**OO Analysis and Design**

**COVER SHEET**

**Thomas O Halloran – R00050862**

**Team Members**

**Darren Smith - R00117899**

**Dave Kavanagh – R00013469**

**DCOM2**

**16/05/2015**

Table of Contents

[1. Part A 3 - 12](#_Toc12011599)

[1.1 Q1 (ii) 3](#_Toc12011600)

[1.2 Q1 (ii) 3](#_Toc12011600)

[1.3 Q1 (iii) 4 - 7](#_Toc12011600)

[1.4 Q1 (iv) 7](#_Toc12011600)

[1.5 Q1 (v) 8 - 11](#_Toc12011600)

[1.6 Q1 (vi) 12](#_Toc12011600)

[2. Part B 12 - 20](#_Toc12011601)

[2.1 Q1 (i) 12](#_Toc12011602)

[2.2 Q1 (ii) 13](#_Toc12011602)

[2.3 Q1 (iii) 14](#_Toc12011602)

[2.4 Q1 (iv) 15](#_Toc12011602)

[2.5 Q2 15 - 20](#_Toc12011602)

**Part A**

Q1 (i)

List All Teams

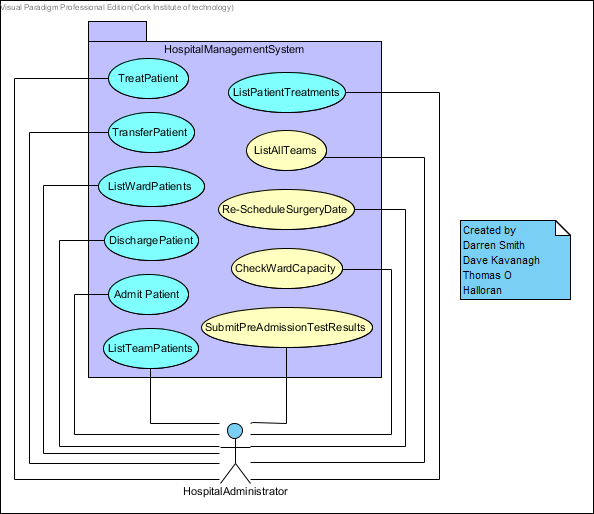
Re-Schedule Surgery Date

Check Ward Capacity

Submit Pre Admission Test Results

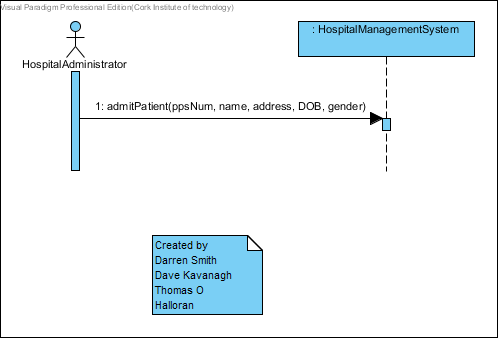
Q1 (ii)

Use Case Diagram

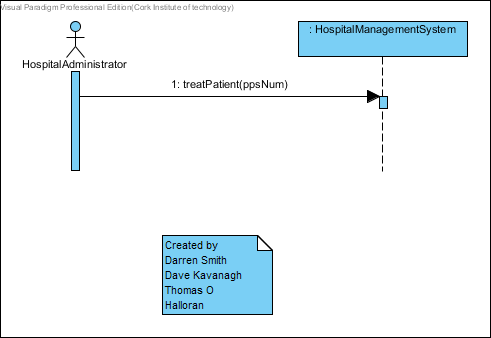


Q1 (iii)

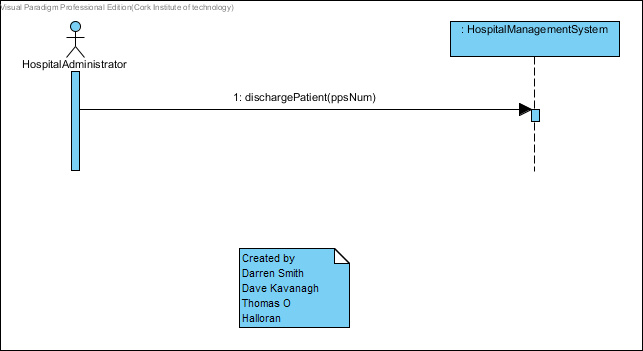
Admit Patient



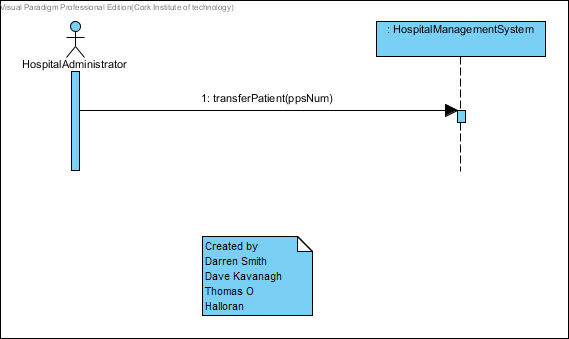
Treat Patient



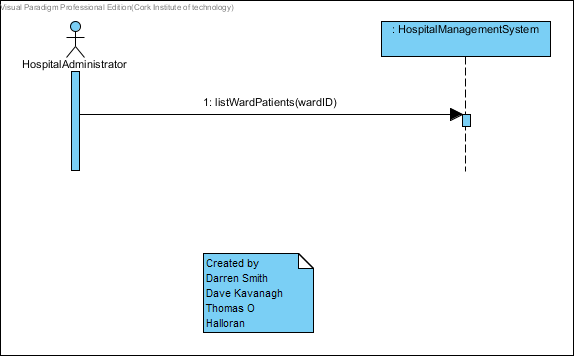
Discharge Patient



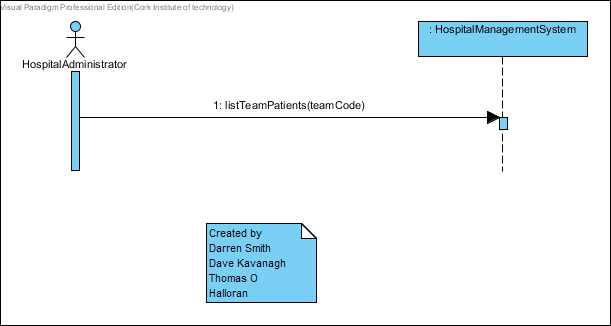
Transfer Patient



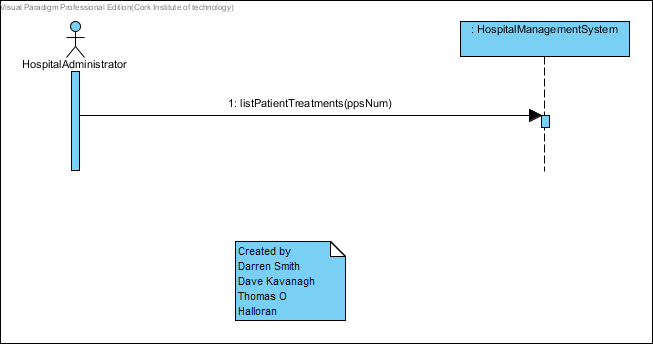
List Ward Patients



List Team Patients

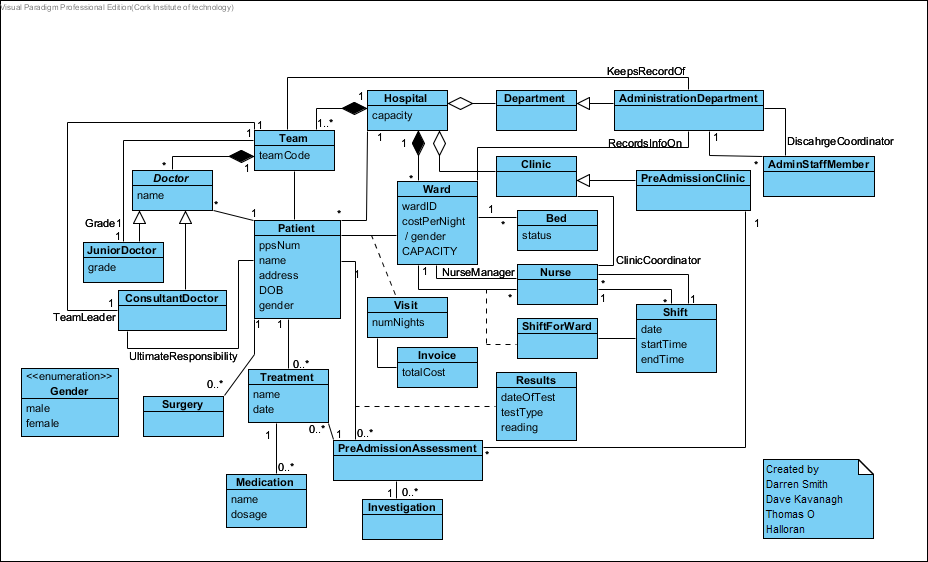


List Patient Treatments



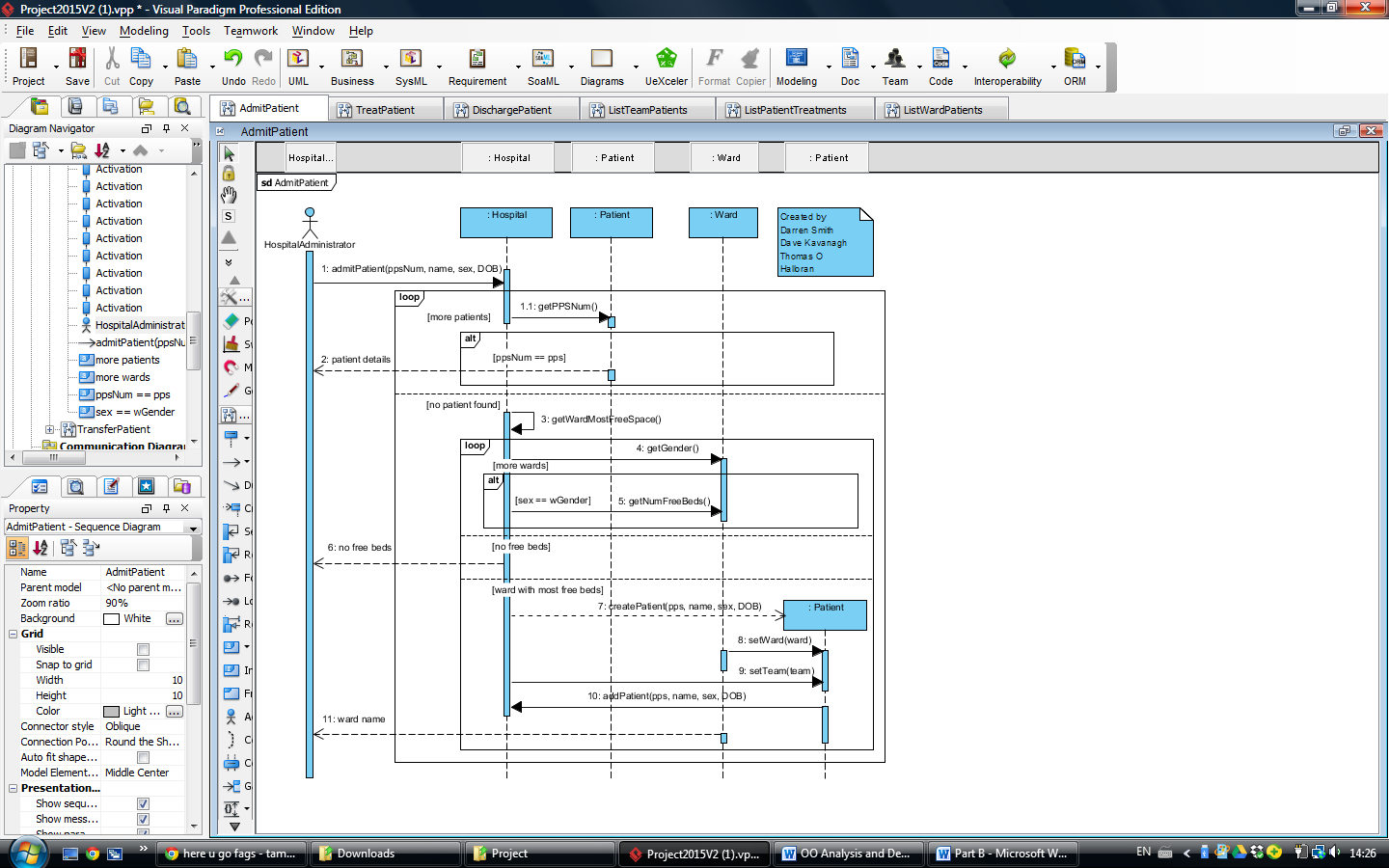
Q1 (iv)

Domain Model

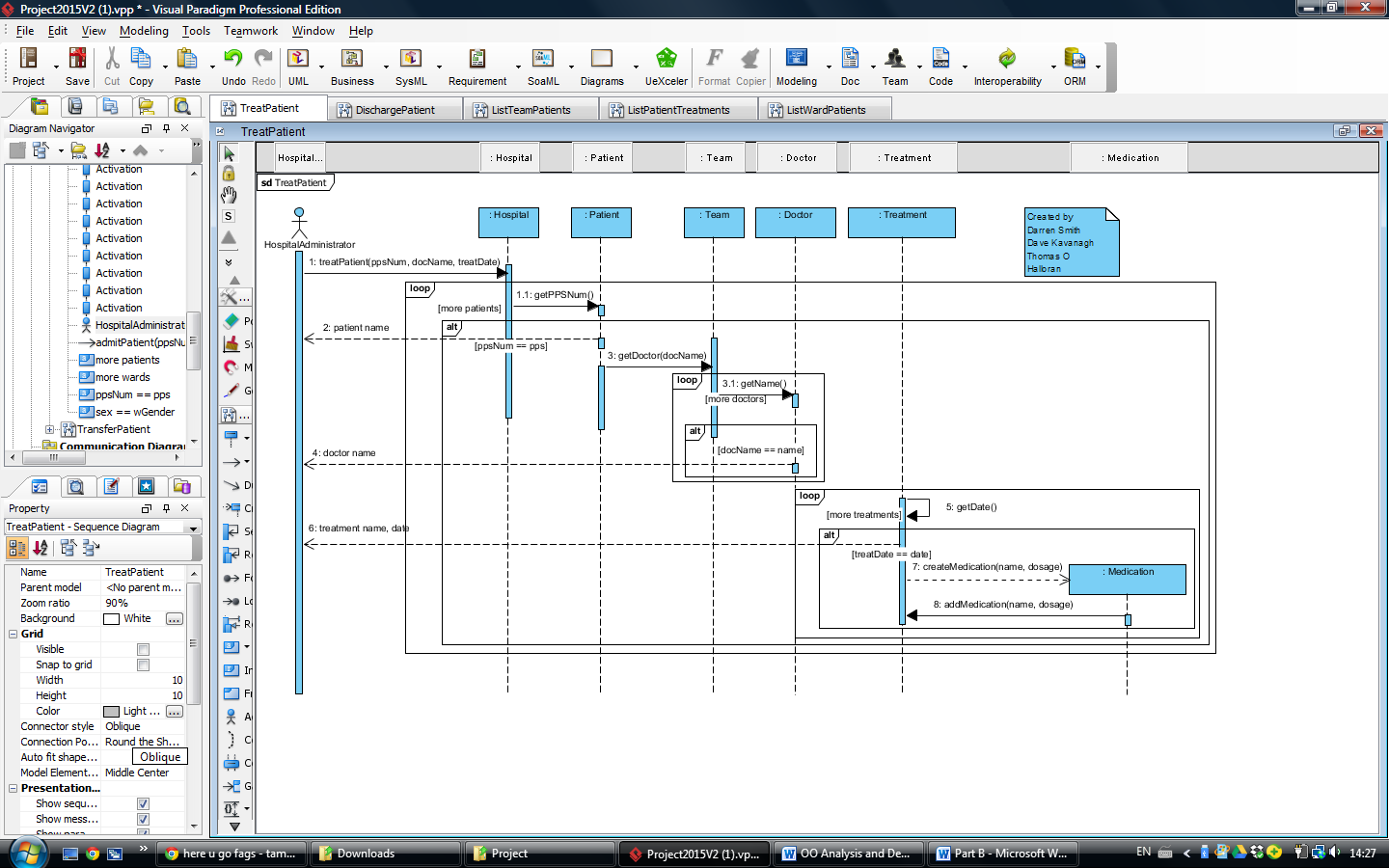


Q1 (v)

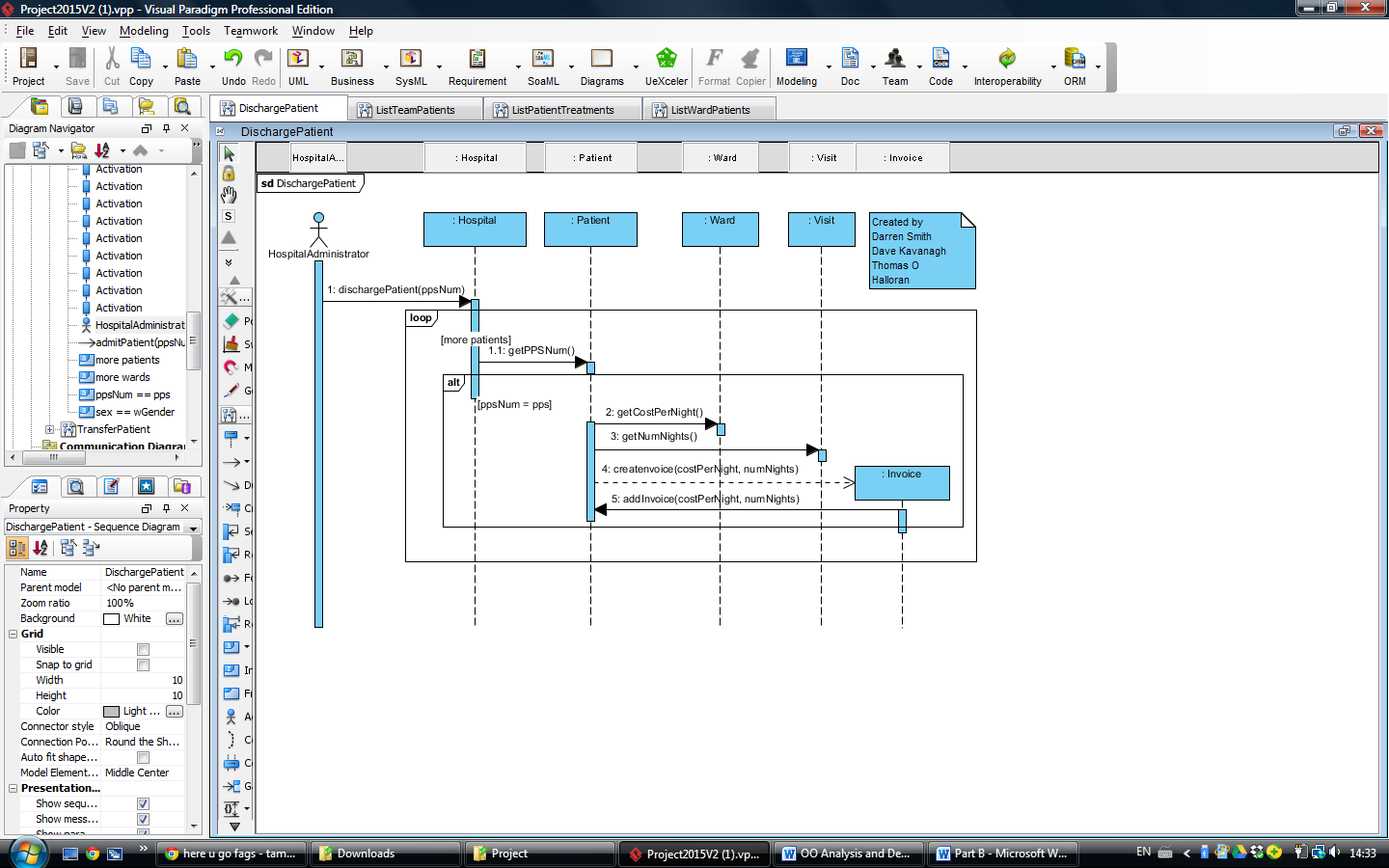
Admit Patient



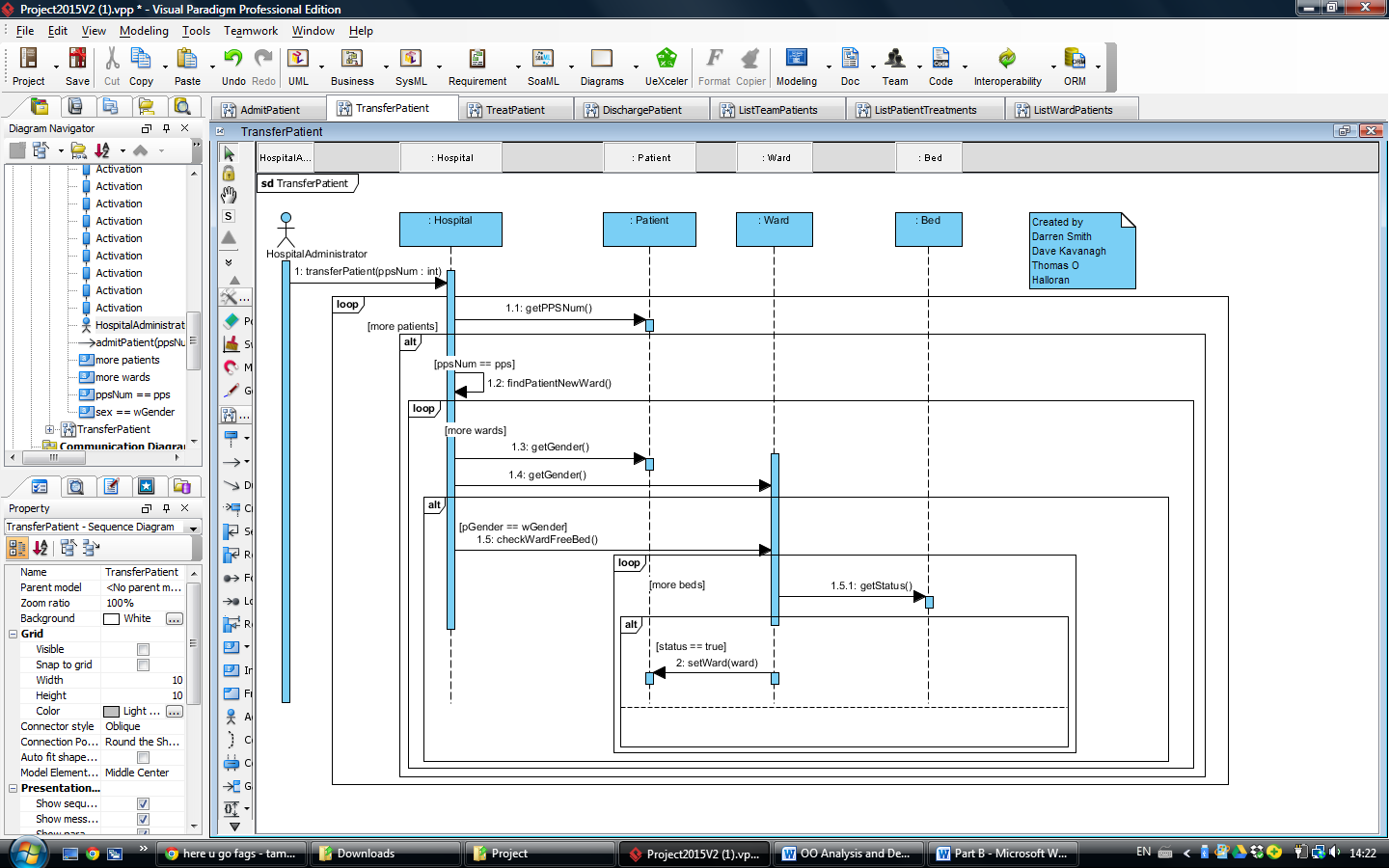
Treat Patient



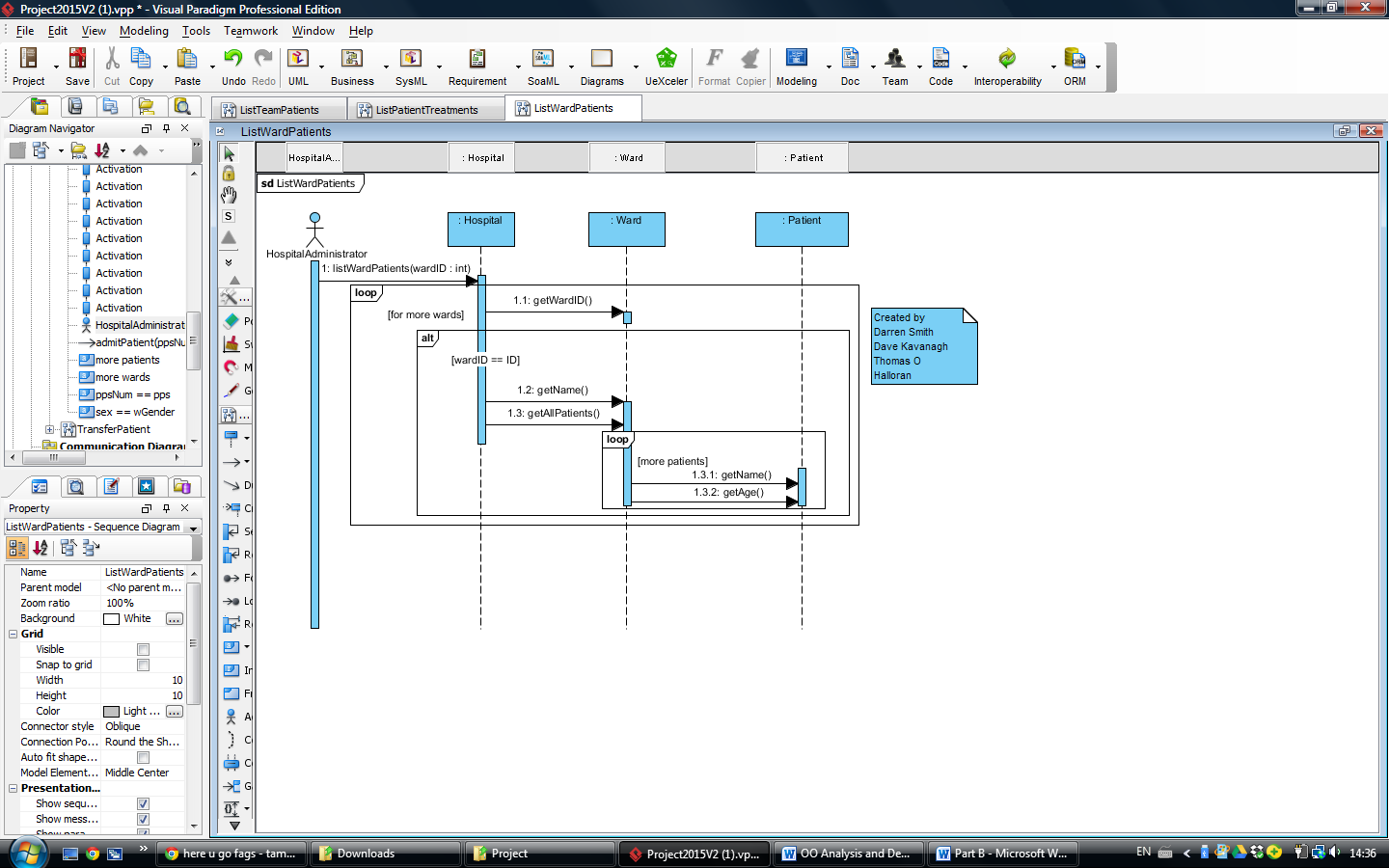
Discharge Patient



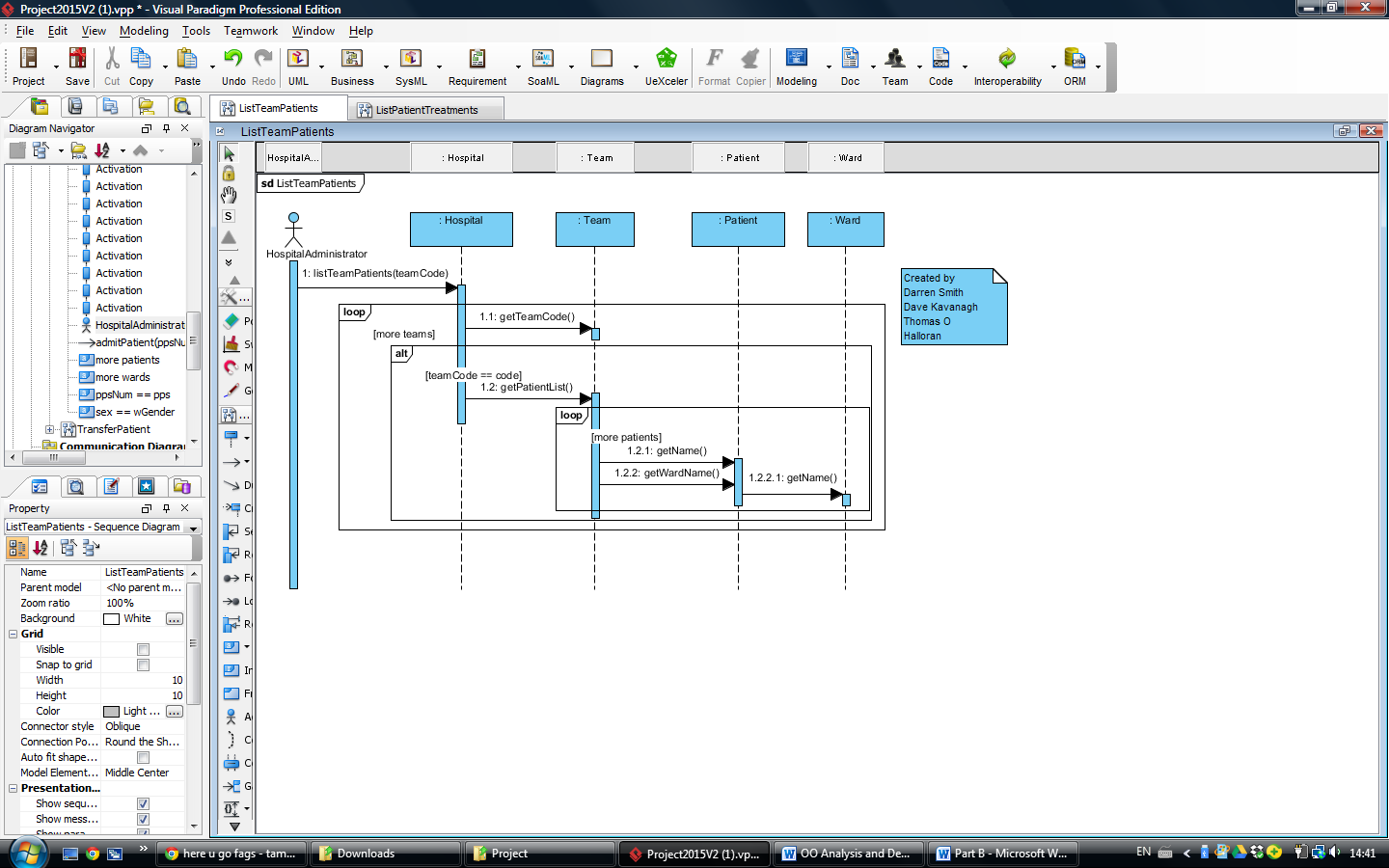
Transfer Patient



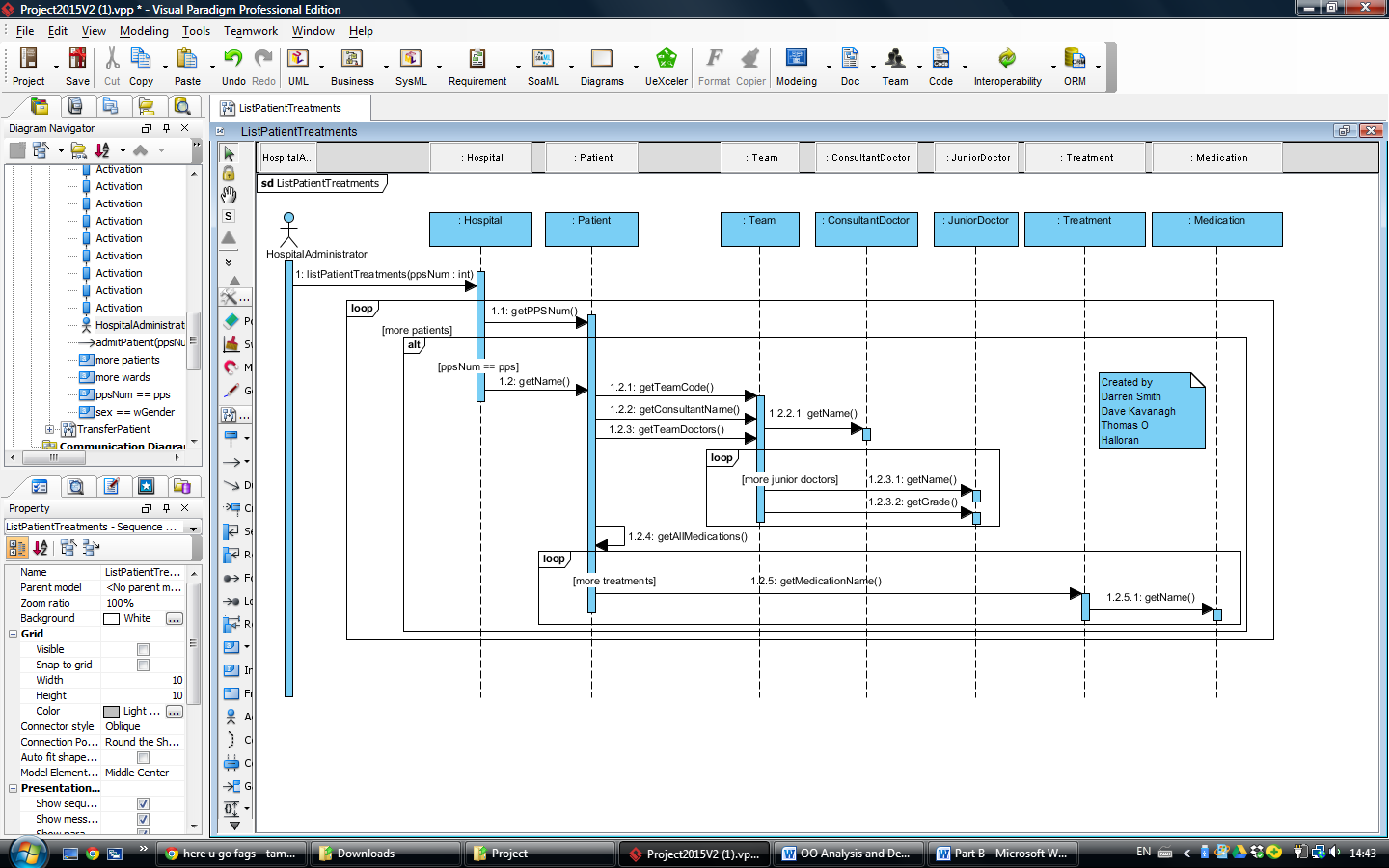
List Ward Patients



List Team Patients

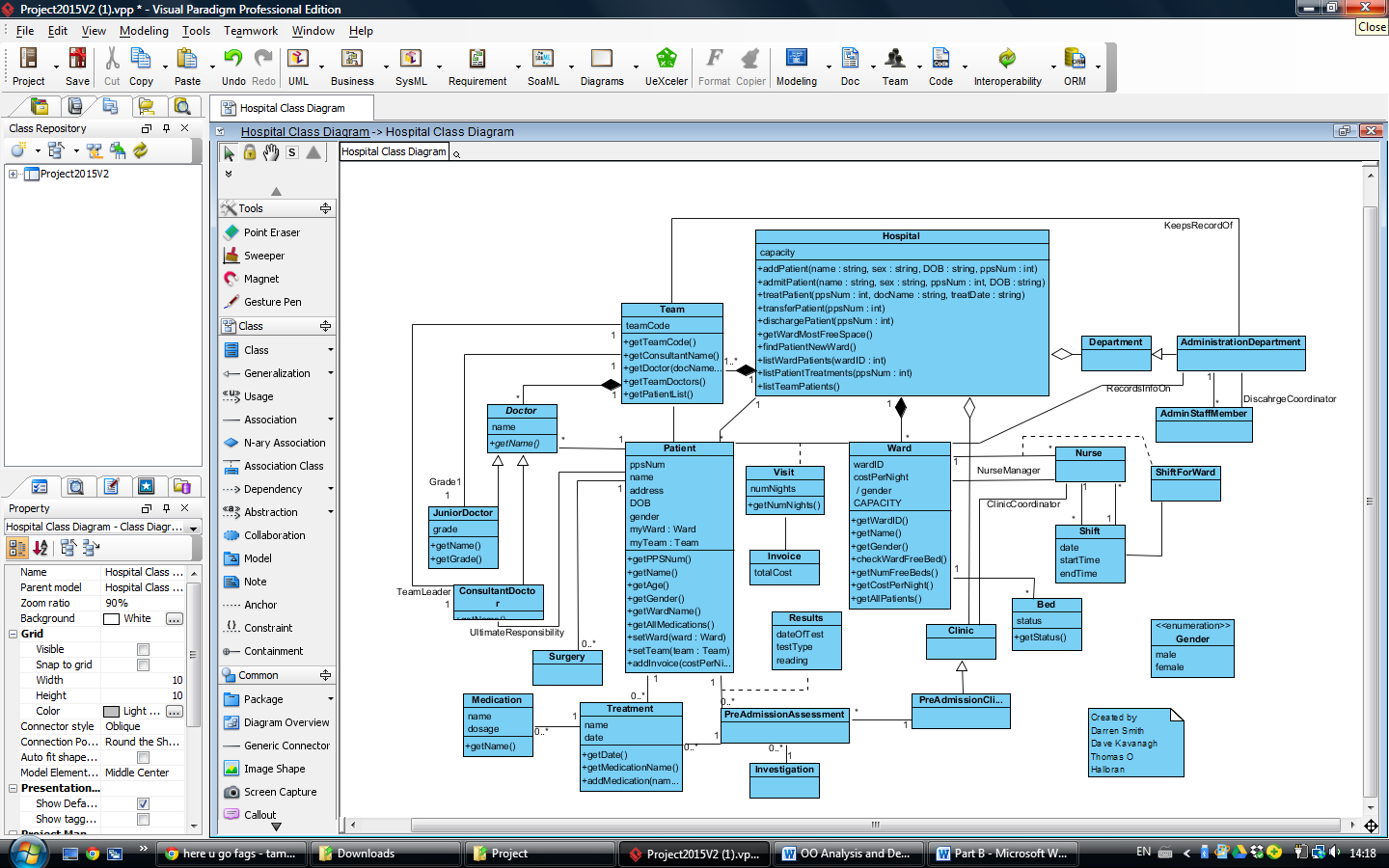


List Patient Treatments



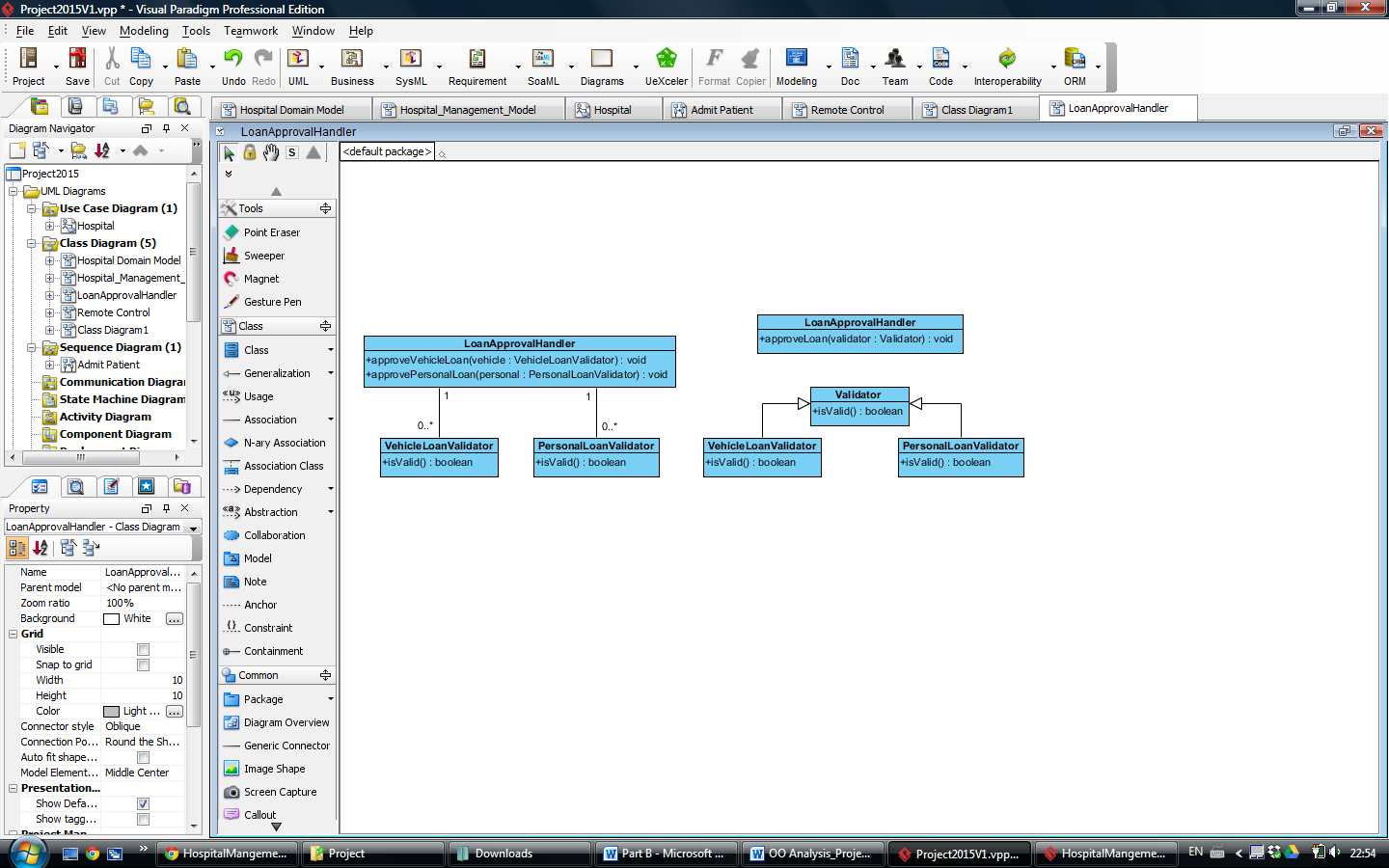
Q1 (vi)

Design Class Diagram



**Part B**

Q1 (i)



Q1 (ii)

The code violates the OCP( Open for extension, Closed for modification) principle because if we add in other types of loans we must modify the code for the LoanApprovalHandler class to handle these types of new loans. If the LoanApprovalHandler class has to be modified to accommodate new loan types then it is not well designed and it violates the OCP principle.

For example if we add a new loan validator for say house loans.

**public class HouseLoanValidator**

**{**

public boolean isValid()

{

//Validation logic

}

}

Then we would have to modify the LoanApprovalHandler class.

**public class LoanApprovalHandler**

{

public void approvePersonalLoan (PersonalLoanValidator validator)

{

if ( validator.isValid())

{

//Process the loan.

}

}

public void approveVehicleLoan (VehicleLoanValidator validator )

{

if ( validator.isValid())

{

//Process the loan.

}

}

public void approveHouseLoan (HouseLoanValidator validator )

{

if ( validator.isValid())

{

//Process the loan.

} } }

Q1 (iii)

**public class LoanApprovalHandler**

{

public void approveLoan (Validator validator)

{

if ( validator.isValid())

{

//Process the loan.

}

}

}

**public abstract class Validator**

{

public boolean isValid();

}

**public class PersonalLoanValidator extends Validator**

{

public boolean isValid()

{

//Validation Logic

}

**public class VehicleLoanValidator extends Validator**

{

public boolean isValid()

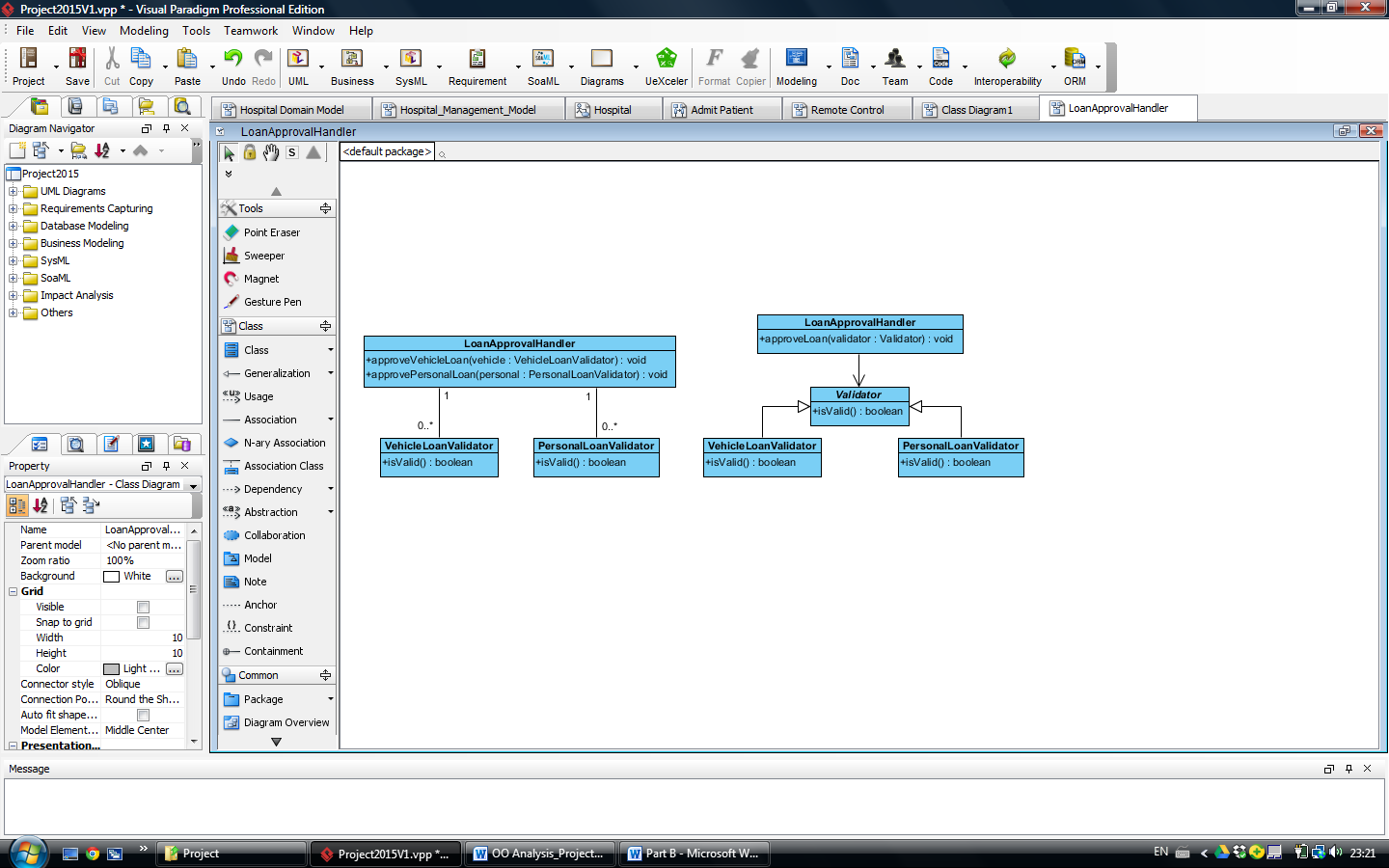
{

//Validation Logic

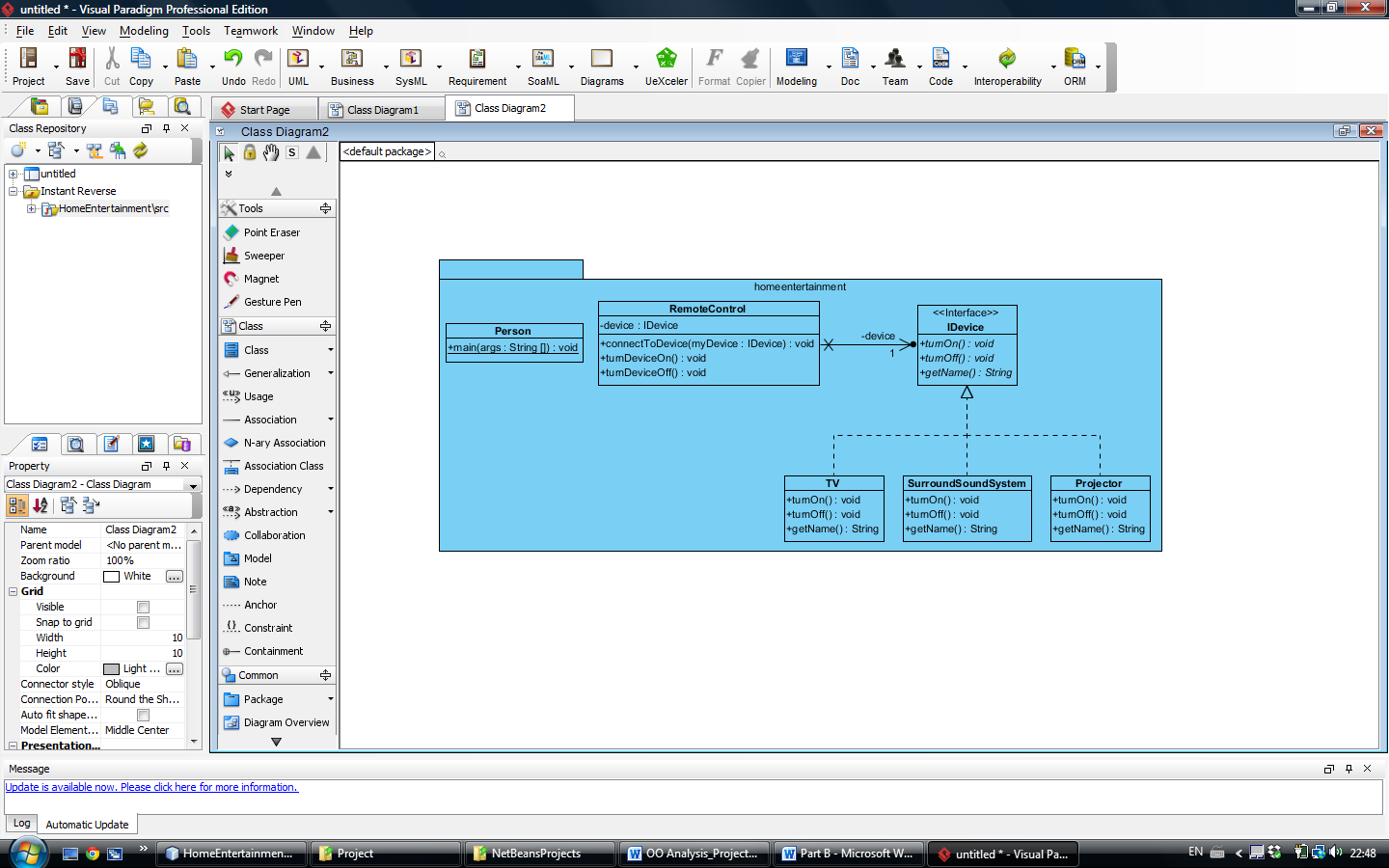
}

}

Q1 (iv)



Q2 Remote Class Diagram



The design pattern I used was Strategy Pattern which tells us to follow the rule “Program to interface, not to completion”. The reason I used this pattern is to remove high coupling between classes and to remove the need for modifying the remote control class if new devices were added and as such it also complies with the OCP (open for extension, closed for modification) principle. If I didn’t use an interface, I would need to have a method for each device within the remote control class. By doing it this way I can create a class for a new device and have it implement the IDevice interface and the remote control class can connect to the new device without any modification as all devices are of type IDevice.

**See below for code for all classes above**

#####################################

public class RemoteControl {

private IDevice device;

public void connectToDevice(IDevice myDevice)

{

device = myDevice;

System.out.println("Remote is connected to: "+device.getName());

}

public void turnDeviceOn()

{

device.turnOn();

}

public void turnDeviceOff()

{

device.turnOff();

}

}

#########################################

public interface IDevice {

public String getName();

public void turnOn();

public void turnOff();

}

#########################################

public class TV implements IDevice {

public String getName()

{

return "TV";

}

public void turnOn()

{

System.out.println("The TV has been turned on!!!!");

}

public void turnOff()

{

System.out.println("The TV has been turned off!!!!");

}

}

#########################################

public class SurroundSoundSystem implements IDevice{

public String getName()

{

return "Surrount Sound System";

}

public void turnOn()

{

System.out.println("The Surround Sound System has been turned on!!!!");

}

public void turnOff()

{

System.out.println("The Surround Sound System has been turned off!!!!");

}

}

#########################################

public class Projector implements IDevice{

public String getName()

{

return "Projector";

}

public void turnOn()

{

System.out.println("The Projector has been turned on!!!!");

}

public void turnOff()

{

System.out.println("The Projector has been turned off!!!!");

}

}

#########################################

public class Person {

public static void main(String[] args) {

RemoteControl rc = new RemoteControl();

IDevice tv = new TV();

IDevice sss = new SurroundSoundSystem();

IDevice prj = new Projector();

Scanner scanner = new Scanner(System.in);

boolean quit = false;

String selection;

while( quit != true)

{

System.out.println("Please choose one of the following:\n"

+"1. Connect to device\n"

+"2. Turn device on\n"

+"3. Turn device off");

selection = scanner.nextLine();

switch (selection)

{

case "1":

System.out.println("Please choose one of the following:\n"

+"1. TV\n"

+"2. Surround Sound System\n"

+"3. Projector ");

selection = scanner.nextLine();

switch (selection)

{

case "1":

System.out.println("|-----------------------------------|");

rc.connectToDevice(tv);

System.out.println("|-----------------------------------|");

break;

case "2":

System.out.println("|-----------------------------------|");

rc.connectToDevice(sss);

System.out.println("|-----------------------------------|");

break;

case "3":

System.out.println("|-----------------------------------|");

rc.connectToDevice(prj);

System.out.println("|-----------------------------------|");

break;

default : System.out.println("Not a valid selection.");

}

break;

case "2":

System.out.println("|-----------------------------------|");

rc.turnDeviceOn();

System.out.println("|-----------------------------------|");

break;

case "3":

System.out.println("|-----------------------------------|");

rc.turnDeviceOff();

System.out.println("|-----------------------------------|");

break;

default : System.out.println("Please eneter a valid selection");

}

}

}

}

Student declaration

I can confirm the following details:

**Student ID/Registration number:** R00050862

**Name:** Thomas O Halloran

**Module Name:** OO Analysis and Design

**Module Lecturer :** Mary Davin

I confirm that this is my own work.

**Due Date:** 17/05/2015

**Student Signature**

**Thomas O Halloran \_**