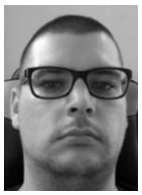
Single – User System

Project Report - Eurofins Steins Laboratory A/S

Lukas Vaisnoras - 280107



Marcel Notenboom – 279910

Deivydas Zibkus – 280133

Gais El-AAsi - 279910

Supervisors:

Mona Wendel Andersen; Allan Rune Henriksen

Num. of characters incl. spaces: 26328

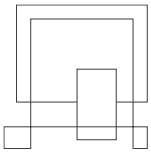


Table of content

ABSTRACT ..... **IV**

1. INTRODUCTION **1**

2. REQUIREMENTS **3**

2.1. FUNCTIONAL REQUIREMENTS 3

2.2. NON-FUNCTIONAL REQUIREMENTS 4

3. ANALYSIS **5**

3.1. USE CASE DIAGRAM 5

3.1.1. Use Case Description *7*

3.1.2. Activity Diagram *10*

4. DESIGN **11**

4.1. WEBSITE DEVELOPMENT 15

5. IMPLEMENTATION **15**

6. TEST **20**

7. RESULTS AND DISCUSSION **21**

8. CONCLUSIONS **23**

9. PROJECT FUTURE **24**

SOURCE OF INFORMATION **25**

APPENDICES **26**

APPENDIX A – USE CASE DIAGRAM; 26

APPENDIX B – USE CASE DESCRIPTION; 26

APPENDIX C – ACTIVITY DIAGRAM; 26

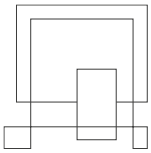
APPENDIX D – CLASS DIAGRAM; 26

APPENDIX F – WEBSITE; 26

APPENDIX E – SEQUENCE DIAGRAM; 26

APPENDIX G – SOURCE CODE; 26

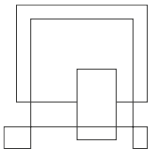
APPENDIX H – JAR FILE; 26



APPENDIX I – JAVA DOCUMENTATION; 26
APPENDIX J – USER GUIDE; 26
APPENDIX K – PROJECT DESCRIPTION 26
APPENDIX L – PROJECT GANTT CHART 26

TABLE OF FIGURES

FIGURE 1. USE CASE DIAGRAM 5
FIGURE 2. CORRELATION BETWEEN USE CASES AND REQUIREMENTS 7
FIGURE 3. USE CASE DESCRIPTION MANAGE SHEETS 9
FIGURE 4. MANAGE EMPLOYEE ACTIVITY DIAGRAM 10
FIGURE 5 SYSTEMADAPTER AND FILEIO CLASS 11
FIGURE 6 WEEK, EMPLOYEE AND ANALYSIS LIST 12
FIGURE 7 TASK AND TASKLIST CLASSES 13
FIGURE 8 EDIT EMPLOYEE 13
FIGURE 9 SEARCH FOR THE EMPLOYEE 14
FIGURE 10 ADD NEW EMPLOYEE 14
FIGURE 11 ANALYSIS CONSTRUCTORS 16
FIGURE 12 EMPLOYEE CONSTRUCTOR 16
FIGURE 13 ADDING AN EMPLOYEE 17
FIGURE 14 LOGIN PROCESS 17
FIGURE 15 SHOWEMPLOYEE CLASS 18
FIGURE 16 ADDING AN EMPLOYEE FROM THE EMPLOYEEPANEL() 19



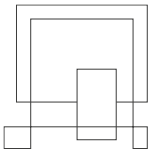
Abstract

The project attempts to provide a work-planning tool for the Chemistry Department of Eurofins Steins Laboratory A/S. The aim is to provide a software system that will ease and automatize the managerial work subsequently freeing up time and effort for the department.

In order to achieve that, the project must be able to provide capabilities to view multiple sheets at a time; modify, store and update data; offer a graphical user interface and basic level of security.

In the development process, different analytical and designing tools were used, everything being done in a Waterfall model and the implementation of the system was done using JAVA programming language.

In the end, the result of this project was summarized in form of a software system, that does provide the basic needs for such a work-planning tool, but not sufficient to make it valuable in the current state. The needed improvements for the software to become what it was intended to are suggested as well as discussions for the current shortcomings.



1. Introduction

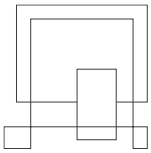
Eurofins Steins Laboratory A/S is Danish subsidiary of a network of laboratories across the globe with their main activity being chemical and microbiological testing of food and related products. The Danish subsidiary has a total number of around 325 employees, chemistry department counting around 60 employees, 50 of them being technicians in production. Every-day workload is high (approx. 30 different analysis each day) meaning that there is a need for careful planning and management of both: employees and activities. The current planning tool of the chemistry department is a spreadsheet (Microsoft Excel). (Viuff, 2018)

The management of the chemistry department is running into different problems in planning the work for its technicians because their Microsoft Excel is not offering enough capabilities for an efficient planning. Moreover, because there is a need of close knowledge about the technicians and activities to be able to perform the planning, the job is very difficult to be passed to other members of the management and requires a significant amount of manual work. Every piece of information is added manually, and to do the planning work, the managers need to juggle and synchronize the sheets described above, which makes their work very difficult and prone to errors. (Viuff, 2018) (Eurofins Steins Laboratory A/S, 2018)

The relevance of this project is given by the desire of the management of the Chemistry Department to have a better tool for their scheduling activity, with a software system that will ease and automatize the planning work subsequently freeing up time and effort for the department. (Eurofins Steins Laboratory A/S, 2018)

In developing the project there are some limitations in regards to what can be used and/or done as described below:

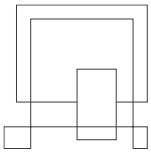
- The solution will not have a client/server system;
- The solution will not be a mobile application;
- The solution will not be a web based application;



The main programing language will be **JAVA**; therefore, it will be **Single – User System**, in form of a computer application. (Henriksen & Andersen, 2018)

The development of the project will go through a few phases, each of them playing a crucial role in the development of the project and will be closely described in the following chapters:

- Definition of the requirements;
- Analysis phase;
- Design phase;
- Implementation phase;
- Testing and post test phase.

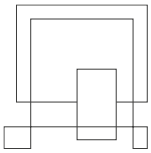


2. Requirements

The requirements were made taking in consideration the described problem by the company and imposed delimitations. SMART principle was used to ensure that the requirements are as precise and testable as possible. (Schedlbauer, 2011) (Eurofins Steins Laboratory A/S, 2018) (Viuff, 2018)

2.1. Functional Requirements

1. The system must be able display all 4 sheets (Schedule / Staff Time / Employee Preferences / Training) in one place;
2. The team leader must be able to register/hide/edit/delete data about an employee;
3. The team leader must be able to register/ edit/delete data about an analysis;
4. When registering an employee the team leader must state the name (first + last name), email and a generated ID (initials + number);
5. The employee parameter must have other fields as:
    - State of employee's training for different types of analysis (not trained (default) / retraining / trained / undertraining);
    - Vacation period (none by default / pending / approved);
    - Preferences (none by default);
    - Other comments (none by default);
6. The name and type must be stated when registering an analysis;
7. The system must update stored information about the employees and analysis when they are modified;
8. The system must check and prevent the user for repetitions (i.e. employee with the same name and ID);
9. The team leader must be able to view / edit the sheets individually;
10. When editing schedule sheet the team leader must be able to edit:
    - Date / week number fields;
    - Week type (S/L)
    - Comments;

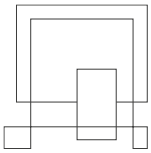


- Type of analysis for each day;
- Employees field;

11. When editing preferences sheet the team leader must be able to edit:
    - Employees preferences;
12. When editing the training sheet the team leader must be able to edit:
    - State of employee's training for different analyses;
13. When editing the staff time sheet the team leader must be able to edit:
    - Analyses;
    - Number of employees needed for each analysis;
    - Type of week;
14. When viewing the 4 sheets the team leader must be able to view past / current / future weeks;
15. The system must be able to provide search capabilities for the employees / analyses fields;
16. Team Leader can cancel any process at any time;
17. The system must offer settings capabilities to set up:
    - What is displayed;
    - Setting / changing colors for different fields;
    - Setting / changing dates for the sheets;
    - Different graphic related settings (fonts, text color, etc.);

2.2. Non-Functional Requirements

1. The main programing language of the system must be JAVA;
2. The system must be a single user software;
3. The system must offer graphic user interface (not command-line);
4. The system must store data for at least one year;



A few things to be pointed out here are:

- One of the most important requirement (ranked 1) is about being able to display all four sheets at the same time. It was ranked first, as during the presentation company's employee has underlined many time that being able to view four sheets at a time it an important feature;
- The last ranked requirement refers to Settings Capabilities for the system, it was agreed to be ranked as least important due to the fact that it does not provide any actual functionality to the system;

3. Analysis

In the analysis part, a Diagram of the Use Cases was developed to underline the main capabilities of the system, together with a table that elaborates on the correlation between Use Case Diagram and the described above requirements the Use Case Diagram can also be seen in Appendix A.

3.1. Use Case Diagram

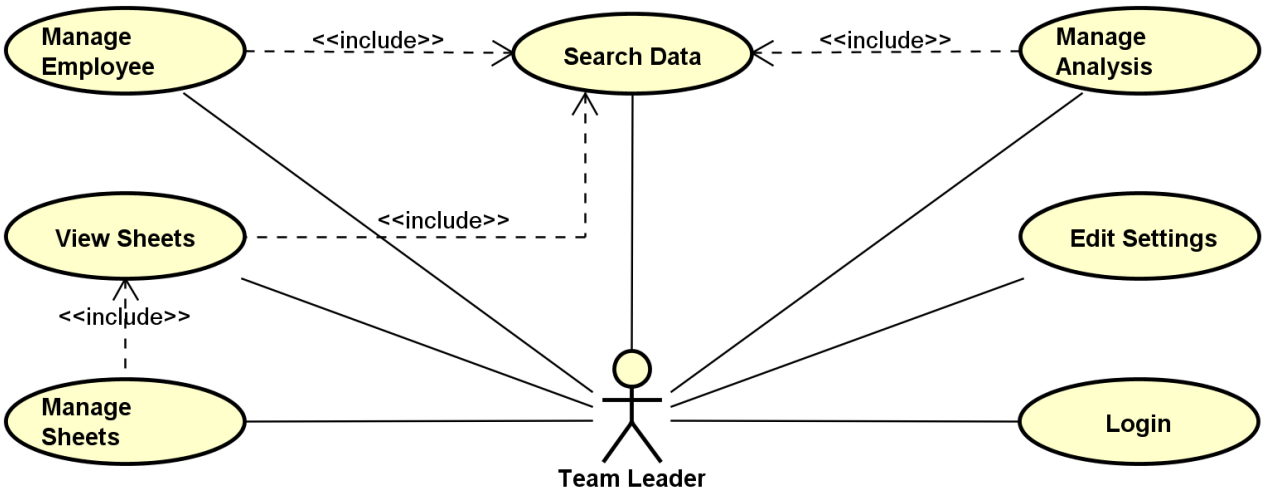
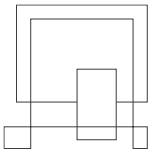


Figure 1. Use Case Diagram

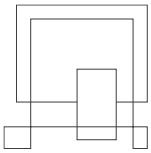
A few things that are significant about the Use Case diagram are:



- The view sheet Use Case was developed as a response to the main requirement of "being able to view all 4 sheets in one window". The View Sheet should provide this feature of choosing between 1 and 4 sheets to be viewed at the same time, also an important thing is that the *Manage Sheets* use case uses *View Sheets* to offer a handy way of editing data in the sheets;
- All use cases that provide a way to manage data (i.e. *Manage Employee, Manage Analysis* and *Manage Sheets*) include the *Search Data* to again ease the work of finding the parameters/data that needs to be edited;

The correlation between requirements and Use Cases is shown in the below table. Some Use Cases solve different requirements because there are different requirements that cover to some extent different issues about a topic. For example, the Manage Employee is addressing the requirements 2, 4, 5, 7, 8, 11, 12, 16.

There are also requirements that are solved by more than one Use Case, for example requirements 7, 8, 16. This happens when a requirement is very broad (i.e. requirements 16 *Team Leader can cancel any process at any time.*).



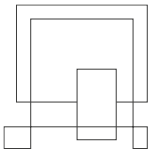
REQUIREMENT	USE CASE
1	View Sheets
2	Manage Employee
3	Manage Analysis
4	Manage Employee
5	Manage Employee
6	Manage Analysis
7	Manage Employee/Manage Analysis/Manage Schedule
8	Manage Employee/Manage Analysis/Manage Schedule
9	View Sheets
10	Manage Schedule
11	Manage Employee
12	Manage Employee
13	Manage Analysis
14	View Sheets
15	Search Data
16	Manage Employee/Manage Analysis/Manage Schedule/Log-in/Search Data/Change Settings
17	Change Settings

Figure 2. Correlation between Use Cases and Requirements

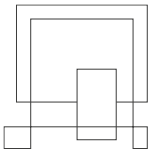
The fact that some Use Cases address more than one requirement and vice versa means that the system is interrelated and works as a whole rather than being an array of 1 to 1 requirement – use case, because of that there will be a higher complexity when designing the system

3.1.1. Use Case Description

As a more in-depth analysis of the Use Cases, for each Use Case were developed Use Case Description to elaborate in a step by step manner, the *Manage Sheet Use Case Description* can be seen below rest of them being included in the Appendix B.



ITEM	VALUE
USE CASE	Manage Sheets
SUMMARY	Team Leader Modifies data in the sheet/sheets;
ACTOR	Team Leader
PRECONDITION	Team Leader needs to be logged in; The Use Case uses Search Data Use Case; Must be in a sheet to modify data;
POST CONDITION	Data in the desired sheet/sheets is modified;
BASE SEQUENCE	1. Choose data to be edited; 2. Team Leader edits desired data; 3. System checks for errors; 4. Data is stored;
BRANCH SEQUENCE	None
EXCEPTION SEQUENCE	Employees are double booked: 1. - 3. same as Base Sequence; 4. Displays message about double booking; Go back to step 2; Employee on vacation is put on a analysis: 1. - 3. same as Base Sequence; 4. Displays message about employee being on vacation; Go back to step 2; Not enough personal on an analysis: 1. - 3. same as Base Sequence; 4. Displays message about not enough personal on an analysis; Go back to step 2; More employees added to an analysis than required: 1. - 3. same as Base Sequence;



	4. Displays message about too many employees; Go back to step 2; Missing non-optional data for the employee: 1. - 3. same as Base Sequence; 4. Displays message about missing non-optional; Go back to step 2; Team Leader tries to add/edit an analysis causing two identical analysis in the system: 1. - 3. same as Base Sequence; 4. Displays message identical analysis; Go back to step 2;
SUB USE CAS	View Sheets;
NOTE	The process can be cancelled at any time;

Figure 3. Use Case Description Manage Sheets

There are a few points that need to be addressed when analyzing the Use Case Description:

- First of all, the team leader must be logged in to access the Use Case, meaning that the system provides a certain level of security;
- Another thing to be pointed out is that the Use Case can use the *Search Data* option but the system must have an opened sheet to be able to modify it;
- It can be seen that from the Base Sequence point of view the Use Case is rather simple, but there are many things that can go wrong, because of that the Exception Sequence is much more developed as it tries to cover every possible way of deviation;
- And of course, because it the system must have an opened sheet to modify it this Use Case uses the *View Sheets* use case;
- Another thing that was decided to be done is to have an employee specify the name and email so it will make sure that there is not another employee with the same data;

3.1.2. Activity Diagram

As well as Use Case Description for each Use Case were developed Activity Diagram which provides a better visual understanding of the Use Case, below is presented an Activity Diagram for the *Manage Employee Use Case*, and the rest of them can be viewed in the Appendix C.

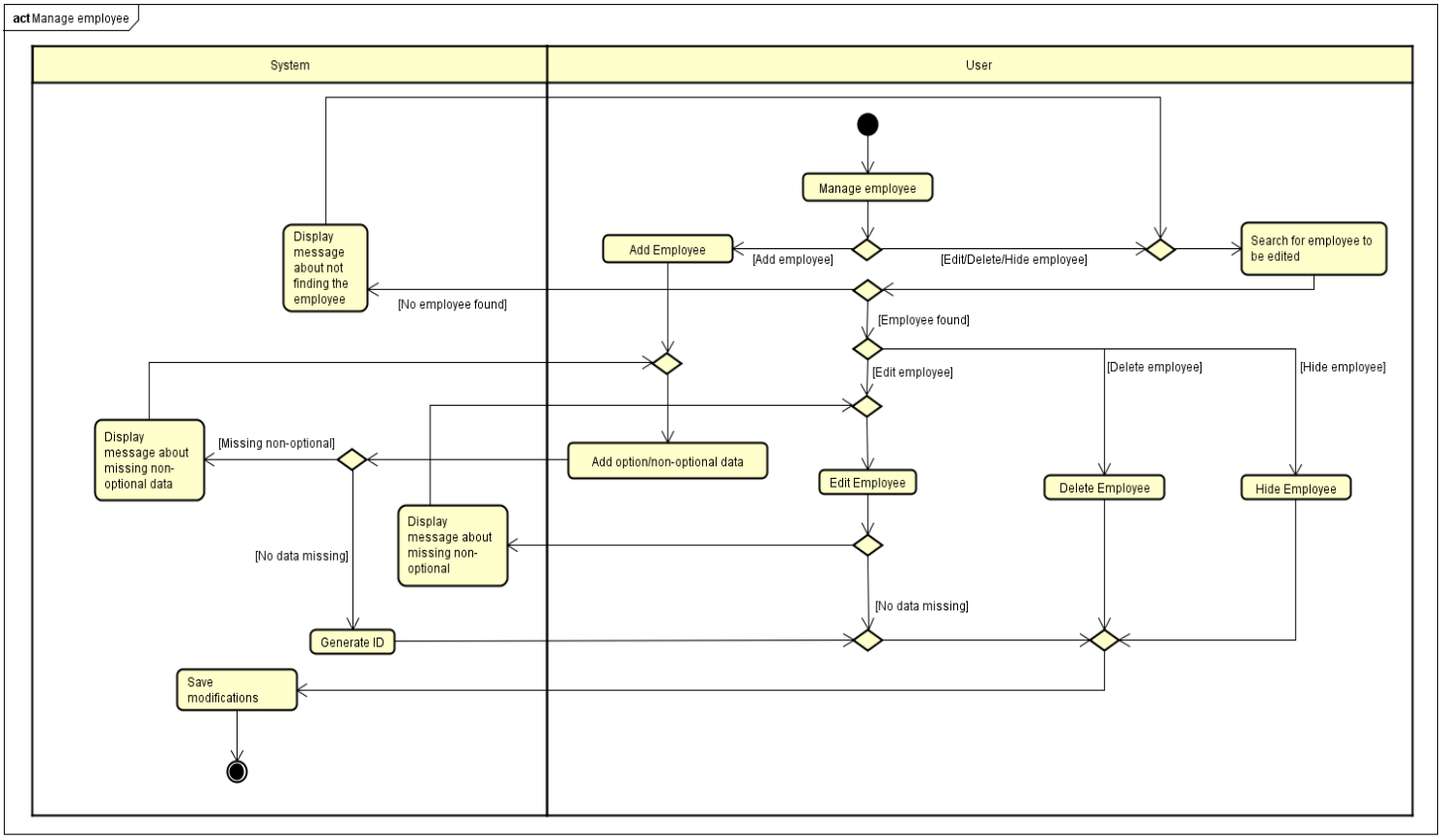
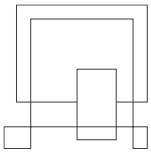


Figure 4. Manage Employee Activity Diagram

Manage Employee activity diagram provides a good overview of the entire process. Because *Manage Employee* Use Case has many options and possibilities and englobes different operation, it can be seen that the activity diagram is rather complex, having many turning points. The Team Leader can chose from a range of option to edit the data about the employee, from adding a non-existing employee, to hiding it from being displayed or deleting it from the system.



Another thing that can be observed is that the *Display message about missing non-optional* appears two times in the system. Even if in its essence it is the same message, it is triggered by different actions it needs to appear in two different positions separately.

As Activity Diagrams for each Use Case were developed the Analysis Phase has ended and the next was Design Phase were the focus will be to provide a more programming orientated developing a model that can be implemented.

4. Design

In this chapter, the structure of the system will be outlined through development of the Class Diagram and Sequence Diagram for the system. The focus will be on providing a blue print for the programming that will be used in the Implementation Phase.

With a focus on the future structure of the system a class diagram was developed, that can be analyzed in the Appendix D

The Class Diagram covers all the classes and methods that will be included in the system as well as the relationship between classes. A few significant things worth describing as they provide a vital coherence of the system are:

- *SystemAdapter Classes (Employee, Analysis and Week)* – that act as a buffer between the *Graphic User Interface*, the *FileIO Class* and the *System Class*;
- *FileIO Class* – that offers Data Saving capabilities to the system;

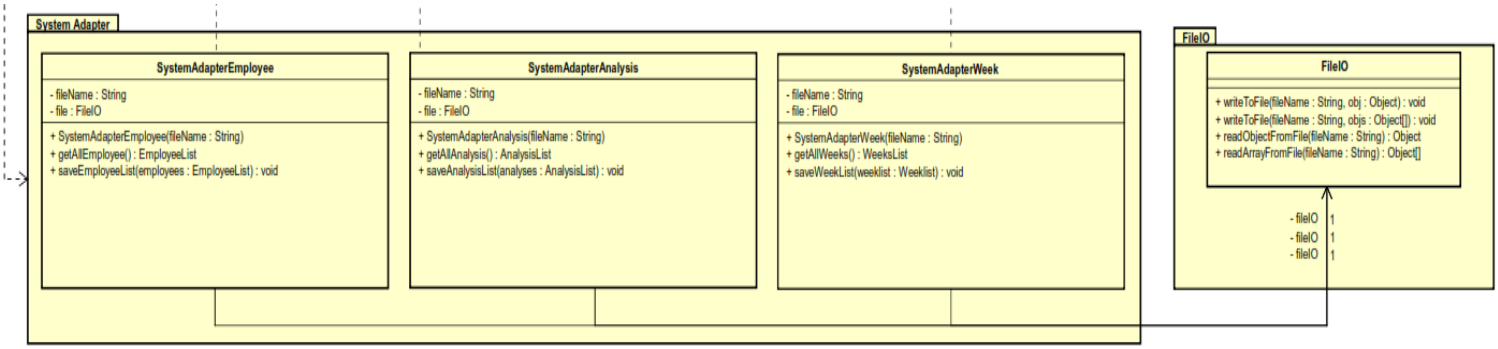


Figure 5 SystemAdapter and FileIO Class

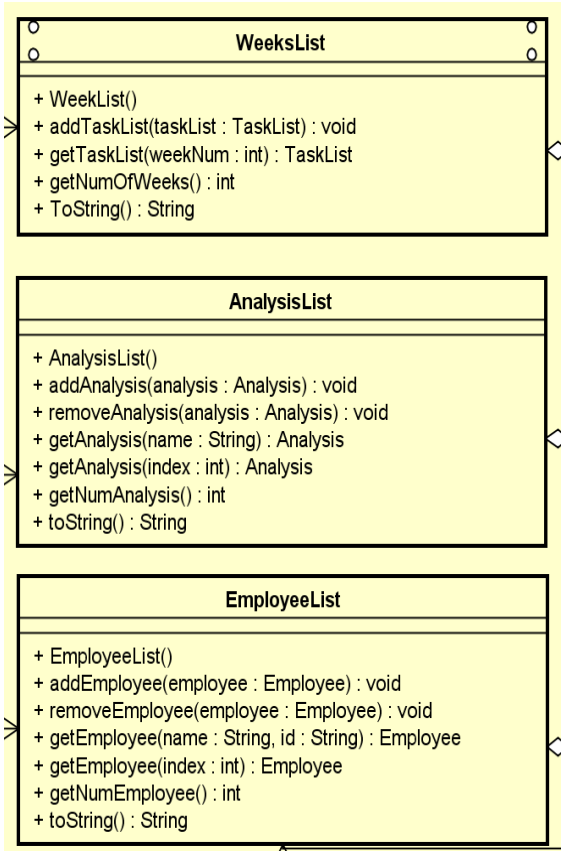


Figure 6 Week, Employee and Analysis List

- Rather than having a system that has as methods that work with arrays to organize weeks, tasks, employees and /or analyses, a *List Class* was developed for each parameter so it will take the functionality of the "array" method;
- Another important thing about the system is that instead of having the *SystemAdapter Classes* directly access and manipulate the *Lists Classes*, it was opted for having a middle class that will act as a "middle-man" between those two to ensure a better organization of the system and to create different layers of abstraction;

- One important thing is that the *Task* class provides in its essence (as it name says) a single analysis that needs to be performed by one or more employee on a certain date. Again to keep the things organized and easy to manipulate a *List Class* was made that takes all the tasks that need to be performed in one day, and another *List Class* was made that will keep 7 (or 5) of the *Task List* Objects that will represent one week. This was made to ensure an easier search and access to the information;

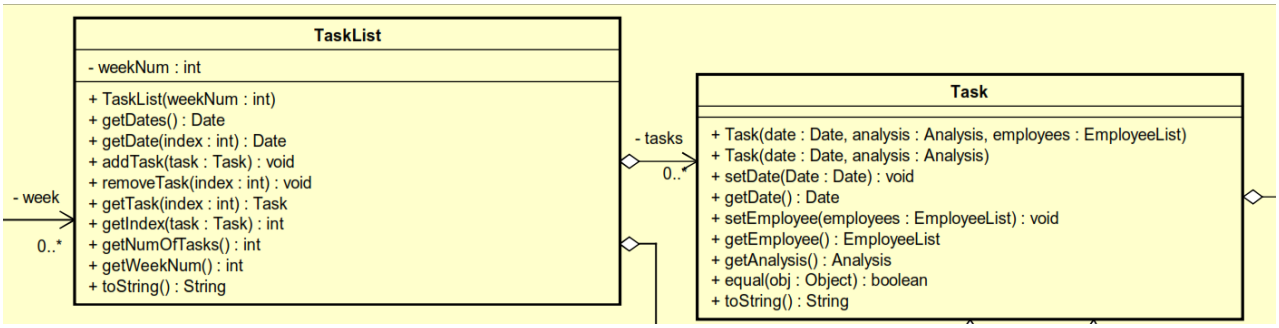


Figure 7 Task and TaskList Classes

- The *Task Class* has an *Employee List Class* but only one *Analysis Class*, this is because a task is considered an Analysis performed by one or more employees in a certain date;
- In most cases, the relationship between classes is an aggregation relationship as one class would be a part of the other class; there is a dependency relationship between *SystemAdapter Classes* and *System Class* that connects two different hemispheres of the project;

A Sequence Diagram was developed to understand better how a process, in this case, the edit employee function will be presented as it can underline the whole system and provide a good overview on how the system work. The whole sequence can be found in the Appendix E. It can be seen that the task involves many processes and goes across the whole system.

The process is triggered by the actor that choses to edit a certain employee, selected in the employee sheet.

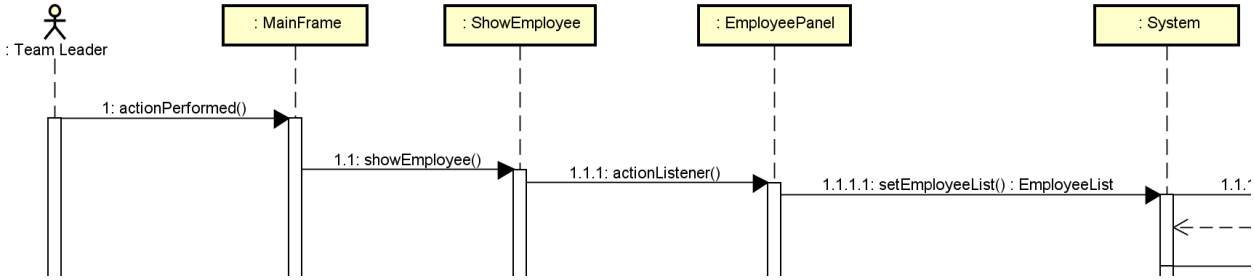
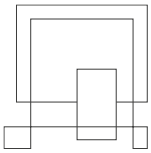


Figure 8 Edit Employee



The system first goes through the Graphic User Interfaces classes until it reaches the *System* class that is asked to set the Employee List that will be retrieved using the *SystemAdapterEmployee* class.

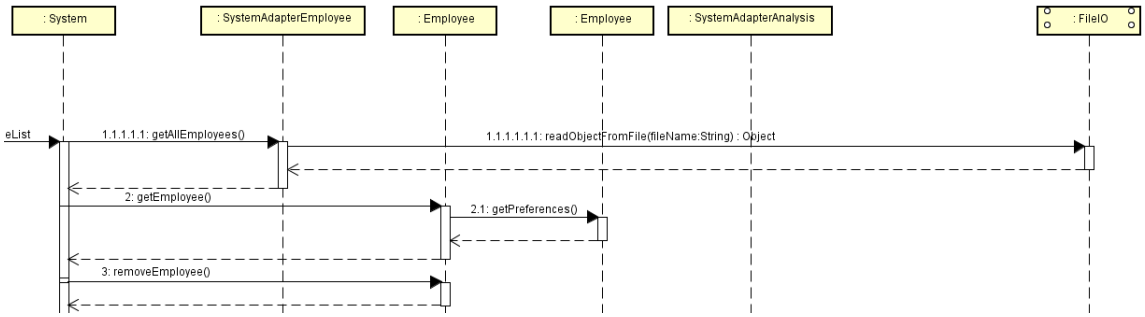


Figure 9 Search for the employee

The *System* gets the current list of the employees from the *SystemAdapterEmployee* that retrieves it using the *FileIO* class. As the *System* gets the list of all employees, it searches for the employee that needs to be edited and temporarily saves the preferences of the employee.

Afterwards that employee is removed from the *System* and lists, and a new employee is created with the new edited information provided by the actor.

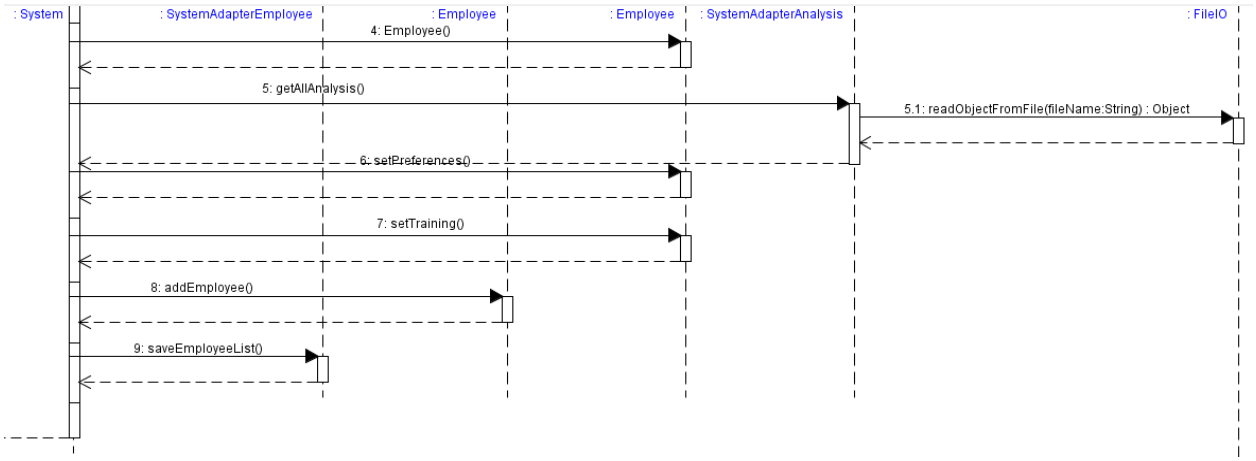
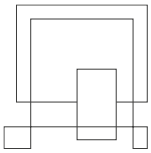


Figure 10 Add new employee

The *System* request the Analysis List from the list from the *SystemAdatperList* through the *FileIO* class in order to set up the training. The Preferences and Training are registered, finishing with adding the employee to the list and saving the list.



We opted for this procedure of retrieving data from a "to be edited" employee deleting the employee and registering a new one, rather than modifying the current employee, because of an easier way to implement in the future.

4.1. Website development

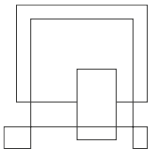
A website was developed for providing basic information about the team that works on the development of the software.
The website was developed using HTML, CSS and JavaScript technologies as well as different frameworks (Bootstrap and JQuery). The website is made of multiple pages that cumulate similar information and can be viewed in the Appendix F.

5. Implementation

The code was developed in a structured manner, providing a common organization for the classes and for the whole project, the entire source code can be seen in the Appendix G as well as the JAVA documentation can be viewed in the Appendix I. Next will be presented a few important implementations of the classes.
First, there are the basic classes that define the three main components of the system:

- Employee;
- Analysis;
- Task;

When defining an Analysis, the name and type must be stated, the type is used to set the total number of employees needed for the analysis. On the other hand, also, an Analysis has a specific number of employees for each day of the week used in the Staff Time Sheet, and that is initialized to zero as a default.



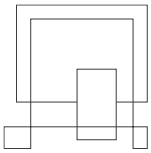
```
/**
 * Two argument constructor sets the name and type of the analysis and the standard number of needed employees
 * @param String name
 * @param type boolean False for Large/True for small
 */
public Analysis(String name, boolean type)
{
   this.name = name;
   this.type = type;

   //Sets up the default number of needed employees per day (for Staff Time Sheet) as 0;

   numOfEmployeeWeekDay = new int[7];
   for(int i = 0; i < numOfEmployeeWeekDay.length; i++)
   {
      numOfEmployeeWeekDay[i] = 0;
   }

   //Sets up the default total number of needed employees for the analysis based on the type of week (small/large)

   if(type)
   {
      setNumEmployee(4);
   }
   else
   {
      setNumEmployee(8);
   }
}
```

Figure 11 Analysis constructors

The employee has two arguments one using the name, email and id, and another using the name and id. In the last case the email is set to " ".

```
/**
 * Three argument constructor
 * @param String name
 * @param String email
 * @param String id
 */
public Employee(String name, String email, String id)
{
   this.name = name;
   this.email = email;
   this.id = id.toUpperCase(); //ID is always upper case

   //Setting up preferences and comments to their default " ", and initiating the training list
   this.preferences = " ";
   this.comments = " ";
   trainings = new ArrayList<Training>();
}

/**
 * Two argument constructor
 * @param String name
 * @param String id
 */
public Employee(String name, String id)
{
   this.name = name;
   this.id = id.toUpperCase();

 //Setting up preferences, comments and email to their default " ", and initiating the training list
   this.email = " ";
   this.preferences = " ";
   this.comments = " ";
   trainings = new ArrayList<Training>();
}
```

Figure 12 Employee constructor



One other feature of the system is that the employees, analysis and tasks, are manages through a list class.

An example of code is that when someone wants to add an employee to the list, first thing is that system checks if the employee already exists based on the name and ID, and does not add it to the list if it already exists.

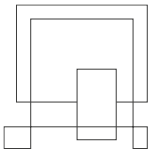
```
/**
 * Adds an employee to the list if the employee does not already exists
 * @param Employee employee
 */
public void addEmployee(Employee employee)
{
   boolean flag = false;

   for(int i = 0; i < employees.size(); i++)
   {
     if(employees.get(i).equals(employee))
     {
        flag = true;
     }
   }

   if(!flag)
   {
      employees.add(employee);
   }
   else
   {
      System.out.println("Already Exists");
   }
}
```

Figure 13 adding an employee

The software first starts with the login window that asks the actor to login in. Next will be presented the procedure of checking the login and password.

```
public void actionPerformed(ActionEvent e)
{
   if (e.getSource() == loginButton)
   {
      String login = loginTextField.getText();                    //Retrieves the login
      String password = String.valueOf(passwordField.getPassword()); //Retrieves the password and makes it into a String

      //Checks credentials and initializes the mainFrame
      if(login.equals("admin") && password.equals("admin"))
      {
         MainFrame mainGUI = new MainFrame();
         setVisible(false); //Becomes invisible afterwards
      }
      else
      {
         JOptionPane.showMessageDialog(new JFrame(), "Wrong login and/or password");

         loginTextField.setText("");
         passwordField.setText("");
      }

   }
   else if(e.getSource() == cancelButton)
   {
      System.exit(0);
   }
}
}
```

Figure 14 Login process



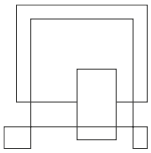
For the password field instead of using a text field we used a password field provided by the Java Swing, which returns an array of characters as a password. After that it is retrieved as a String using the ValueOf() method. The password and login are checked for matching. If they match mainframe of the system is started, otherwise a message is displayed showing that the password is wrong.

Each sheet is displayed in the system as an internal frame, next will be presented the internal frame of the employee sheet.

```
public class ShowEmployee extends JInternalFrame
{
   private static ShowEmployee showEmployee;
   private EmployeePanel allEmployeePanel;

   /**
    * Using singleton pattern to ensure that only one internal frame of the same type is displayed at a time
    * @return showAnalysis
    */
   public static ShowEmployee getInstance()
   {
      if(showEmployee == null)
      {
         showEmployee = new ShowEmployee();
      }
      return showEmployee;
   }

   /**
    * No argument constructor
    */
   public ShowEmployee()
   {
      super("Employee");

      allEmployeePanel = new EmployeePanel();
      add(allEmployeePanel, BorderLayout.CENTER);

      setClosable(true);
      setMaximizable(true);
      setIconifiable(true);
      setResizable(true);
   }
}
```

Figure 15 ShowEmployee class

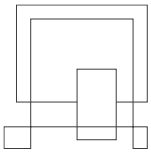
The singleton patter is used to ensure that only one sheet of the same type is opened at the same time. The solution was inspired from a suggestion to a similar issue on StackOverflow and adapter to the system (user2204480, 2018). The sheet uses employee's panel that contains the table with data and functionalities. Next will be presented the process of creating/adding a new employee. It can be followed and compared with the sequence diagram because a part of the sequence diagram corresponds to the code below.



```
public void actionPerformed(ActionEvent e)
{
   if(e.getSource() == createButton)
   {
      CreateNewEmployee newEmployeePanel = new CreateNewEmployee(); //Initializing the panel for adding employees

      int n = JOptionPane.showConfirmDialog(null, newEmployeePanel, "Enter data about the new employee", JOptionPane.OK_CANCEL_OPTION); //Displaying th

      //Checking for the okay option
      if(n == JOptionPane.OK_OPTION)
      {
         String name = newEmployeePanel.nameField.getText();  //Retrieves the name from the field
         String id = newEmployeePanel.idField.getText();      //Retrieves the ID from the field
         String email = newEmployeePanel.emailField.getText();//Retrieves the email from the field

         //Checks for null input to avoid null exceptions
         if(name == null)
         {
            name = " ";
         }
         else if(id == null)
         {
            id = " ";
         }
         else if(email == null)
         {
            email = " ";
         }
         else
         {
            Employee newEmployee = new Employee(name, email, id); //Creates new employee with the retrieved data

            system.setAnalaysisList(systemAdapterAnalysis.getAllAnalysis()); //Gets the analysis list from the bin
            AnalysisList analyses = system.getAnalysisList(); //Initiates the analysis list
            newEmployee.setTraining(analyses); //Sets the default training for the analyses

            system.setEmployeeList(systemAdapterEmployee.getAllEmployee()); //Gets the employee list from the bin
            system.getEmployeeList().addEmployee(newEmployee); //Adds the new employee to the list
            systemAdapterEmployee.saveEmployeeList(system.getEmployeeList()); //Saves the list

            //Updates the table
            updateEmployeeTable();
         }
      }
   }
```

Figure 16 adding an employee from the employeePanel()

As it can be observed, rather than modifying the existing employee, when the system edits an employee, it takes the data from the employee, temporarily saves it and then creates a new employee with the edited data.

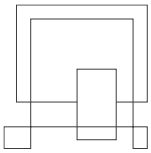


6. Test

The testing of the software was done based on the requirements developed before; it will show how many of the requirements were achieved. The test will ensure that when a requirement can be tested, it is either Fully Functional (No issue); Functional (Minor issues); Partially Functional (Major issues); Not Functional (Does not perform in anyway); Cannot be tested (it is absent from the software).

The testing was done by trying the actual software (the software in a .jar form can be found in the Appendix H and the User Guide can found in the Appendix J), and trying to do what requirement specified that it should do.

REQUIREMENT	USE CASE
1	Tested – Fully Functional
2	Tested – Fully Functional
3	Tested – Fully functional
4	Tested – Functional
5	Tested – Functional
6	Tested – Functional
7	Tested – Functional
8	Tested – Functional
9	Tested – Fully functional
10	Tested – Not functional
11	Tested – Fully functional
12	Tested – Fully functional
13	Tested – Functional
14	Cannot be tested
15	Cannot be tested
16	Tested – Fully functional
17	Cannot be tested



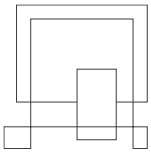
The final count of requirements can be summarized as follows:

- 7/17 – Fully Functional;
- 6/17 – Functional;
- 1/17 – Not Functional;
- 3/17 – Cannot be tested;

7. Results and Discussion

In this chapter, the results of the software will be discussed, which of the requirements are ready for the actor to be used and which are not. From the requirements point of view, a requirement is ready to use when it performs without any issues.

1. The first requirement is that the software should provide a possibility of viewing all four sheets (Schedule, Staff Time, Training, Employee Preferences) at the same time, as tested the software does that, it provides the possibility to view all the sheets, along with Show Employees and Show Analysis;
2. The actor must be able to register/edit/delete data about employee – the software provides functionality for each aspect of this matter;
3. The actor must be able to register/edit/delete data about analysis – the software provides functionality for each aspect of this matter;
4. When registering an employee the actor must state the name, e-mail and ID – the requirement is partially achieved, when registering an employee the actor is asked to introduce name/email/id but if the actor does not than the system will fill in the fields with blank to avoid null exceptions;
5. The employee parameter must have other fields as – the software provides ways of adding/editing employees training and preferences, also provides possibility for adding comments and vacation but it is not integrated with the graphical user interface, so it is not ready for the customer;
6. The name/type/required number of employee must be stated when registering an analysis – the software does that, again if the fields are left empty software will fill in as blank;



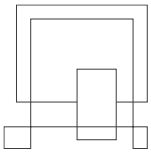
7. The system must update stored information about the employee and analysis – the system stores and updates data;
8. The system must check and prevent the user for repetitions – the system does that when it is needed;
9. The actor must be able to view / edit the sheets individually – it can be done without any issues;
10. Ability to edit the schedule sheet – it is neither properly developed nor integrated in the graphic user interface;
11. Ability to edit preferences – software provides this functionality;
12. Ability to edit training - software provides this functionality;
13. Ability to edit staff time - software provides this functionality;
14. Must be able to view past/current/future – not applicable due to missing schedule capabilities;
15. Search capabilities – does not provide;
16. Cancel any process at any time – software provides this functionality;
17. Setting – does not provide;

Overall, it can be concluded that most of the requirements are achieved, even if unfortunately one the most important Schedule Sheet was not achieved.

Most of the requirements that were not achieved or only partially achieved is due to lack of knowledge or/and time. As an example, the last requirement Settings capabilities were not provided due to the lack of time, as most attention was payed for more important requirement.

The reason for not providing Schedule functionalities is the fact that the software was supposed to use other functionalities in order to make a schedule, this meaning that before the Scheduler could have been developed all of the other functionalities must have been ready to use.

Search capabilities again were not one of the core functionalities and was left to be done when everything else was working in a proper way.



8. Conclusions

When summarizing everything that was done, beginning with the introduction to the project case and finish up with the testing, it can be observed this project has achieved most of what it was supposed to.

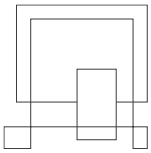
When defining the requirement a SMART model was used to ensure an easier way of testing, which proved to be extremely useful in the testing chapter. The requirement described what the system must do and what should be tested in the end.

As the requirements was defined, in the analysis phase, a more in-depth look to the structure of the project was payed. The use cases and description for them was developed providing a prime view of how the system should look and perform. Activity diagrams were developed for the use cases to observe how they should perform on a step-by-step manner.

In the design phase the system started to take even more shape, the sequence and class diagrams were developed providing a structure for the shape as well as a more detailed view of what the system will include and how it will work.
In the implementation based on what was developed before a program was developed in JAVA programing language. All the phases above presented their specific challenges that had to be overcame for the system to exist.

In the last two chapters the performance of the system was assessed, which requirements were achieved and functional, and which were not. It is important to know what works and what does not to be able to anticipate different issues.

Overall, the system does perform some of the main features; unfortunately, it cannot be used due to the absence of an integrated graphical user interface for the Schedule Sheet.



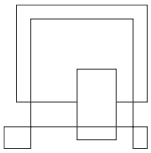
9. Project future

When defining the project the main aim was to provide a software system that will ease and automatize the planning work subsequently freeing up time and effort for the Chemistry Department of Eurofins Steins Laboratory A/S (Check Project Description Appendix K). Analysing if the project did fulfil its purpose, it cannot be stated that it did, due to the lack of many functionalities needed to do so.

Due to the lack of time and knowledge, as well as efficiency (the time-line of the project development can be assessed in the Appendix L), the system still needs to be worked upon, in order to become what it was supposed to. The functionalities that are there, are good and useful, the only issue is that they are not enough for the software to be used in a practical way.

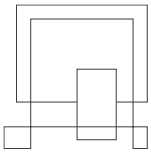
In the future, adding more functionalities to it, as well as integrating the existing functionalities with a graphical user interface would greatly improve its capabilities to truly become a work-planning tool.

In order for the project to become scalable, it is needed a design as well as code optimization and proof testing to ensure that any type of issues were addressed and solved.



Source of Information

- Eurofins Steins Laboratory A/S, 2018. *Presentation of work planning tool.* Horsens, s.n.
- Henriksen, A. R. & Andersen, M. W., 2018. *Summary presentation and delimitations.* Horsens: s.n.
- NISO, 2010. *Scientific and Technical Reports -,* Baltimore: National Information Standards Oganization.
- Schedlbauer, M., 2011. *BA Times.* [Online]
  Available at: https://www.batimes.com/articles/the-quest-for-good-requirements.html
  [Accessed 22 11 2018].
- user2204480, 2018. *Stack Overflow.* [Online]
  Available at: https://stackoverflow.com/questions/15461002/only-open-one-jinternalframe-in-one-time
  [Accessed 12 15 2018].
- VIA Engineering, in preparation. *Confidential Student Reports,* s.l.: s.n.
- Viuff, A., 2018. *Work Planning Tool* [Interview] (07 September 2018).



Appendices

Appendix A – Use Case Diagram;
Appendix B – Use Case Description;
Appendix C – Activity Diagram;
Appendix D – Class Diagram;
Appendix F – Website;
Appendix E – Sequence Diagram;
Appendix G – Source Code;
Appendix H – JAR file;
Appendix I – JAVA Documentation;
Appendix J – User Guide;
Appendix K – Project Description
Appendix L – Project Gantt chart