University of South-Eastern Norway

EXAMINATION INFORMATION PAGE
Written examination

Subject code: **IIA1319**	Subject name: Software Engineering	
Examination date: 5-JUN-20	Examination time from/to: 09:00 to 12:00	Total hours: 3
Responsible subject teacher: Nils-Olav Skeie		
Campus: Porsgrunn	Faculty: TNM	
No. of assignments: 14	No. of attachments: 1	No. of pages incl. front page and attachments: 10

Permitted aids:

- All learning aids, including Internet, can be used.
- **Any communication with others, electronically or oral, is cheating!**

Information regarding attachments:

Comments:

- Include/merge all information into one single pdf document (report),
- No need to include exam task descriptions nor a 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!**

Select the type of examination paper

Spreadsheets ☐ Line sheets ☐

1.1 Description

You are working as a software designer in a company with several process control systems (PCS). A challenge for the PCS operators are handling alarm states and especially shelving alarms. During this Covid-19 situation you decided to make an alarm system training simulator that these operators can use for training purpose when staying at the home office. The training simulator will contain logic to simulate the filling fraction and the temperature in a set of tanks. However, the main purpose of the training simulator is to include an alarm system with shelving functionality. The shelving functionality will let the operator move information for an alarm from a main alarm list (display) to a shelve alarm list (display) to reduce the number of important alarms in the main alarm list.

So the main functionality of the training simulator will be to train the operators in handling alarms, managing the alarm states and an option of changing some of the alarm limits. The temperature alarms consists of two limits, low temperature limit (LOT) and high temperature limit (HIT), for each tank. The alarm limits can be changed by the operator, but will always start with the configured limits. The default configured limits are 10 °C for LOT and 30 °C for HIT. The sensors, temperature and level, may have an general I/O error (I/O-ERR). The configuration for the training simulator will be stored in a XML file to simplify the installation of the training simulator on different type of computers. The configuration data will be used when starting the training simulator, and can be changed and saved by the operator.

The training simulator will contain sensor models for simulating the filling fraction and temperature values for the tanks with logic for realistic input signals internal in the training simulator. This logic will control the load and discharge of the tanks, and adjustment of the temperatures. The sensor models and tanks will be updated based on a sampling time that can be configured. The default sampling time will be 10 seconds. The number of tanks can also be configured, and the default number will be 20 tanks. The process overview for the training simulator will contain a list of tanks with temperature value, filling fraction and alarm limits together with a main alarm list.

Many alarm systems used are based on the Norwegian alarm standard YA-710/YA-711, while some newer alarm systems are based on the new international alarm standard IEC 62682. The main differences between the YA-710/YA-711 and the IEC 62682 are more real-time requirements in IEC 62682 regarding the maximum number of alarms that an operator can deal with during specific time periods. You have decided to go for the YA-710/YA-711 standard. The YA-710/YA-711 alarm standard is shown in Figure 1.1 and you have decided not to include the alarm filtering nor the alarm suppression functionality in this training simulator.

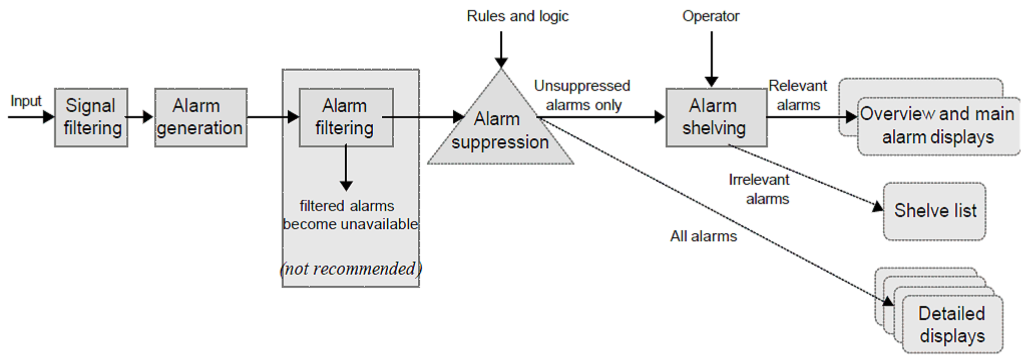

Figure 1.1: The main functions of an alarm system ((YA-711 2001)).

The passive, active, acknowledge and shelving alarm states should be implemented in this training simulator according to Figure 1.2. The shelve alarm list must be an optional alarm list that can be selected by the operator. Any alarms that are in the acknowledge state can also be changed by the operator to the shelve state. The operator can change any alarms in the shelve state back to the acknowledge state. The system should change the shelve state back to acknowledge state after a time-out period, defined as part of the configuration. The default value will be one hour for this training simulator. The system will also change the alarm state from either acknowledge or shelving to passive when the alarm condition is no longer valid. So the alarm state changes that the system can influence is passive to active, acknowledge to passive, shelving to acknowledge at timeout, and shelving to passive when the process value is outside the alarm limit range. The alarm state that the user can change is active to acknowledge, acknowledge

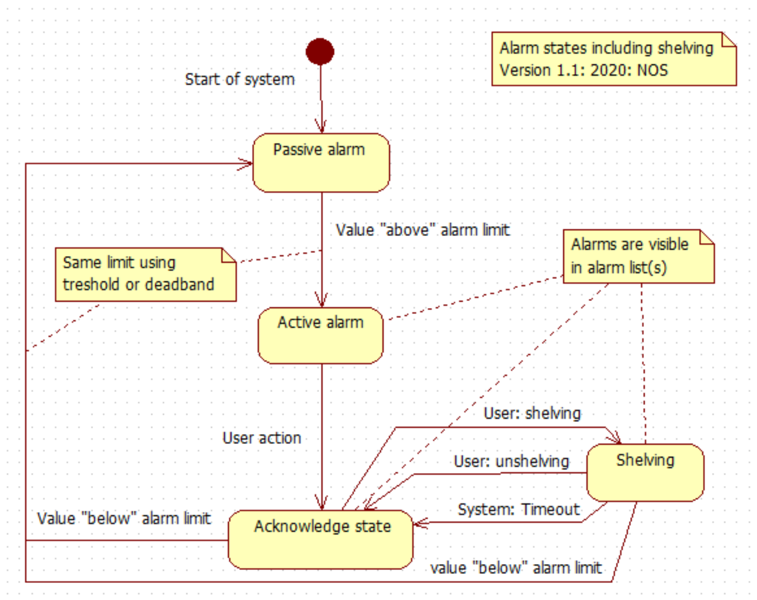

Figure 1.2: The alarm states for the training simulator.

to shelving, and shelving to acknowledge (unshelving). Note that a passive alarm state is not visible in the system and can be removed.

This project will focus on analysis and design of the software, preferably using the C# programming language. It was decided that the Unified Process (UP) should be used as the software development process and any diagram should be made using Unified Modeling Language (UML). The following tasks must be documented in the project:

1.2 Tasks

1. (10%) Define the requirements for your software (the whole training simulator),
2. (5%) Make a use case diagram for this software,
3. (7%) Make a domain model,
4. (2%) Discuss the number of architectural layers (tiers) that you will use in your application,

   The use case handling the system alarm states is the most important use case as the alarm system is in focus. If several use cases are handling system alarm states, select the most important use case of these use cases.
5. (5%) Explain briefly how you will use the Unified Process (UP) to develop this software, including an indication of the development time.
6. (5%) Make a sketch of the user interface for handling the alarms according to your requirements.
7. (15%) Make a fully dressed use case document for the first use case according to the UP,
8. (5%) Discuss the data structure that you will use to store the alarm data in your software, and indicate any parameters that will be part of this data structure.
9. (5%) Explain **how** and **why** you will look for any superclasses and subclasses for your software, and indicate any possible superclasses in your software,
10. (18%) Make an interaction diagram for the first use case according to the UP, with an indicating of the design patterns you have used.
11. (4%) Make a class diagram based on the interaction diagram,

12. (1%) Explain how many interaction diagrams and class diagrams you will have when your software is finished.

13. (3%) Indicate how you will test your software in this iteration.

14. (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.Threading;
//
namespace AlarmShelving
{
    class LayerUser
    {
        LayerBusiness rLayerBusiness;
        public LayerUser(LayerBusiness plb)
        {
            rLayerBusiness = plb;
        }
        public void InitSimulator()
        {
            rLayerBusiness.LoadConfigurationData();
            rLayerBusiness.InitTrainingSimulator();
        }
        public void StartSimulator()
        {
            rLayerBusiness.StartSimulator();
            Console.WriteLine(" Alarm training simualtor");
            Console.WriteLine(" P - Process values and main alarm list");
            Console.WriteLine(" L - main and shelving alarm lists");
            Console.WriteLine(" A - Acknowledge current alarm");
            Console.WriteLine(" S - Shelve current alarm");
            Console.WriteLine(" U - Unshelve current alarm");
            Console.WriteLine(" Q - Quit the simulator");
            UserSelection();
            rLayerBusiness.StopSimulator();
        }
        public void UserSelection()
        {
            bool bWait = true;

            while (bWait == true)
            {
                switch (Console.Read())
                {
                    case 'Q':
                    case 'q':
                        bWait = false;
                        break;
                    default:
                        break;
                }
            }
        }
    }
    class LayerBusiness
    {
        const int MAX_TANK_ID = 16;
        const int MAX_ALARM_ID = MAX_TANK_ID;
        LayerData rLayerData;
        ConfigData rConfigData;
        SimulateProcessvalues rSimulateProcessvalues;
        ManageAlarmStates rManageAlarmStates;
        public LayerBusiness(LayerData pld)
        {
            rLayerData = pld;
        }
        public void LoadConfigurationData()
        {
            rConfigData = new ConfigData(rLayerData);
        }
        public void InitTrainingSimulator()
        {
            rManageAlarmStates = new ManageAlarmStates(MAX_ALARM_ID);
            rSimulateProcessvalues = new SimulateProcessvalues(MAX_TANK_ID,
                rConfigData.GetProcessSamplingTime(), rManageAlarmStates);
        }
        public void StartSimulator()
        {
            rSimulateProcessvalues.StartSimulator();
        }
        public void StopSimulator()
        {
            rSimulateProcessvalues.StopSimulator();
        }
    }
    class LayerData
    {
        public LayerData()
        {

        }
        public bool LoadConfigData()
        {
            return false;
        }
        public bool SaveConfigData()
        {
            return false;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            ProgramStartup rProgramStartup;
            rProgramStartup = new ProgramStartup();
        }
    }
    class ProgramStartup
    {
        public ProgramStartup()
        {
            LayerData rLayerData;
            LayerBusiness rLayerBusiness;
            LayerUser rLayerUser;

            rLayerData = new LayerData();
            rLayerBusiness = new LayerBusiness(rLayerData);
            rLayerUser = new LayerUser(rLayerBusiness);
            rLayerUser.InitSimulator();
            rLayerUser.StartSimulator();
        }
    }
    class ConfigData
    {
        private int AlarmTimeOut;
        private int ProcessSamplingTime;
        public ConfigData(LayerData pld)
        {
            if (pld.LoadConfigData() == false)
            {
                UseDefaultSetting();
            }
        }
        private void UseDefaultSetting()
        {
            AlarmTimeOut = 120;
            ProcessSamplingTime = 10000;
        }
        public int GetProcessSamplingTime()
        {
            return ProcessSamplingTime;
        }
    }
    class SimulateProcessvalues
    {
        TagTankValue[] rTagTankValue;
        Thread rTagTankThread;
        int iTagId = 0;
        int iSamplingTime;
        bool bRunFlag;
        ManageAlarmStates rManageAlarmStates;
        public SimulateProcessvalues(int iMaxTId, int iSampTime, ManageAlarmStates pmas)
        {
            int iCnt;

            rManageAlarmStates = pmas;
            rTagTankValue = new TagTankValue[iMaxTId];
            for (iCnt = 0; iCnt < rTagTankValue.Length; iCnt++)
            {
                rTagTankValue[iCnt] = new TagTankValue((double)iCnt, (double)iCnt);
            }
            iTagId = 0;
            iSamplingTime = iSampTime / rTagTankValue.Length;
            bRunFlag = true;
            rTagTankThread = new Thread(new ThreadStart(this.Run));
        }
        public void StartSimulator()
        {
            bRunFlag = true;
            rTagTankThread.Start();
        }
        public void StopSimulator()
        {
            bRunFlag = false;
        }
        public void Run()
        {
            while (bRunFlag == true)
            {
                if (iTagId < rTagTankValue.Length)
                {
                    rManageAlarmStates.CheckTagAlarmState(iTagId);
                    iTagId++;
                    Console.Write(" TagId=" + iTagId.ToString());
                }
                else
                {
                    iTagId = 0;
                }
                Thread.Sleep(iSamplingTime);
            }
        }
    }
    class TagTankValue
    {
        double dLevel;
        double dTemp;
        public TagTankValue(double dLVal, double dTVal)
        {
            dLevel = dLVal;
            dTemp = dTVal;
        }
    }
    class ManageAlarmStates
    {
        TagAlarmInfo[] rTagAlarmInfo;

        public ManageAlarmStates(int iMaxTId)
        {
            int iCnt;

            rTagAlarmInfo = new TagAlarmInfo[iMaxTId];
            for (iCnt = 0; iCnt < rTagAlarmInfo.Length; iCnt++)
            {
                rTagAlarmInfo[iCnt] = new TagAlarmInfo();
            }
        }
        public void CheckTagAlarmState(int iId)
        {
        }
    }
    class TagAlarmInfo
    {
        double dAlarmLimit;
        int iAlarmState;
        int iTagTankId;
        int iTagAlarmType;
        public TagAlarmInfo()
        {
            dAlarmLimit = 0.0F;
            iAlarmState = 0;
            iTagTankId = -1;
            iTagAlarmType = -1;
        }
    }
}
////////////////////////////////////////////////////////////////
```

Bibliography

YA-711 (2001), Principles for alarm system design (ya-711), Technical report, Norwegian Petroleum Directorate.