ENGLISH

University of South-Eastern Norway

EXAMINATION INFORMATION PAGE

Home exam / Portfolio assessment / Report / Semester assignment

Subject code: **IIA1319**	Subject name: Software Engineering	
Responsible subject teacher: Nils-Olav Skeie	Campus: Porsgrunn	Faculty: TNM
Assignment given in WISEflow (date and time): 2-JUN-21 9:00	Submission time in WISEflow (date and time): 2-JUN-21 12:00	
No. of assignments: 15	No. of attachments: 1	No. of pages incl. front page and attachments: 12

Aids and collaboration:

Permitted aids: All aids are allowed.

	Yes	No
Is it an individual exam?	☒	☐
Is it allowed to collaborate with other persons?	☐	☒

Description of individual examination and illegal cooperation will be found at [my.usn.no](my.usn.no)

Criteria for the answers:

Font type: **Calibri or Times New Roman**	Font size: **11 or 12**	Line spacing:
No. of words (min/max):	Maximum no. of pages excl. front page and attachments:	

Source reference:
All external sources, including your own assignments, should be referenced using standard referencing techniques.

Other important information:

- Include/merge all information into one single pdf document,
- No need to include task descriptions nor front page, only the task numbers with your answers.
- The format of this document **must** be pdf,
- The WISEFLOW system will do a plagiarism control of all pdf files so always do referencing when copy information from any source,
- **Any communication with others, electronically or oral, is cheating!**

THE CANDIDATE MUST CHECK THAT THE ASSIGNMENT SET IS COMPLETE

1.1 Description

You are hired by a power supply company to develop a system for estimating snow density and snow depth in various inaccessible places. The company is interested in the Snow Water Equivalent (SWE) parameter which indicates how much water the snow contains. SWE is based on the equation:

$$SWE = h\frac{\rho}{\rho_0}$$

where h is the snow depth, ρ is the density of the snow and ρ_0 is the density of water. Density of water is known, so the snow depth and the average of the snow density is needed for your system. The suggestion is to use small remote snow measuring nodes (RSMN) that measure the capacitance of snow at different depths of the snow pack, and estimates density and snow depth based on this measured information. Your approach is then to use a data-driven model to estimate the relationship between the measured capacitance, the snow depth and the snow density, and the data driven modeling method will be supervised machine learning. In supervised machine learning a large amount of data is needed, including the relationship between the measurements (x) and the parameters that you want to estimate (y) to develop the model ($y = f(x)$). So you need measurement systems and software for storing these parameters, both the measurement values (x) and any manually read values for the parameters that you want to estimate (y). There has already been made some prototypes of RSMN that can be located in areas with snow. See the right part of Figure 1.1 for a picture of one of these RSMN.

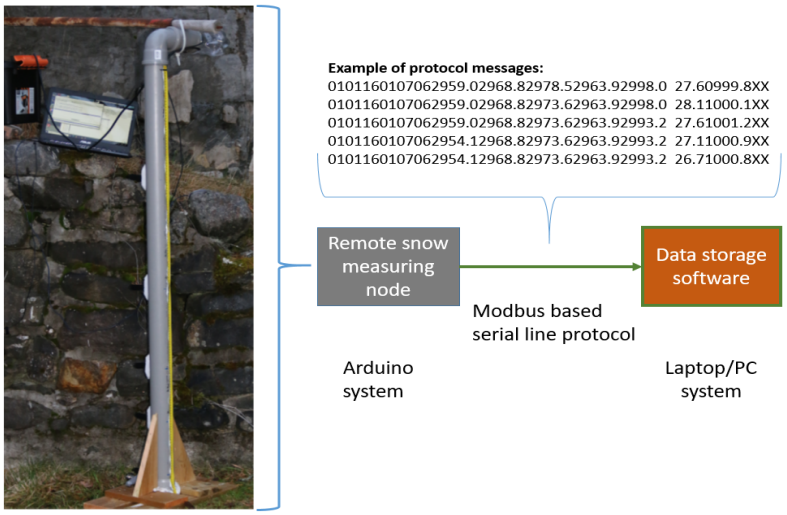

Figure 1.1: To the left, a prototype of the remote snow measuring node, a plastic pipe with seven sensor devices installed together with a centimeter tape glued to the pipe. To the right is a system overview of the remote snow measuring node (RSMN), using an Arduino, examples of the output protocol, and the data storage software (DSS) running on a PC and/or laptop. The focus for this project will be the data storage software (DSS).

The number of capacitance sensors can vary between 1 and 7, and are mounted at fixed heights. In addition, an extra temperature sensor for measuring the ambient temperature and a pressure sensor for measuring the atmospheric pressure, may also be installed. All these sensors are connected to an Arduino system, located at the top of the RSMN, which collects the sensor values and sends these values using a Modbus (Modbus.org 2012) based protocol via the USB port. The format of the protocol is:

$$AVCRNLxxxxxxxxxxxxxxxxxxxxxxxxXX$$

where A is the address of the measuring node $(01-99)$, V is protocol version $(01-09)$, C is the Modbus command $(00-32)$, R is the register start address $(00-64)$ which is Modbus specific (Modbus.org 2012) but not used in this project, N is the number of sensor devices used $(01-16)$ and L is the length of each sensor value $(00-08)$. xxx is the sensor values, according to $N*L$ and the message ends with the letters XX. An example of such a message can be $0101160107062959.02968.82978.52963.92998.00027.60999.8XX$. The address for this node is 01, protocol version is 01, the Modbus command is 16 (ReadMultipleRegisters) (Modbus.org 2012) and the number of sensor devices installed are 07. Modbus command 16

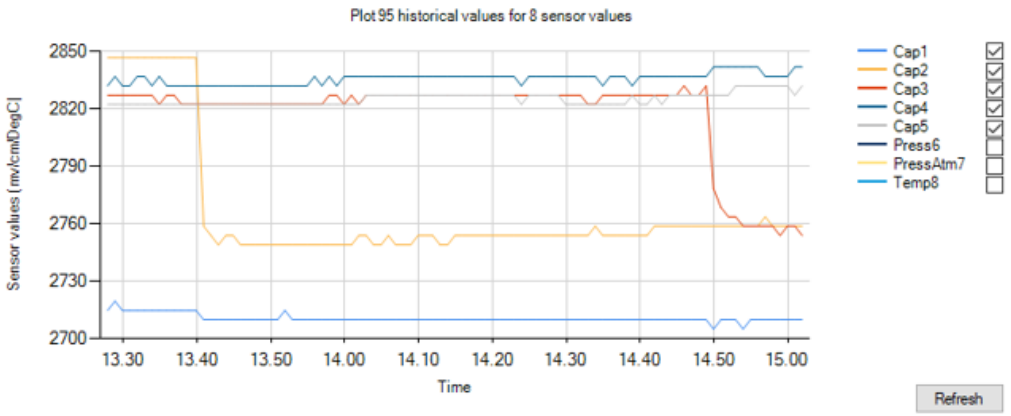

Figure 1.2: An example of information in the plot window, showing the values from the capacitance sensors. Cap1 is the lowest sensor covered by snow, while Cap2 and Cap3 are covered during the plot time.

($0x10$) is the only command supported by the RSMN. More examples of these messages are shown in the upper right part of Figure 1.1. The Arduino software in the RSMN, will run only once every minute, collects a set of samples from the sensor devices and sends these values on the USB port (according to the protocol), to save energy. The USB port is used as a serial port that can be read by any data storage software (DSS) connected with a USB cable. The final version of the RSMN will have a GSM module installed sending the same protocol as an SMS message to the data center of the power supply company.

You need to develop a DSS that can connect to each of these measurement nodes to get the data for developing your models, see Figure 1.1. It is decided to store the data from these RSMN in a Comma Separated Values (CSV) file for further processing of machine learning algorithms. The DSS needs to handle the configuration of the sensor values received from each of the RSMN. The DSS will only log values from one RSMN at a time, so the address (A) of the RSMN should be included in the CSV files together with the sensor values. As your approach is supervised machine learning, you must also be able to input the corresponding snow depth and the snow density in the DSS and store these parameters in the CSV file together with the measured values. The DSS must also include a low pass filtering module for filtering of the measured values, using a moving average filter. Only the filtered values should be stored in the CSV file. The size of the filter must be part of the configuration, the valid range should be between 4 and 64. The snow usually doesn't fall that quickly, so there is no need to save too many equal samples on the CSV file, the time between each sample should also be part of configuration where the valid range should be between 30 minutes and 24 hours.

Since a serial port is used for communication with the Arduino system, the serial port must also be configured with the right name and baud rate ($4800, 9600, 19200$ or 38400). The configuration should be stored in an XML file and should be read by the program at startup. An option can be to have different names of these XML files as the configuration may be different for several of these RSMN.

The DSS should have a user interface with at least one window for configuration parameters, one window for listing the messages received from the serial port, one window showing the actual values from each sensor device, one window for plotting the sensor values for a specific time period, and one status window. An example of information in the plot window is shown in Figure 1.2.

The window showing the actual values (last values) should be the raw values, the filtered values and the last values stored in the CSV file. The software should be developed to run on a laptop for easy connection to the RSMN. It was decided that the program should be developed using the Unified Process (UP) development process, the Unified Modeling Language (UML) for documentation, and the C # programming language for implementing the program.

The following tasks must be documented in the project:

1.2 Tasks

1. (10%) Define the requirements for your software (only the data storage software).
2. (5%) Make a use case diagram for this software.

3. (5%) Make a domain model.
4. (3%) Explain **how** and **why** you will look for any superclasses and subclasses for your software, and indicate any possible superclasses in your software.
5. (3%) Indicate the number of architectural tiers (layers) that you will use in your application and why you will use this architecture.

   The serial port use case is the most important use case as data storage is in focus. If several use cases are handling the serial port, select the most important use case of these use cases.
6. (5%) Explain briefly how you will use the Unified Process (UP) to develop this software, including an indication of the estimated development time.
7. (3%) Make a sketch of the user interface.
8. (3%) Document the format of the CSV file.
9. (7%) Discuss any data structure(s) that you will use to store the sensor data values and the manual input values in your software, and indicate any parameters that will be part of these data structure(s).
10. (14%) Make a fully dressed use case document for the first use case according to the UP.
11. (15%) Make an interaction diagram for the first use case according to the UP, with an indication of the design patterns you have used.
12. (3%) Explain how you will use a version control system in developing your software.
13. (3%) Make a class diagram based on the interaction diagram.
14. (6%) Indicate how you will test your software in this iteration.
15. (15%) A former colleague made a draft for such an application but without any documentation. Do a reverse engineering by making an interaction diagram of this code, but focus on the main structure, and indicate any limitations of this software. The source code is included in the Appendix.

Appendix A

Source code listing

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.IO;
using System.IO.Ports;

namespace DataStorageApp
{
    class Program
    {
        static void Main(string[] args)
        {
            new ProgramStart();
        }
    }
    class ProgramStart
    {
        LayerData rLayerData;
        LayerBusiness rLayerBusiness;
        LayerUser rLayerUser;

        public ProgramStart()
        {
            try
            {
                rLayerData = new LayerData();
                rLayerBusiness = new LayerBusiness(rLayerData);
                rLayerUser = new LayerUser(rLayerBusiness);
                rLayerUser.MainLoop();
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error starting application: " + ex.Message);
            }
        }
    }
    public class LayerData
    {
        SerialPort rSerialPort;
        bool bOpenPort;
```

```
public bool Open(string sPortName, out string sOpenMsg)
{
    if (bOpenPort == true)
    {
        Close();
    }
    try
    {
        rSerialPort = new SerialPort(sPortName, 4800, Parity.None, 8, StopBits.One);
        rSerialPort.ReadTimeout = 500;
        rSerialPort.WriteTimeout = 500;
        rSerialPort.Open();
        sOpenMsg = "Open <" + sPortName + "> serial port OK!";
        bOpenPort = true;
    }
    catch (Exception e)
    {
        sOpenMsg = "Error open <" + sPortName + "> serial port: " + e.Message;
        bOpenPort = false;
    }
    return bOpenPort;
}
public string WaitRead(int iCntMax, int iSleep, int iMaxLoop, bool bTimeoutMsg,
                       out bool bRxTimeout)
{
    int iCnt = 0;
    string sRxMsg = "";
    bRxTimeout = false;
    while (iCnt < iMaxLoop)
    {
        Thread.Sleep(iSleep);
        sRxMsg = sRxMsg + Read(iCntMax, bTimeoutMsg);
        if (sRxMsg.Length >= iCntMax)
        {
            iCnt = iMaxLoop;
            bRxTimeout = false;
        }
        else
        {
            iCnt++;
        }
    }
    return sRxMsg;
}
public string EmptyReadBuffer(int iMaxSize, int iWaitLoops, int iWaitTime)
{
    int iRxMax, iRxLen, iCntLoop, iOffset;
    string sMsgBuf;
    byte[] bMsgBuf;

    iCntLoop = 0;
    iOffset = 0;
    try
    {
        bMsgBuf = new byte[iMaxSize + 16];        // Add some extra bytes to buffer
        sMsgBuf = "";
        while (iCntLoop < iWaitLoops)
        {
```

```
                Thread.Sleep(iWaitTime);
                if ((iRxMax = rSerialPort.BytesToRead) > (iMaxSize - iOffset))
                {
                    iRxMax = (iMaxSize - iOffset);
                }
                if (iRxMax > 0)
                {
                    iRxLen = rSerialPort.Read(bMsgBuf, iOffset, iRxMax);
                    iOffset += iRxLen;
                }
                if (iOffset >= iMaxSize)
                {
                    iCntLoop = iWaitLoops;
                }
                else
                {
                    iCntLoop++;
                }
            }
            sMsgBuf = Encoding.Default.GetString(bMsgBuf);
        }
        catch (Exception e)
        {
            sMsgBuf = "<ErrorSerialPort=" + e.Message + ">";
        }
        return sMsgBuf;
    }
    public string Read(int iCntMax, bool bTimeoutMsg)
    {
        int iLen, iMsgCnt, iOffset, iMaxLoop;
        string sMsgBuf;
        byte[] bMsgBuf;

        iLen = 0;
        sMsgBuf = "";
        try
        {
            bMsgBuf = new byte[iCntMax + 16];
            iOffset = 0;
            try
            {
                iMaxLoop = 0;
                while (iOffset < iCntMax && iMaxLoop < 64)
                {
                    iLen = rSerialPort.Read(bMsgBuf, iOffset, (bMsgBuf.Length - iOffset));
                    iOffset += iLen;
                    iMaxLoop++;
                }
                for (iMsgCnt = 0; iMsgCnt < iOffset; iMsgCnt++)
                {
                    sMsgBuf = sMsgBuf + Convert.ToChar(bMsgBuf[iMsgCnt]);
                }
            }
            catch (TimeoutException)
            {
                if (iOffset > 0)
                {
                    for (iMsgCnt = 0; iMsgCnt < iOffset; iMsgCnt++)
```

```
                {
                    sMsgBuf = sMsgBuf + Convert.ToChar(bMsgBuf[iMsgCnt]);
                }
                if (bTimeoutMsg == true)
                {
                    sMsgBuf = sMsgBuf + "<Timeout>";
                }
            }
        }
        catch (Exception e)
        {
            sMsgBuf = sMsgBuf + "<Exception=" + e.Message + ">";
        }
    }
    catch (Exception e)
    {
        sMsgBuf = sMsgBuf + "<Serial port error=" + e.Message + ">";
    }
    return sMsgBuf;
}
public void Close()
{
    if (bOpenPort == true)
    {
        rSerialPort.Close();
        bOpenPort = false;
    }
}
public bool ChkOpen
{
    get
    {
        return bOpenPort;
    }
}
public void UpdateLogCsvFile(string sFileName, double[] dVals, out string sErrMsg)
{
    string sMsg;
    int iCntr;
    TextWriter rLogFile;

    if (dVals.Length > 0)
    {
        sMsg = DateTime.Now.ToString("dd-MMM-yyy;HH:mm:ss");
        for (iCntr = 0; iCntr < dVals.Length; iCntr++)
        {
            sMsg = sMsg + "; " + dVals[iCntr].ToString("F1");
        }
        try
        {
            rLogFile = new StreamWriter(sFileName, true);
            rLogFile.WriteLine(sMsg);
            rLogFile.Close();
            sErrMsg = "";
        }
        catch (Exception ex)
        {
            sErrMsg = "Err: Writing log file <" + sFileName + ">: " +
```

```
                                ex.Message;
            }
        }
        else
        {
            sErrMsg = "Err: No writing to log file <" + sFileName +
                                ">: Empty value buffer";
        }
    }
    public void HeaderLogCvsFile(string sFilename, string[] sHeader, out string sErrMsg)
    {
        string sMsg;
        int iCntr;
        TextWriter rLogFile;

        if (File.Exists(sFilename) == false)
        {
            sMsg = "Date; Time";
            for (iCntr = 0; iCntr < sHeader.Length; iCntr++)
            {
                sMsg = sMsg + "; " + sHeader[iCntr];
            }
            sMsg = sMsg + "; V01";
            try
            {
                rLogFile = new StreamWriter(sFilename, false);
                rLogFile.WriteLine(sMsg);
                rLogFile.Close();
                sErrMsg = "";
            }
            catch (Exception ex)
            {
                sErrMsg = "Err: Writing header log file <" + sFilename + ">: " +
                           ex.Message;
            }
        }
        else
        {
            sErrMsg = "Err: Header log file <" + sFilename + "> already exists!";
        }
    }
}
public class LayerBusiness
{
    LayerData rLayerData;
    public LayerBusiness(LayerData pld)
    {
        rLayerData = pld;
    }
    public string GetSerialPortData(int iMaxSize)
    {
        return rLayerData.EmptyReadBuffer(iMaxSize, 16, 500);
    }
    public string OpenComPort(string sComPort)
    {
        string sBuf;

        rLayerData.Open(sComPort, out sBuf);
```

```
            return sBuf;
        }
        public void CloseComPort()
        {
            rLayerData.Close();
        }
        public bool ChkOpenComPort
        {
            get
            {
                return rLayerData.ChkOpen;
            }
        }
    }
    public class LayerUser
    {
        LayerBusiness rLayerBusiness;
        public LayerUser(LayerBusiness plb)
        {
            rLayerBusiness = plb;
        }
        public void MainLoop()
        {
            ConsoleKey cKey = ConsoleKey.S;
            ConsoleKeyInfo cKeyInfo;

            while (cKey != ConsoleKey.Q)
            {
                Console.WriteLine(" R: display of raw COM port data");
                Console.WriteLine(" Q: Quit application");
                cKeyInfo = Console.ReadKey();
                cKey = cKeyInfo.Key;
                cKey = UserSelection(cKey);
            }
            rLayerBusiness.CloseComPort();
        }
        private ConsoleKey UserSelection(ConsoleKey cKey)
        {
            string sBuf;
            int iCnt = 0;

            switch (cKey)
            {
                case ConsoleKey.R:
                    Console.WriteLine(" Display raw data ..");
                    rLayerBusiness.OpenComPort("COM1");
                    while (cKey != ConsoleKey.Q)
                    {
                        sBuf = rLayerBusiness.GetSerialPortData(128);
                        if (sBuf.Length > 0)
                        {
                            Console.WriteLine("Rx[" + iCnt.ToString() + "]=<" + sBuf + "> ");
                            iCnt++;
                        }
                        if (Console.KeyAvailable == true)
                        {
                            cKey = Console.ReadKey().Key;
                        }
```

```
                    }
                    rLayerBusiness.CloseComPort();
                    break;
                default:
                    break;
            }
            return cKey;
        }
    }
}
```

Bibliography

Modbus.org (2012), Modbus application protocol specification, Technical report, Modbus.org. Version 1.1b3.