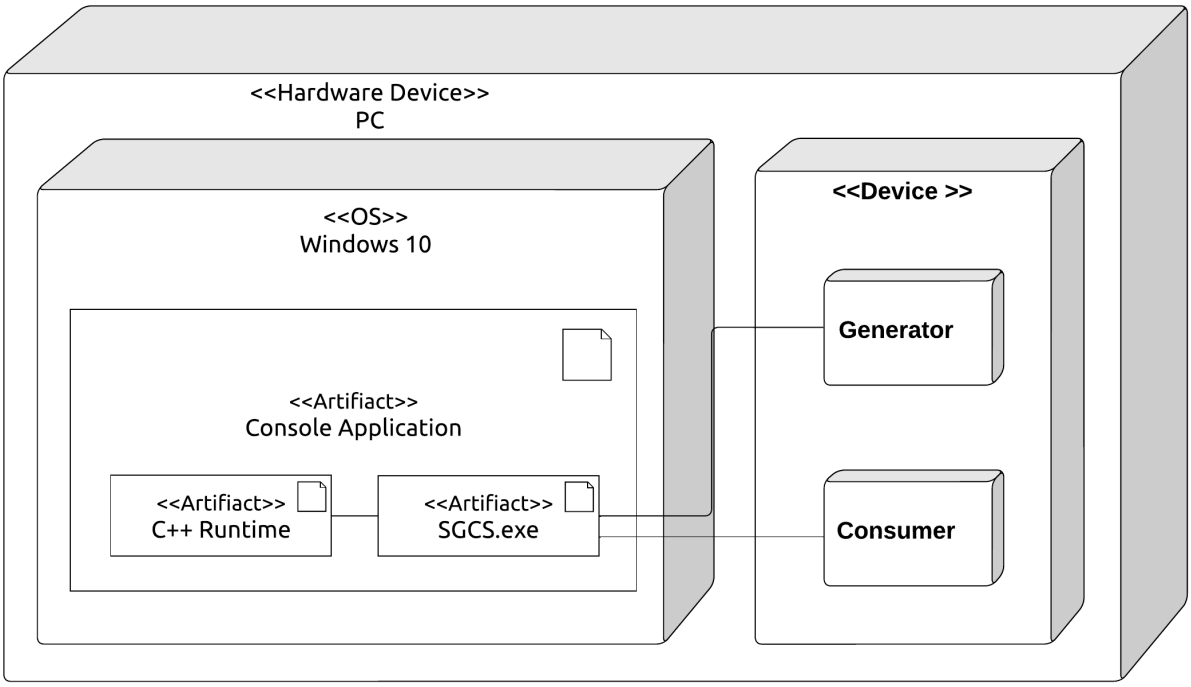
2.2. PROCESS

The Process View of SGCS demonstrated through Sequence Diagram



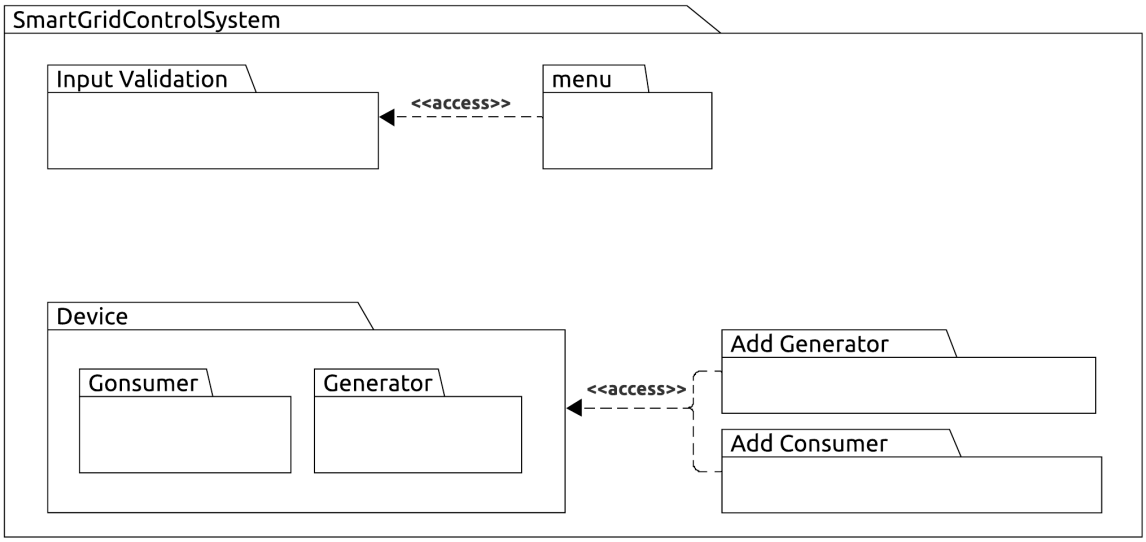
2.3. PHYSICAL

The Physical View can be demonstrated by Deployment Diagram



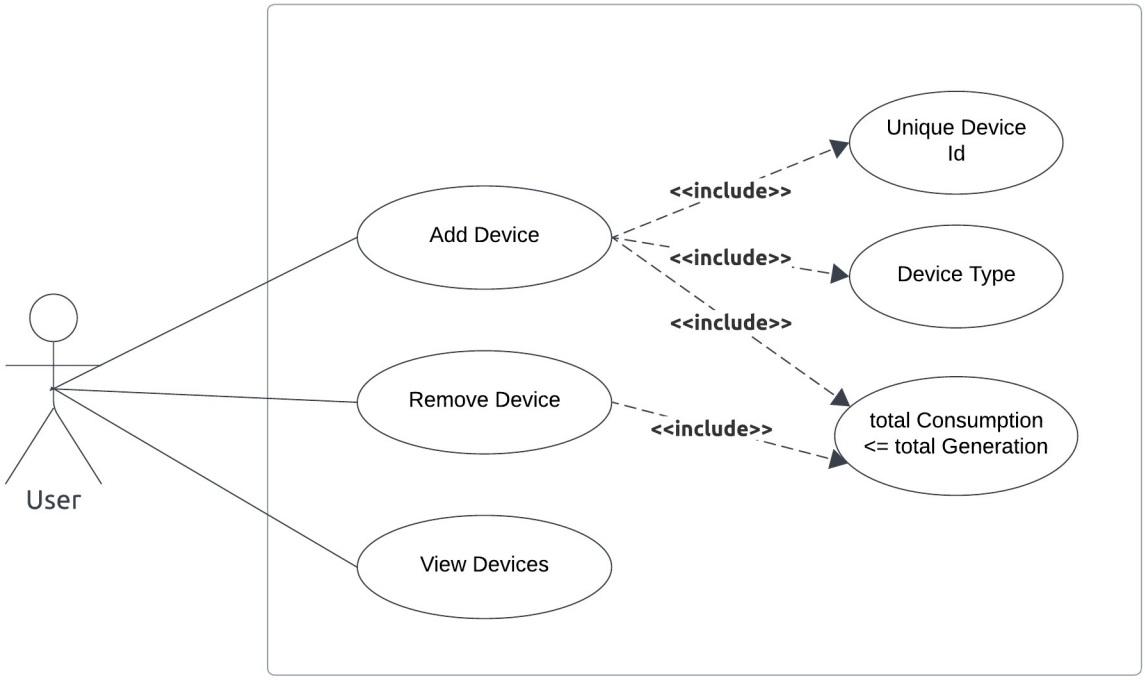
2.4. DEVELOPMENT

The Development View can be demonstrated with Package Diagram



2.5. SCENARIO

The Development View can be demonstrated with Scenario Diagram



3. VDM SPECIFICATION

The Specification of *Three* Identified Classes is as follows:

SmartGridControlSystem

1. Class Specification

types

PowerSignal = <INC_CONSUMPTION> | <DEC_CONSUMPTION> | <INC_GENERATION> | <INC_GENERATION>

DeviceType = <CONSUMER> | <GENERATOR>

Device : : *deviceId : String*
deviceType : DeviceType

init mk- *Device (string devId, DeviceType devType)* $\triangleq$ (*deviceId* = *devId*) $\wedge$ (*DeviceType* = *devType*)

*connectedDevices: Device**

*String = Char**

values

POWER_CONSUMPTION : $\mathbb{Z}$ = 250

POWER_GENERATION : $\mathbb{Z}$ = 500

state *SmartGridControlSystem* **of**

totalConsumption : $\mathbb{Z}$

totalGeneration : $\mathbb{Z}$

connectedDevices : *Device**

— the totalConsumption should always be less than or equal to totalGeneration

inv mk- *SmartGridControlSystem ()* $\triangleq$ *totalConsumption* $\leq$ *totalGeneration*

— both the totalConsumption and totalGeneration are equal to zero and there are no connectedDevices when system is initialized

init mk- *SmartGridControlSystem ()* $\triangleq$ (*totalConsumption* = 0) $\wedge$ (*totalGeneration* = 0)
$\wedge$ (*connectedDevices* = [ ])

end

functions

GCMenu() gcChoice : $\mathbb{Z}$

pre TRUE

— the gcChoice will always be in range [1,4]

post *gcChoice* = *i* $\in$ [1,2,3,4]

takeDeviceIdInput() deviceId : String

pre TRUE

post *deviceId*

takeDeviceTypeInput() deviceType: DeviceType

pre TRUE

post (*deviceType* = <CONSUMER>) ∨ (*deviceType* = < GENERATOR >)

checkUnmatchedInput(choice : ℤ)

pre TRUE

post TRUE

operations

isUnique(String : devId) result : B

ext rd *connectedDevices* : *Device**

pre **len** *connectedDevices* > 0

— for each device in connectedDevices Sequence check whether the Id exists or not

post (*result* = *true* ∧ ∀*i* ∈ **inds** *connectedDevices* • *connectedDevices(i).deviceId* = *devId*) ∨
(*result* = *false* ∧ ∃*i* ∈ **inds** *connectedDevices* • *connectedDevices(i).deviceId* = *devId)*

getDeviceIndex(String : devId) result : ℤ

ext rd *connectedDevices* : *Device**

pre ∃*i* ∈ **inds** *connectedDevices* • *connectedDevices(i).deviceId* = *devId*

— for each device in connectedDevices Sequence find the index corresponding to given Id

post *(result* = *i* ∈ **inds** *connectedDevices* ∧ *connectedDevices(i).deviceId* = *devId)* ∨
(*result* = *-1* ∧ ∃*i* ∉ **inds** *connectedDevices* • *connectedDevices(i).deviceId* = *devId)*

addDevice(String : devId, DeviceType : devType) powSignalOut : PowerSignal

ext wr *connectedDevices : Device**

wr *totalConsumption : ℤ*

wr *totalGeneration : ℤ*

pre (∃*i* ∉ **inds** *connectedDevices* • *connectedDevices(i).deviceId* = *devId*) ∧
((deviceType = <CONSUMER> ∧ *totalGeneration* ≥ *totalConsumption* + *POWER_CONSUMPTION*)
(deviceType = <GENERATOR> ∧ TRUE))

post *connectedDevices* = $\overline{connectedDevices}$ ^ [**mk-Device**(*devId*, *devType*)] ∧
(*powSignalOut* = <INC_CONSUMPTION> ∧ *totalConsumption* = $\overline{totalConsumption}$ + *POWER_CONSUMPTION)*
∧ (*powSignalOut* = <INC_GENERATION> ∧ *totalConsumption* = $\overline{totalConsumption}$ + *POWER_CONSUMPTION)*

removeDevice(index : ℤ) powSignalOut : PowerSignal

ext wr *connectedDevices : Device**

wr *totalConsumption : ℤ*

wr *totalGeneration : ℤ*

pre (*index* ≤ (**len** *connectedDevices*)) ∧ *index* ≥ 0 ∧
((*connectedDevices(index).deviceType* = <GENERATOR>
∧ *totalConsumption* ≤ *totalGeneration* - *POWER_GENERATION*)
∨ (*connectedDevices(index).deviceType* = <CONSUMER> ∧ TRUE))

-- *The sequence comprehension here provides a new sequence that does not contain connectedDevice(index) object*

post *connectedDevices* =

$[\overline{connectedDevices(i)} \mid i \in \textbf{inds}\ \overline{connectedDevices} \bullet \overline{connectedDevices(i)} \neq \overline{connectedDevices(index)}] \wedge$
(*powSignalOut* = <DEC_CONSUMPTION> ∧ *totalConsumption* = $\overline{totalConsumption}$ - *POWER_CONSUMPTION)*
∧ (*powSignalOut* = <DEC_GENERATION> ∧ *totalGeneration* = $\overline{totalGeneration}$ - *POWER_GENERATION)*

displaySystemData ()

ext rd *totalConsumption : ℤ*

rd *totalGeneration : ℤ*

rd *connectedDevices : Device**

pre TRUE

post TRUE

2. Class VDM Translation

a. Types

PowerSignal = <INC_CONSUMPTION> | <DEC_CONSUMPTION> | <INC_GENERATION> | <INC_GENERATION>	`enum PowerSignal { INC_CONSUMPTION,DEC_CONSUMPTION, INC_GENERATION,DEC_GENERATION };`
DeviceType = <CONSUMER> | <GENERATOR>	`enum DeviceType { CONSUMER,GENERATOR };`
Device :: *deviceId : String*	`class Device { public: string deviceId;`

deviceType : DeviceType **init mk-** *Device (string devId, DeviceType devType)* $\triangleq$ $(deviceId = devId) \wedge (DeviceType = devType)$	DeviceType deviceType; public: // Default constructor is mandatory in C++ Device() {} Device(string devId, DeviceType devType) { deviceId = devId; deviceType = devType; } };

b. Values

POWER_CONSUMPTION : $\mathbb{Z} = 250$ *POWER_GENERATION* : $\mathbb{Z} = 500$	const int POWER_CONSUMPTION = 250; const int POWER_GENERATION = 500;

c. State Clause

state *SmartGridControlSystem* **of** *totalConsumption* : $\mathbb{Z}$ *totalGeneration* : $\mathbb{Z}$ *connectedDevices* : *Device**	public: int totalConsumption; int totalGeneration; vector<Device> connectedDevices;

d. Invariant Clause

The ***InvariantCheck*** Abstract Class
(C++ does not provide interfaces however their concept can be implemented by use of *abstract class*. Interfaces can be implemented by inheriting them and then overriding their methods through use override operator, in this way each class can provide its own invariant whilst using the same *interface(abstract class)*)

```
class InvariantCheck {
public:
    virtual bool inv() = 0;
};
```

inv mk- *SmartGridControlSystem ()* $\triangleq$ $totalConsumption \leq totalGeneration$	bool inv() override { bool result = (totalConsumption <= totalGeneration); if(!result) { cout << "Power Balance Invariant Violated" << endl; } return result;}

e. Initialization Clause

init mk- *SmartGridControlSystem ()* $\underline{\Delta}$ $(totalConsumption = 0) \wedge (totalGeneration = 0)$ $\wedge (connectedDevices = [\,])$	`SmartGridControlSystem() {` `  totalConsumption = 0;` `  totalGeneration = 0;` `  connectedDevices = vector<Device>(0);` `  VDM::invTest(*this);` `}`

f. Functions

GCMenu() gcChoice : $\mathbb{Z}$ **pre** TRUE — the gcChoice will always be in range [1,4] **post** *gcChoice* $= i \in [1,2,3,4]$	`int GCMenu() {` `  int gcChoice = 0;` `  cout << endl;` `  cout << "Choose Any one following" << endl;` `  cout << "1 - Display Devices" << endl;` `  cout << "2 - Add Devices" << endl;` `  cout << "3 - Remove Devices" << endl;` `  cout << "4 - Exit" << endl;` `  checkUnmatchedInput(gcChoice);` `  while(gcChoice < 1 ||  gcChoice > 4) {` `  cout << "Please Enter Correct Option (1-4): ";` `  checkUnmatchedInput(gcChoice);` `  }` `  return gcChoice;` `}`

takeDeviceIdInput() deviceId : String **pre** TRUE **post** *deviceId*	`string takeDeviceIdInput() {` `  string deviceId = "";` `  cout << "Enter Device ID: "; cin >> deviceId;` `  return deviceId;` `}`

takeDeviceTypeInput() deviceType: DeviceType **pre** TRUE **post** $(deviceType = \langle \text{CONSUMER} \rangle) \vee$ $(deviceType = \langle \text{GENERATOR} \rangle)$	`DeviceType takeDeviceTypeInput() {` `  int devTypeInt = 0;` `  DeviceType deviceType;` `  cout << "Choose Device Type (1-2)" << endl;` `  cout << "1 - Consumer" << endl;` `  cout << "2 - Generator" << endl;` `  cin >> devTypeInt;` `  while(devTypeInt < 1 && devTypeInt > 2) {` `  cout << "Please Choose Correct Option (1-2)" << endl;` `  }` `  switch (devTypeInt)` `  {` `  case 1:` `  deviceType = DeviceType::CONSUMER;` `  break;` `  case 2:` `  deviceType = DeviceType::GENERATOR;` `  break;`

	default: break; } return deviceType; };

*In VDM, choice: ℤ specifies an integer parameter, and in the C++ code, int &choice represents a reference to an integer. Both are **equivalent**, indicating that changes to the parameter inside the function persist outside it*

checkUnmatchedInput(choice : ℤ) **pre** TRUE **post** TRUE	void checkUnmatchedInput(int &choice) { if (!(cin >> choice)) { cout << "Invalid input. Please enter a number." << endl; cin.clear(); cin.ignore(numeric_limits<streamsize>::max(), '\n'); } }

g. Operations

The ***UniqueExistCheck*** Abstract Class
(C++ does not provide interfaces however their concept can be implemented by use of *abstract class*. Interfaces can be implemented by inheriting them and then overriding their methods through use override operator, in this way each class can provide its own invariant whilst using the same *interface(abstract class)*)

```
class UniqueExistCheck {
public:
    virtual bool isUnique(string Id) = 0;
};
```

isUnique(String : devId) result : B **ext rd** *connectedDevices* : *Device** **pre** **len** *connectedDevices* > 0 **post** (*result* = *true* ∧ ∀*i* ∈ **inds** *connectedDevices* • *connectedDevices(i).deviceId* = *devId*) ∨ (*result* = *false* ∧ ∃*i* ∈ **inds** *connectedDevices* • *connectedDevices(i).deviceId* = *devId*)	bool isUnique(string devId) override { VDM::preTest(connectedDevices.size() >= 0); bool result = true; if(connectedDevices.size() > 0) { for(int i = 0; i < connectedDevices.size(); ++i) { if(connectedDevices[i].deviceId == devId) { result = false; } } } if(!result) { cout << "Device Id: " << devId << " already Exists" << endl; } return result; }

getDeviceIndex(String : devId) result : ℤ	int getDeviceIndex(string devId) {

ext rd *connectedDevices* : *Device**

pre $\exists i \in \textbf{inds}\ connectedDevices \bullet$
connectedDevices(i).deviceId = *devId*

post *(result* = $i \in \textbf{inds}\ connectedDevices \wedge$
connectedDevices(i).deviceId = *devId)* $\vee$
(result = *-1* $\wedge \exists i \notin \textbf{inds}\ connectedDevices \bullet$
connectedDevices(i).deviceId = *devId)*

```
 VDM::preTest(!VDM::uniqueExists(*this, devId));
 // above mean id is not unique and is existant
 int result = -1;
 for(int i = 0; i < connectedDevices.size(); ++i) {
  if(connectedDevices[i].deviceId == devId) {
   result = i;
   cout << "Index for: " << devId << " is: " << i <<
endl;
    return result;
   }
  }
 return result;
}
```

addDevice(String : devId, DevType : deviceType) powSignalOut : PowerSignal

ext wr *connectedDevices : Device**

wr *totalConsumption* : $\mathbb{Z}$

wr *totalGeneration* : $\mathbb{Z}$

pre $(\exists i \notin \textbf{inds}\ connectedDevices \bullet$
connectedDevices(i).deviceId = *devId)* $\wedge$
((deviceType = <CONSUMER> $\wedge$ *totalGeneration* $\geq$ *totalConsumption* + *POWER_CONSUMPTION*)
(deviceType = <GENERATOR> $\wedge$ TRUE))

post *connectedDevices* = $\overline{connectedDevices}$ $\wedge$
[**mk-Device**(*deviceId*, *deviceType*)] $\wedge$
(*powSignalOut* = <INC_CONSUMPTION> $\wedge$ *totalConsumption*= $\overline{totalConsumption}$ + *POWER_CONSUMPTION)*
$\wedge$ (*powSignalOut* = <INC_GENERATION> $\wedge$ *totalConsumption*= $\overline{totalConsumption}$ + *POWER_CONSUMPTION)*

```
PowerSignal addDevice(string devId,DeviceTypedevType)
{

 bool result = (devType == DeviceType::CONSUMER);
 VDM::preTest(VDM::uniqueExists(*this, devId) &&
 ((devType == DeviceType::CONSUMER ?
 (totalGeneration >= totalConsumption + POWER_CONSUMPTION)
  : true)
 ));

 PowerSignal powSignalOut;
 Device device = Device(devId, devType);
 connectedDevices.push_back(device);

 if(devType == DeviceType::CONSUMER) {
  totalConsumption += POWER_CONSUMPTION;
  powSignalOut =  PowerSignal::INC_CONSUMPTION;
 }
 else {
  totalGeneration += POWER_GENERATION;
  powSignalOut =  PowerSignal::INC_GENERATION;
 }

 VDM::invTest(*this);
 return powSignalOut;
}
```

removeDevice(index : $\mathbb{Z}$) powSignalOut : PowerSignal

ext wr *connectedDevices : Device**

wr *totalConsumption* : $\mathbb{Z}$

wr *totalGeneration* : $\mathbb{Z}$

pre (*index* $\leq$ (**len** *connectedDevices*)) $\wedge$ *index* $\geq 0 \wedge$
((*connectedDevices(index).deviceType* = <GENERATOR>

```
PowerSignal removeDevice(int index) {
 VDM::preTest((index <= (connectedDevices.size() - 1)
 && index >= 0) && (
 connectedDevices[index].deviceType ==
 DeviceType::GENERATOR
 ? (totalConsumption <= totalGeneration - POWER_GENERATION)
 : true
 ));

 PowerSignal powSignalOut;

 if(connectedDevices[index].deviceType==DeviceType::CONSUMER)
 {
```

∧ *totalConsumption* ≤ *totalGeneration* - *POWER_GENERATION*) ∨ (*connectedDevices(index).deviceType* = <CONSUMER> ∧ TRUE)) -- *The sequence comprehension here provides a new sequence that does not contain connectedDevice(index) object* **post** *connectedDevices* = [$\overline{connectedDevices(i)}$ | *i* ∈ **inds** $\overline{connectedDevices}$ • $\overline{connectedDevices(i)} \neq \overline{connectedDevices(index)}$] ∧ (*powSignalOut* = <DEC_CONSUMPTION> ∧ *totalConsumption*= $\overline{totalConsumption}$ - *POWER_CONSUMPTION)* ∧ (*powSignalOut* = <DEC_GENERATION> ∧ *totalGeneration*= $\overline{totalGeneration}$ - *POWER_GENERATION)*	totalConsumption -= POWER_CONSUMPTION; powSignalOut = PowerSignal::DEC_CONSUMPTION; } else { totalGeneration -= POWER_GENERATION; powSignalOut = PowerSignal::DEC_GENERATION; } connectedDevices.erase(connectedDevices.begin() + index); VDM::invTest(*this); return powSignalOut; }

displaySystemData () **ext** **rd** *totalConsumption* : ℤ **rd** *totalGeneration* : ℤ **rd** *connectedDevices* : *Device** **pre** TRUE **post** TRUE	void displaySystemData() { cout << endl; if(connectedDevices.size() > 0) { cout << setw(20) << "Device Num" << setw(20) << "Device ID" << setw(20) << "DeviceType" << setw(20) << endl; for(int i = 0; i < connectedDevices.size(); ++i) { cout << setw(20) << i+1 \<< setw(20) << connectedDevices[i].deviceId \<< setw(20) << (connectedDevices[i].deviceType == DeviceType::GENERATOR ? "Generator" : "Consumer") \<< endl; } cout << "-----------" << endl; cout << "Total Consumption: " << totalConsumption << endl; cout << "Total Generation: " << totalGeneration << endl; } else { cout << "No Devices Connected Yet" << endl; } }

3. Class Code

`SmartGridControlSystem` uses function from InvariantCheck and UniqueExistCheck abstract class by inheriting and thus enabling user to write their own overriding implementation.

It also inherits methods from VDM class.

This allow C++ to access functions of class without creating object through use of scope Resolution (**::**) operator. This also helps in call VDM invariantTest and preTest functions.

Note: VDM and VDMException User Defined Classes can be found at end of this report

```
class InvariantCheck {
public:
    virtual bool inv() = 0;
};
class UniqueExistCheck {
public:
    virtual bool isUnique(string Id) = 0;
};

enum DeviceType {
    CONSUMER,
    GENERATOR
};
enum PowerSignal {
  INC_CONSUMPTION,
  DEC_CONSUMPTION,
  INC_GENERATION,
  DEC_GENERATION
};

class Device {
  public:
    string deviceId;
    DeviceType deviceType;

  public:
    Device() {}
    Device(string devId, DeviceType devType) {
      deviceId = devId;
      deviceType = devType;
    }
};


void checkUnmatchedInput(int &choice) {
  if (!(cin >> choice)) {

    cout << "Invalid input. Please enter a number." << endl;

    cin.clear();

    cin.ignore(numeric_limits<streamsize>::max(), '\n');
 }
```

```
}

int GCMenu() {


  int gcChoice = 0;
  cout << endl;
  cout << "Choose Any one following" << endl;
  cout << "1 - Display Devices" << endl;
  cout << "2 - Add Devices" << endl;
  cout << "3 - Remove Devices" << endl;
  cout << "4 - Exit" << endl;

  checkUnmatchedInput(gcChoice);

  while(gcChoice < 1 ||  gcChoice > 4) {
    cout << "Please Enter Correct Option (1-4): ";
    checkUnmatchedInput(gcChoice);
  }

  return gcChoice;
}

string takeDeviceIdInput() {
  string deviceId = "";
  cout << "Enter Device ID: "; cin >> deviceId;
  return deviceId;
}

DeviceType takeDeviceTypeInput() {
  int devTypeInt = 0;
  DeviceType deviceType;

  cout << "Choose Device Type (1-2)" << endl;
  cout << "1 - Consumer" << endl;
  cout << "2 - Generator" << endl;
  cin >> devTypeInt;

  while(devTypeInt < 1 && devTypeInt > 2) {
    cout << "Please Choose Correct Option (1-2)" << endl;
  }

  switch (devTypeInt)
  {
    case 1:
        deviceType = DeviceType::CONSUMER;
      break;
    case 2:
        deviceType = DeviceType::GENERATOR;
      break;
    default:
      break;
  }
  return deviceType;
};
```

```
// Inheriting in order to call functions

class SmartGridControlSystem : public VDM, public InvariantCheck, public UniqueExistCheck {
  public:
    const int POWER_CONSUMPTION = 250;
    const int POWER_GENERATION = 500;
    int totalConsumption;
    int totalGeneration;
    vector<Device> connectedDevices;

  public:
    SmartGridControlSystem() {
      connectedDevices = vector<Device>(0);
      totalGeneration = 0;
      totalConsumption = 0;
      VDM::invTest(*this);
    }

    bool inv() override {
        bool result = (totalConsumption <= totalGeneration);
        if(!result) {
          cout << "Power Balance Invariant Violated" << endl;
        }
        return result;
    }

    bool isUnique(string devId) override {
      VDM::preTest(connectedDevices.size() >= 0);
      bool result = true;
      if(connectedDevices.size() > 0) {
        for(int i = 0; i < connectedDevices.size(); ++i) {
          if(connectedDevices[i].deviceId == devId) {
            result = false;
          }
        }
      }
      if(!result) {
        cout << "Device Id: " << devId << " already Exists" << endl;
      }
      return result;
    }



    int getDeviceIndex(string devId) {
      VDM::preTest(!VDM::uniqueExists(*this, devId));
      // above mean id is not unique and is existant
      int result = -1;
      for(int i = 0; i < connectedDevices.size(); ++i) {
        if(connectedDevices[i].deviceId == devId) {
          result = i;
          cout << "Index for: " << devId << " is: " << i << endl;
          return result;
        }
      }
      return result;
    }

    PowerSignal addDevice(string devId, DeviceType devType) {

      bool result = (devType == DeviceType::CONSUMER);
```

```
      // below mean id is (unique and is non-existant)
      VDM::preTest(VDM::uniqueExists(*this, devId) &&
      (
        (devType == DeviceType::CONSUMER
        ? (totalGeneration >= totalConsumption + POWER_CONSUMPTION)
        : true)
      ));

      PowerSignal powSignalOut;
      Device device = Device(devId, devType);
      connectedDevices.push_back(device);

      if(devType == DeviceType::CONSUMER) {
        totalConsumption += POWER_CONSUMPTION;
        powSignalOut =  PowerSignal::INC_CONSUMPTION;
      }
      else {
        totalGeneration += POWER_GENERATION;
        powSignalOut =  PowerSignal::INC_GENERATION;
      }

      VDM::invTest(*this);
      return powSignalOut;
    }

    PowerSignal removeDevice(int index) {
      VDM::preTest((index <= (connectedDevices.size() - 1) && index >= 0) && (
        connectedDevices[index].deviceType == DeviceType::GENERATOR
        ? (totalConsumption <= totalGeneration - POWER_GENERATION)
        : true
      ));

      PowerSignal powSignalOut;

      if(connectedDevices[index].deviceType == DeviceType::CONSUMER) {
        totalConsumption -= POWER_CONSUMPTION;
        powSignalOut = PowerSignal::DEC_CONSUMPTION;
      }
      else {
        totalGeneration -= POWER_GENERATION;
        powSignalOut = PowerSignal::DEC_GENERATION;
      }
      connectedDevices.erase(connectedDevices.begin() + index);
      VDM::invTest(*this);
      return powSignalOut;
    }

    void displaySystemData() {
      cout << endl;
      if(connectedDevices.size() > 0) {

        cout << setw(20) << "Device Num" << setw(20) << "Device ID" << setw(20) << "DeviceType" <<
setw(20) << endl;
        for(int i = 0; i < connectedDevices.size(); ++i) {
          cout << setw(20) << i+1
               << setw(20) << connectedDevices[i].deviceId
               << setw(20) << (connectedDevices[i].deviceType == DeviceType::GENERATOR ? "Generator" :
"Consumer" )
               << endl;
        }
```

```
        cout << "-------------------------------------------------------------------------------" <<
endl;
        cout << "Total Consumption: " << totalConsumption << endl;
        cout << "Total Generation: " << totalGeneration << endl;

        }
      else {
        cout << "No Devices Connected Yet" << endl;
      }
    }
}
```

4. C++ DRIVER CODE

```
int main() {

  SmartGridControlSystem gC = SmartGridControlSystem();

  cout << endl; cout << "-------------" << endl; cout << endl;
  cout << "SMART GRID CONTROL SYSTEM v1.0" << endl;
  cout << "Choose from following" << endl;

  int GCMenuChoice =  0;

  while(GCMenuChoice != 4) {
    GCMenuChoice = GCMenu();
    switch (GCMenuChoice) {
      case 1: {
        gC.displaySystemData(); break;
      }
      case 2: {
        string deviceId = takeDeviceIdInput();
        DeviceType deviceType = takeDeviceTypeInput();

        try {
          gC.addDevice(deviceId,deviceType);
        } catch (VDMException &ex) {
          cout << "**********************************************************************" << endl;
          cout << ex.getMessage() << endl;
          std::cerr << "Exception caught at line: " << __LINE__ << endl;
          cout << "**********************************************************************" << endl;
        }
        break;
      }
      case 3:
        if(gC.connectedDevices.size() > 0) {
          string deviceId = takeDeviceIdInput();
          bool deviceIdIsExisting = gC.isUnique(deviceId);

          if(deviceIdIsExisting == false) {
            int index;
              index = gC.getDeviceIndex(deviceId);
            try {
              gC.removeDevice(index);
            } catch (VDMException &ex) {
              std::cerr << "Exception caught at line: " << __LINE__ << endl;
              cout << ex.getMessage() << endl;
            }
          }
          else {
            cout << "Device Id Does not Exist Please Enter a Unique Id" << endl;
          }

          break;
        }
        else {
          cout << "No Devices Added Yet" << endl;
        }
        case 4:
```

```
            break;
        }
    }


  return 0;
}
```

5. TESTING CLASS

5.1 Test Class and Driver Code

```
class SGCSTest {
  private:
    vector<string> input;
    vector<string> expectedOutput;
    vector<string> actualOutput;
    vector<string> status;
    vector<string> testName;
    SmartGridControlSystem sgcs;
    const string PASSED = "|        PASSED        |";
    const string FAILED = "|        FAILED        |";
  public:
    SGCSTest() : sgcs(SmartGridControlSystem()) {
        input =  vector<string>(0);
        expectedOutput =  vector<string>(0);
        actualOutput =  vector<string>(0);
        status =  vector<string>(0);
        testName =  vector<string>(0);

    }
    void deviceIdisUnique_test_1() {
      // Test: 1 -> 'D-1', device added, device added, passed give existingId and check existence
      sgcs.addDevice("D-1", DeviceType::GENERATOR);
      testName.push_back("Existing-ID & Check Presence");
      string inp = "D-1";
      string expOut = "|     ID Not Unique    |";
      bool result = sgcs.isUnique(inp);
      input.push_back("|          " + inp + "          |");
      expectedOutput.push_back(expOut);
      if(result == false) {
        actualOutput.push_back(expOut);
        status.push_back(PASSED);
      }
      else {
        actualOutput.push_back("|       Id Unique      |");
        status.push_back(FAILED);
      }
    }
    void deviceIdisUnique_test_2() {
```

```
    // Test: 2 -> 'D-2'  give Non-existingId and check absence
    testName.push_back("Non-Existing-ID & Check Absence");
    string inp = "D-2";
    string expOut = "|       ID Unique      |";
    bool result = sgcs.isUnique(inp);
    input.push_back("|          " + inp + "          |");
    expectedOutput.push_back(expOut);
    if(result == true) {
      actualOutput.push_back(expOut);
      status.push_back(PASSED);
    }
    else {
      actualOutput.push_back("|     ID Not Unique     ");
      status.push_back(FAILED);
    }
  }
  void getDeviceIndex_test_1() {
    // Test: 1 -> returns -1 for non-existing Id
    testName.push_back("Non-Existing-ID & Invalid index returned");
    string inp = "D-2";
    string expOut = "|Invalid Index Returned|";

    string actOutput = "| Valid Index Returned |";
    string stat = FAILED;


    input.push_back("|          " + inp + "          |");
    expectedOutput.push_back(expOut);
    try {
      bool result = sgcs.getDeviceIndex(inp);


    } catch (VDMException &ex) {
      actOutput = expOut;
      stat = PASSED;
    }

    actualOutput.push_back(actOutput);
    status.push_back(stat);
  }
  void getDeviceIndex_test_2() {
    // Test: 1 -> returns -1 for non-existing Id
    testName.push_back("Existing-ID & returned valid index");
    string inp = "D-1";
    string expOut = "| Valid Index Returned |";

    string actOutput = "";
    string stat = "";

    input.push_back("|          " + inp + "          |");
    expectedOutput.push_back(expOut);
    try {
      bool result = sgcs.getDeviceIndex(inp);
      actOutput = expOut;
      stat = PASSED;
    } catch (VDMException &ex) {
      actOutput = "|Invalid Index Returned|";
      stat = FAILED;
    }

    actualOutput.push_back(actOutput);
```

```
    status.push_back(stat);
  }
  void addDevice_Test_1() {
    // Test: 1 -> 1st Consumer => powerSignal: increaseConsumption
    testName.push_back("For 1 Generaor Add 1st Consumer");

    string expOut = "|    INC_CONSUMPTION   |";
    string actOutput = "";
    string stat = "";
    PowerSignal pS;

    input.push_back("|    D-2, CONSUMER     |");
    expectedOutput.push_back(expOut);

    try {
      pS = sgcs.addDevice("D-2", DeviceType::CONSUMER);
    } catch (VDMException &ex) {
      actOutput = "|      No Signal       |";
      stat = FAILED;
    }
    if(pS == PowerSignal::INC_CONSUMPTION) {
      actOutput = expOut;
      stat = PASSED;
    }
    actualOutput.push_back(actOutput);
    status.push_back(stat);
  }

  void addDevice_Test_2() {
  // Test: 2 -> 3rd Consumer Addition for 1 Generator: invariant Violated
    sgcs.addDevice("D-3", DeviceType::CONSUMER);
    testName.push_back("For 1 Generator Add 3rd Consumer");

    string expOut = "|      No Signal       |";
    string actOutput = "";
    string stat = "";
    PowerSignal pS = PowerSignal::DEC_CONSUMPTION;

    input.push_back("|    D-4, CONSUMER     |");
    expectedOutput.push_back(expOut);

    try {
      pS = sgcs.addDevice("D-4", DeviceType::CONSUMER);
    } catch (VDMException &ex) {

      cout << "Hello" << endl;
      cout << pS << endl;
      actOutput = expOut;
      stat = PASSED;
    }
    if(pS == PowerSignal::INC_CONSUMPTION) {
      cout << "INC_CONS RUN" << endl;
      actOutput = "|   INC_CONSUMPTION    |";
      stat = FAILED;
    }
    actualOutput.push_back(actOutput);
    status.push_back(stat);
  }

  void addDevice_Test_3() {
    // Test: 3 -> 2nd Generator => powerSignal: increaseGeneration
```

```
    testName.push_back("Adding 2nd Generator");

    string expOut = "|    INC_GENERATION    |";
    string actOutput = "";
    string stat = "";
    PowerSignal pS;

    input.push_back("|    D-4, GENERATOR    |");
    expectedOutput.push_back(expOut);

    try {
      pS = sgcs.addDevice("D-4", DeviceType::GENERATOR);
    } catch (VDMException &ex) {
      actOutput = "|      No Signal       |";
      stat = FAILED;
    }
    if(pS == PowerSignal::INC_GENERATION) {
      actOutput = expOut;
      stat = PASSED;
    }
    actualOutput.push_back(actOutput);
    status.push_back(stat);
  }
  void removeDevice_Test_1() {
    // Test: 1 -> 2nd Generator remove => powerSignal: decreaseGeneration
    testName.push_back("Removing 2nd Generator");

    string expOut = "|    DEC_GENERATION    |";
    string actOutput = "";
    string stat = "";
    PowerSignal pS;

    int index = sgcs.getDeviceIndex("D-4");
    input.push_back("|      Index: " + to_string(index) + "        |");
    expectedOutput.push_back(expOut);

    try {
      pS = sgcs.removeDevice(index);
    } catch (VDMException &ex) {
      actOutput = "|      No Signal       |";
      stat = FAILED;
    }
    if(pS == PowerSignal::DEC_GENERATION) {
      actOutput = expOut;
      stat = PASSED;
    }
    actualOutput.push_back(actOutput);
    status.push_back(stat);
  }
  void removeDevice_Test_2() {
    // Test: 2 -> 2nd Consumer remove => powerSignal: decreaseConsumption
    testName.push_back("Removing 2nd Consumer");

    string expOut = "|   DEC_CONSUMPTION    |";
    string actOutput = "";
    string stat = "";
    PowerSignal pS;

    int index = sgcs.getDeviceIndex("D-3");
    input.push_back("|      Index: " + to_string(index) + "        |");
    expectedOutput.push_back(expOut);
```

```
      try {
        pS = sgcs.removeDevice(index);
      } catch (VDMException &ex) {
        actOutput = "|       No Signal      |";
        stat = FAILED;
      }
      if(pS == PowerSignal::DEC_CONSUMPTION) {
        actOutput = expOut;
        stat = PASSED;
      }
      actualOutput.push_back(actOutput);
      status.push_back(stat);
    }
    void removeDevice_Test_3() {
      // Test: 3 -> 1st Generator remove => no_Signal
     testName.push_back("Remove 1st Generator in presence of 1st Consumer");

      string expOut = "|       No Signal      |";
      string actOutput = "";
      string stat = "";
      PowerSignal pS;

      int index = sgcs.getDeviceIndex("D-1");
      input.push_back("|      Index: " + to_string(index) + "        |");
      expectedOutput.push_back(expOut);

      try {
        pS = sgcs.removeDevice(index);
      } catch (VDMException &ex) {
        actOutput = expOut;
        stat = PASSED;
      }
      if(pS == PowerSignal::DEC_GENERATION) {
        actOutput = expOut;
        stat = FAILED;
      }
      actualOutput.push_back(actOutput);
      status.push_back(stat);
    }
    void executeTests() {
      this->deviceIdisUnique_test_1();
      this->deviceIdisUnique_test_2();
      this->getDeviceIndex_test_1();
      this->getDeviceIndex_test_2();
      this->addDevice_Test_1();
      this->addDevice_Test_2();
      this->addDevice_Test_3();
      this->removeDevice_Test_1();
      this->removeDevice_Test_2();
      this->removeDevice_Test_3();

      this->displayTestResult();
    }
    void displayTestResult() {
      cout <<
"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++" << endl;
      cout << "                                                    Test
Result                                              "  << endl;
```

```
      cout <<
"++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++" << endl << endl;

      cout << "-----------------------------------------------------------------------------------------------
--------------------------" << endl;
      cout << left << setw(30) << "|        Input          |"
              << setw(30) << "|   Expected Output    |" << setw(30) << "|    Actual Output     |"
              << setw(30) << "|        STATUS        |" << endl ;
      cout << "-----------------------------------------------------------------------------------------------
--------------------------" << endl<< endl;
    // Table content
    for (int i = 0; i < status.size(); ++i) {
        cout << "=> " << testName[i] << endl;
        cout << "-----------------------------------------------------------------------------------------------
--------------------------" << endl;
        cout  << left << setw(30) << input[i]
                  << setw(30) << expectedOutput[i] << setw(30) << actualOutput[i]
                  << setw(30) << status[i] << endl;
         cout << "-----------------------------------------------------------------------------------------------
--------------------------" << endl << endl;
    }
    }
};


int main() {

  SGCSTest testSuite = SGCSTest();
  Suite.executeTests();

  return 0;
}
```

5.2. Testing Output

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                                                    Test Result
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

-------------------------------------------------------------------------------------------------------------------------------------------------
|        Input         |      |   Expected Output    |      |    Actual Output     |      |        STATUS        |
-------------------------------------------------------------------------------------------------------------------------------------------------

=> Existing-ID & Check Presence
-------------------------------------------------------------------------------------------------------------------------------------------------
|          D-1         |      |     ID Not Unique    |      |     ID Not Unique    |      |        PASSED        |
-------------------------------------------------------------------------------------------------------------------------------------------------

=> Non-Existing-ID & Check Absence
-------------------------------------------------------------------------------------------------------------------------------------------------
|          D-2         |      |       ID Unique      |      |       ID Unique      |      |        PASSED        |
-------------------------------------------------------------------------------------------------------------------------------------------------

=> Non-Existing-ID & Invalid index returned
-------------------------------------------------------------------------------------------------------------------------------------------------
|          D-2         |      |Invalid Index Returned|      |Invalid Index Returned|      |        PASSED        |
-------------------------------------------------------------------------------------------------------------------------------------------------

=> Existing-ID & returned valid index
-------------------------------------------------------------------------------------------------------------------------------------------------
|          D-1         |      | Valid Index Returned |      | Valid Index Returned |      |        PASSED        |
-------------------------------------------------------------------------------------------------------------------------------------------------

=> For 1 Generaor Add 1st Consumer
-------------------------------------------------------------------------------------------------------------------------------------------------
|    D-2, CONSUMER     |      |    INC_CONSUMPTION   |      |    INC_CONSUMPTION   |      |        PASSED        |
-------------------------------------------------------------------------------------------------------------------------------------------------

=> For 1 Generator Add 3rd Consumer
-------------------------------------------------------------------------------------------------------------------------------------------------
|    D-4, CONSUMER     |      |      No Signal       |      |      No Signal       |      |        PASSED        |
-------------------------------------------------------------------------------------------------------------------------------------------------
=> Removing 2nd Generator
-------------------------------------------------------------------------------------------------------------------------------------------------
|      Index: 3        |      |    DEC_GENERATION    |      |    DEC_GENERATION    |      |        PASSED        |
-------------------------------------------------------------------------------------------------------------------------------------------------

=> Removing 2nd Consumer
-------------------------------------------------------------------------------------------------------------------------------------------------
|      Index: 2        |      |   DEC_CONSUMPTION    |      |   DEC_CONSUMPTION    |      |        PASSED        |
-------------------------------------------------------------------------------------------------------------------------------------------------

=> Remove 1st Generator in presence of 1st Consumer
-------------------------------------------------------------------------------------------------------------------------------------------------
|      Index: 0        |      |       No Signal      |      |       No Signal      |      |        PASSED        |
-------------------------------------------------------------------------------------------------------------------------------------------------
```

6. USER DEFINED CLASSES

User Defined Classes for VDM

The VDM class provides methods for invariant test, pre condition test and isUnique test

```
#pragma once
#include <string>
#include <iostream>
#include <stdexcept>
using namespace std;

class VDMException : public exception{
private:
    std::string message;

public:
    VDMException(std::string msg) : message(msg) {}

    std::string& getMessage() {
        return message;
    }
};


class VDM {
public:
    template <typename T>
    void invTest(T& sysObject) {
        if (!sysObject.inv()) {
            throw VDMException("VDMException Invariant Violation");
        }
    }

    void preTest(bool preCondition) {
        if (!preCondition) {
            throw VDMException("VDMException PreTest Violation");
        }
    }

    template <typename T>
    bool uniqueExists(T& sysObject, string Id) {
      return sysObject.isUnique(Id);
    }

};
```