

VIA University
College

SEP1X-A18 Project Report

Eurofins Management Software.

Members:

Denis-Alexandru Turcu (279955)

Lau Ravn Nielsen (280337)

Sébastien Malmberg (279937)

Tamás Fekete (266931)

Supervisors:

Mona Wendel Andersen

Allan Rune Henriksen

Alexis Clare Walhovd

Number of characters:

30.237

Deadline:

12:00 19 / 12 / 2018

Contents

ABSTRACT	4
INTRODUCTION	5
REQUIREMENTS	6
FUNCTIONAL REQUIREMENTS	6
NON-FUNCTIONAL REQUIREMENTS	6
ANALYSIS	7
USE-CASE DIAGRAM	7
USE-CASE DESCRIPTION	8
USE CASE: MANAGE EMPLOYEE(S)	8
USE-CASE: MANAGE SCHEDULES	9
USE-CASE: VIEW EMPLOYEE LIST	10
USE-CASE: VIEW SCHEDULES	10
ACTIVITY DIAGRAM	11
ANALYSIS CLASS DIAGRAM	12
DESIGN	13
DESIGN CLASS DIAGRAM	13
SEQUENCE DIAGRAM	14
UI DESIGN CHOICES	15
IMPLEMENTATION	18
TEST	22
RESULTS AND DISCUSSION	24
RESULTS	24
FUNCTIONAL REQUIREMENTS	24
NON-FUNCTIONAL REQUIREMENTS	24
DISCUSSION	25
CONCLUSION	26
PROJECT FUTURE	27

SOURCES OF INFORMATION **28**

APPENDICES **28**

Figure 1 - Use-case diagram	7
Figure 2 - Use-case description - Manage employee(s)	8
Figure 3 - Use-case description - Manage schedules	9
Figure 4 - Activity diagram - Branch sequence ' Assign employee to schedule'	11
Figure 5 - Analysis class diagram	12
Figure 6 - Design class diagram	13
Figure 7 - Sequence diagram – getAllSchedules()	14
Figure 8 - Main panel GUI	15
Figure 9 – Manage employee window (All employee tab)	16
Figure 10 - Viewing a schedule	17
Figure 11 - Assign employee tab	17
Figure 12 - Main panel	18
Figure 13 - Schedule class fields	18
Figure 14 – Schedule constructor if-statements for types 'L' and 'S'	19
Figure 15 - Assign employee method	19
Figure 16 - WorkScheduleTable rows 147-161	20
Figure 17 - Action listener of the buttons inside the table	20
Figure 18 – Code showcasing how a schedule is created when new rows are added	21
 Table 1 - Testing criteria and implementation status	 22
Table 2 - Manage employee testing	23
Table 3 - Manage schedule testing	23

Abstract

The aim of this project is to create a program which will substitute Eurofins current work-planning tool. Eurofins, a company which conducts analysis on feed and food products, wishes for the new tool to be less time-consuming to use. To achieve this objective a single user system was developed in java from a set of established functional and non-functional requirements. In addition, use case description were made from the requirements which aim was to clarify how the user would utilize the program. The functionality was later implemented in a graphical user interface which design aims to streamline the vital content. The result shows that the system fails to fulfil all the requirements. Although the system is deemed functional, and the crucial requirements have been implemented, the lack of important features such as “Change analysis”, “hide employees” and “Add analysis” sets a low limit to the system’s functionality and can therefore not be considered a solution to Eurofins request.

Introduction

Eurofins is a company consisting of a network of laboratories, which are located all over the world. In these laboratories they conduct chemical, microbiological and molecular biological analyses. Eurofins Steins Laboratorium A/S located in Vejen, focuses on analysis of food and feed products. There is a total of 325 employees in Vejen, whereof 60 employees belong to the chemistry department of Steins Laboratory. The chemistry department has addressed an internal problem, implying that the issue is within the company. They require a new way of managing their employees (Agnete, 2018).

Eurofins Steins Laboratory A/S chemistry department wishes for a new tool to replace the company's current way of displaying their employee schedule.

Currently they are using excel and in excel they have 4 separate sheets, the four sheets consist of the work-plan which is the major working document, a staff time overview, a training overview and preferences from annual performance review. Together these tables make the entire employee schedule.

Eurofins has documented a set of issues they encounter daily whilst utilising Excel. First off, only one person can edit and add information at a time which has proven to be inconvenient and time consuming. Therefore, they wish for a solution which allows editing and viewing simultaneously. Agnete (2018) stated that preferably only team leaders should have the authority to edit. Moreover, according to Agnete (2018) the chemistry department finds their current way of displaying data to be quite troublesome whilst juggling information. Instead they would prefer all data to be displayed in one sheet.

Furthermore, there is a need for a mobile application which would allow for a more flexible workspace, granting employees the ability to view their schedule no matter their location. Additionally, Eurofins has a series of requests that they would like implemented, such as, the ability to jump to the current date, the ability to search/navigate the schedule with ease. Along with a short list of minor features. Nevertheless, this should all be made possible without installing any software.

Excel does not meet the requirements stated above which confirms its inefficiency in relation to Eurofins.

Maintaining the sheets up to date is very time consuming and Eurofins wishes for a cleaner and more intuitive interface. This can be recognised as a system migration problem.

Legacy systems pose a lot of issues on the information flow within a company. A legacy system is basically “any information system that significantly resists modification and evolution” (Wu, et al., 1997). Because of this resistance towards improvement, and other critical flaws such as unintuitive interfaces and maintenance costs, companies are now choosing to migrate to new, and more efficient solutions.

A tool which meets the new requirements set by Eurofins, would save them a substantial amount of time, as well as energy. The time saved could be used elsewhere, perhaps to increase the growth of the company or conserve money, etc.

Requirements

The requirements are deducted based on the first customer meeting we had with Eurofins. The system is a single user system therefore its main and only actor is the Team leader. The functional requirements defined below, are written in prioritized order and mainly establishes the system requirements, as well as what authority the team leader is given.

Functional requirements

1. The system must allow the creation of a new schedule.
2. The system must allow multiple schedules to be created and stored.
3. The system must store and allow the visualization of the information corresponding to Eurofins old schedules (Work plan, Training schedule, Staff time overview and Preferences from annual performance review).
4. The system must store and allow the visualization of all Employees.
5. The system must indicate whether an Employee is available, on vacation, sick or busy.
6. The system must allow a Team leader to add and remove/hide employees.
7. The system must allow a Team leader to assign available employees to any task.
8. The system must allow multiple employees to be assigned to the same task.
9. The system must be able to store comments and analysis.
10. The system must allow Team leader to change, add and remove comments or analysis.
11. The system must allow a user to navigate between the different schedules.
12. The system must store outdated schedules for minimum 1-year, current schedules and future schedules.
13. The system must allow the user to view outdated schedules, current schedules, and future schedules.
14. The system should inform the user on what type of schedule is being viewed and the corresponding dates.
15. The system should allow the visualization of different colours to simplify the interface.

Non-functional Requirements

1. System must be a single-user Java application
2. All four instances/schedules must be implemented in a single graphical user interface (GUI).
3. System must be manageable with keyboard and mouse

Analysis

Eurofins steins laboratory wishes for a new work-planning tool. The goal of the new tool is to conserve time and ease their planning. See definition of purpose, appendix A. The goal will be achieved by establishing a set of functional and non-functional requirements for the program in creation. The following diagrams and use cases will establish the functionality of the program.

Use-case Diagram

The use-case diagram shows the connections between the different use-cases within the program and the user.

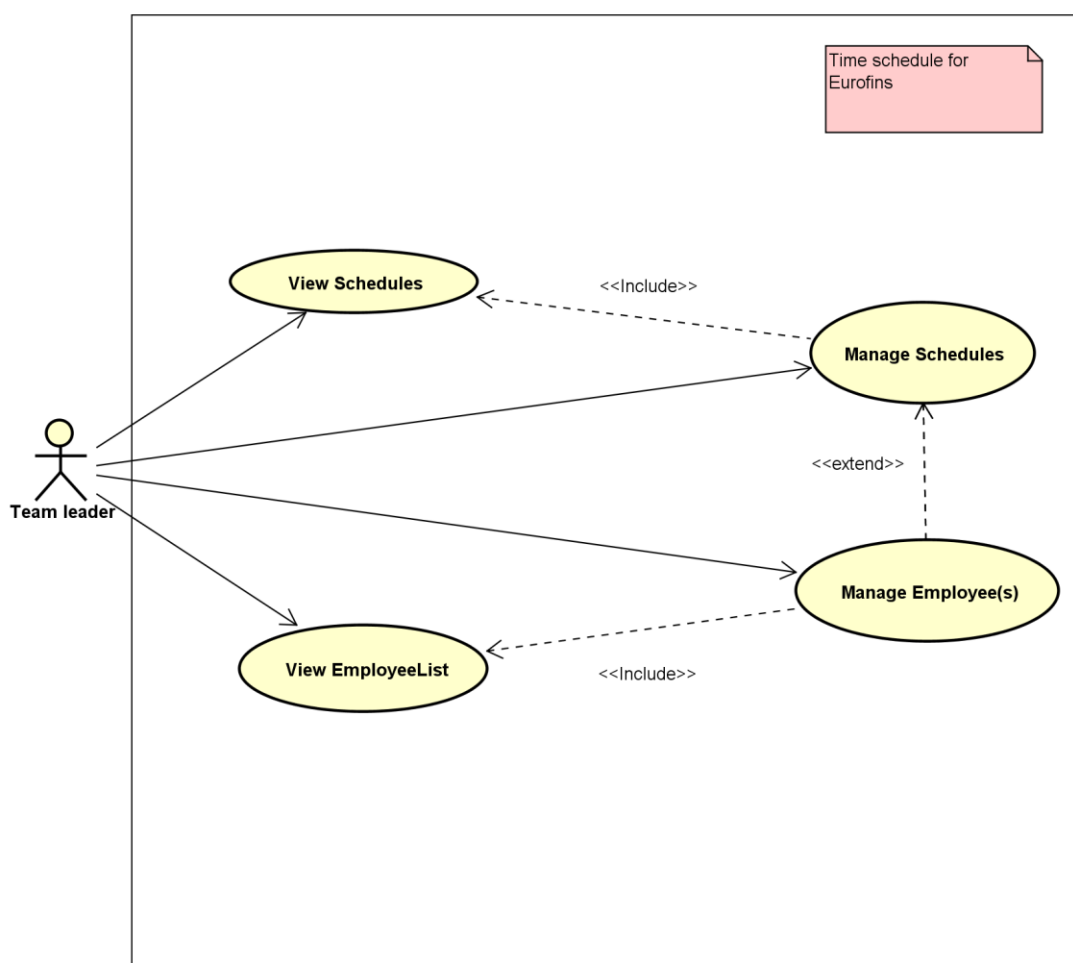


Figure 1 - Use-case diagram

Use-case Description

Based on the use-case diagram we have made detailed use-case descriptions of what happens in the different use-cases:

Use case: Manage Employee(s)

UseCase	Manage Employee(s)
<i>Summary</i>	Team leader can manage employee(s), by either, removing adding or edit employee information.
<i>Actor</i>	Team Leader
<i>Precondition</i>	
<i>Postcondition</i>	
<i>Base sequence</i>	1. Team leader selects Manage Employees
<i>Branch sequence</i>	Add an employee: <ol style="list-style-type: none"> 1. As base sequence. Team leader chooses to add employee. System display window in which the user can insert information about the new employee. Team leader inserts information. Team leader choose to save. System stores the new employee. Remove employee: <ol style="list-style-type: none"> 1. As bade sequence. Team leader selects an employee from list. Team leader choose to remove employee. System removes the employee. Change Employee Data: <ol style="list-style-type: none"> 1. As base sequence. Team leader selects an employee from list. Team leader chooses what data that needs changing (e.g. phone number or address) New information is inserted. Save is selected Changes are stored.
<i>Exception sequence</i>	Exception sequence: Added employee already exists: <ol style="list-style-type: none"> 1 as base sequence. 1-5 in branch sequence 'Add employee' System returns error message "Added employee already exists" Return to step 4. In branch sequence 'Add employee'
<i>Sub UseCase</i>	
<i>Note</i>	User can cancel at any time.

Figure 2 - Use-case description - Manage employee(s)

Use-case: Manage Schedules

UseCase	Manage Schedules
<i>Summary</i>	Team leader can manage employee(s), by either assign employees to schedule, creating, or removing schedules.
<i>Actor</i>	Team Leader
<i>Precondition</i>	
<i>Postcondition</i>	
<i>Base sequence</i>	1. Team leader selects manage schedule.
<i>Branch sequence</i>	Create schedule: <ol style="list-style-type: none"> 1. As base sequence. Team leader inserts a name for the schedule. Team leader inserts a date for the schedule. If needed team leader selects a button with 'Add rows'. If needed team leader can inserts a comment to the schedule. Team leader selects a schedule type. Team leader selects create schedule. The schedule is saved into the system. Remove schedule: <ol style="list-style-type: none"> 1. As base sequence. Team leader views the schedule list. Team leader selects a schedule to be removed. Team leader selects 'Delete'. The schedule is removed from the system. Assign employee to schedule. <ol style="list-style-type: none"> As base sequence. Team leaders selects a schedule. Team leader selects view schedule. Team leader selects the day and the analysis where he or she wants to assign an employee. Team leader selects an employee from the list of employees. Team leader assigns the employee. The employee is assigned to that schedule.
<i>Exception sequence</i>	Exception sequence: Invalid name. <ol style="list-style-type: none"> 1. As base sequence. 1-7 in branch sequence 'Create schedule'. Display error "The chosen name is invalid: Either empty or already taken." Return to step 2. In branch sequence 'Create schedule'.
<i>Sub UseCase</i>	
<i>Note</i>	User can cancel at any time.

Figure 3 - Use-case description - Manage schedules

Use-case: View employee list

UseCase	View employee list.
<i>Summary</i>	Team leader can view the employee list.
<i>Actor</i>	Team Leader
<i>Precondition</i>	
<i>Postcondition</i>	
<i>Base sequence</i>	<ol style="list-style-type: none"> 1. Team leader selects manage employee. 2. Team leader selects 'Employee list' 3. Employee list is visualised.
<i>Branch sequence</i>	
<i>Exception sequence</i>	
<i>Sub UseCase</i>	
<i>Note</i>	User can cancel at any time.

Table 2 - Use-case description - View employee list

Use-case: View Schedules

UseCase	View schedules.
<i>Summary</i>	Team leader can visualise a schedule.
<i>Actor</i>	Team leader
<i>Precondition</i>	
<i>Postcondition</i>	
<i>Base sequence</i>	<ol style="list-style-type: none"> 1. Team leader selects 'Manage schedules. 2. Team leader selects a schedule. 3. The schedule is visualised.
<i>Branch sequence</i>	
<i>Exception sequence</i>	
<i>Sub UseCase</i>	
<i>Note</i>	User can cancel at any time.

Table 3 - Use-case description - View schedules

Activity diagram

The purpose of activity diagrams is to give a visual representation of what happens in the different use-cases. A full list of activity diagrams can be found in appendix B.

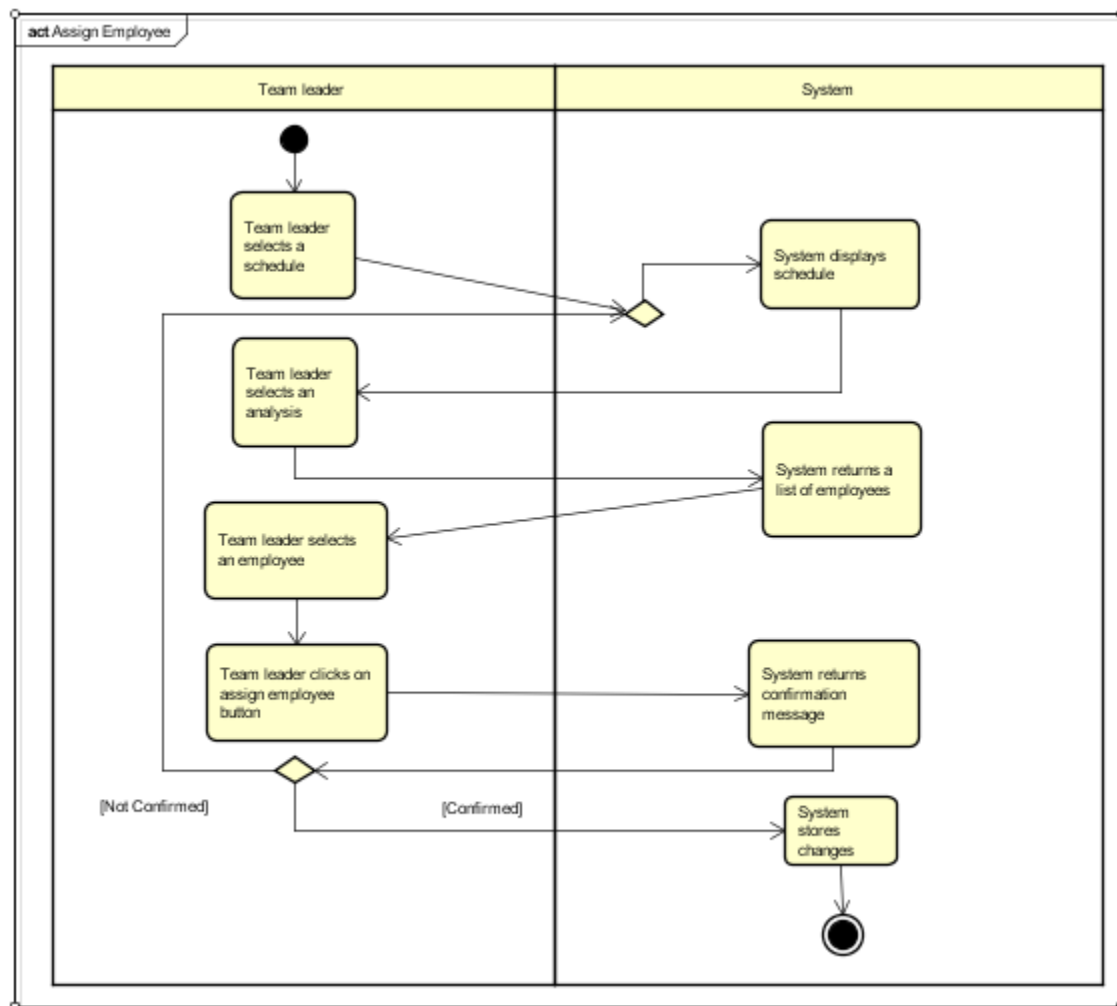


Figure 4 - Activity diagram - Branch sequence 'Assign employee to schedule'

The activity diagram above shows the branch sequence 'Assign employee to schedule' in the use-case 'Manage schedules'.

1. When the team leader selects a schedule, the selected schedule tab is shown first.
2. The team leader then selects an analysis, corresponding window will display the employee list.
3. The team leader selects an employee from the corresponding employee list.
4. When the team leader selects "Assign Employee", the window should close, and the employee is assigned to the analysis inside of the schedule when the confirmation message is approved.

Analysis class diagram

An analysis class diagram has been made based on the prior analysis work that has been done.

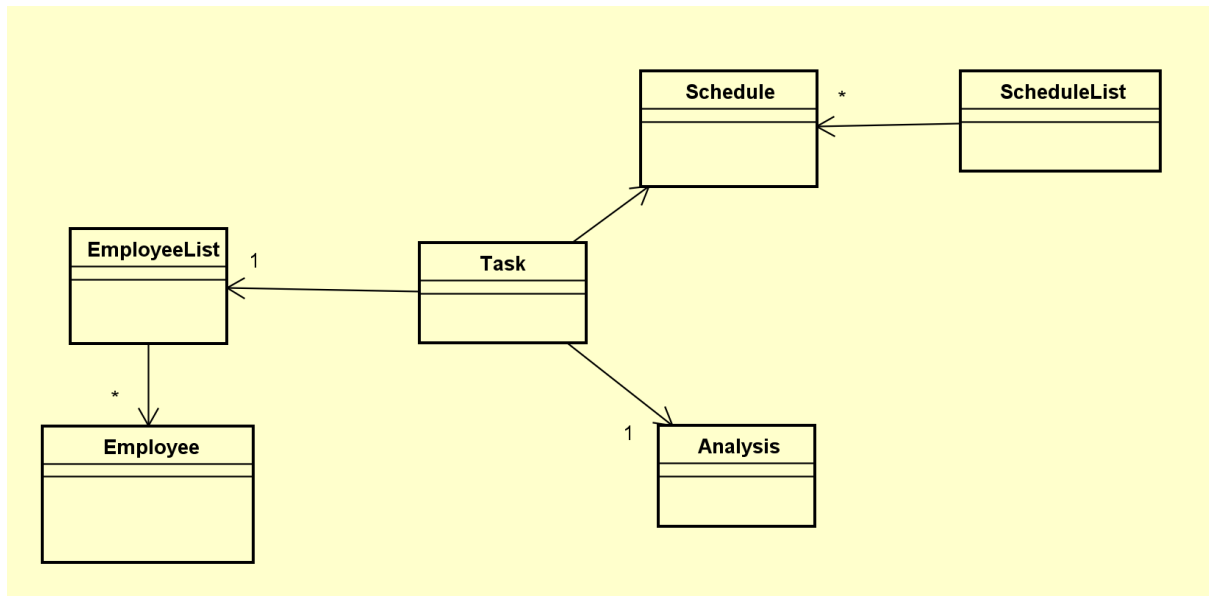


Figure 5 - Analysis class diagram

Design

Design was used to go more in-depth with our analysis and design a the structure and functionality of the program with the analysis phase kept in mind.

Design class diagram

The design class diagram is an extension of the analysis class diagram, where it has been expanded upon to be ready for implementation. Thus, containing all fields and methods for our program.

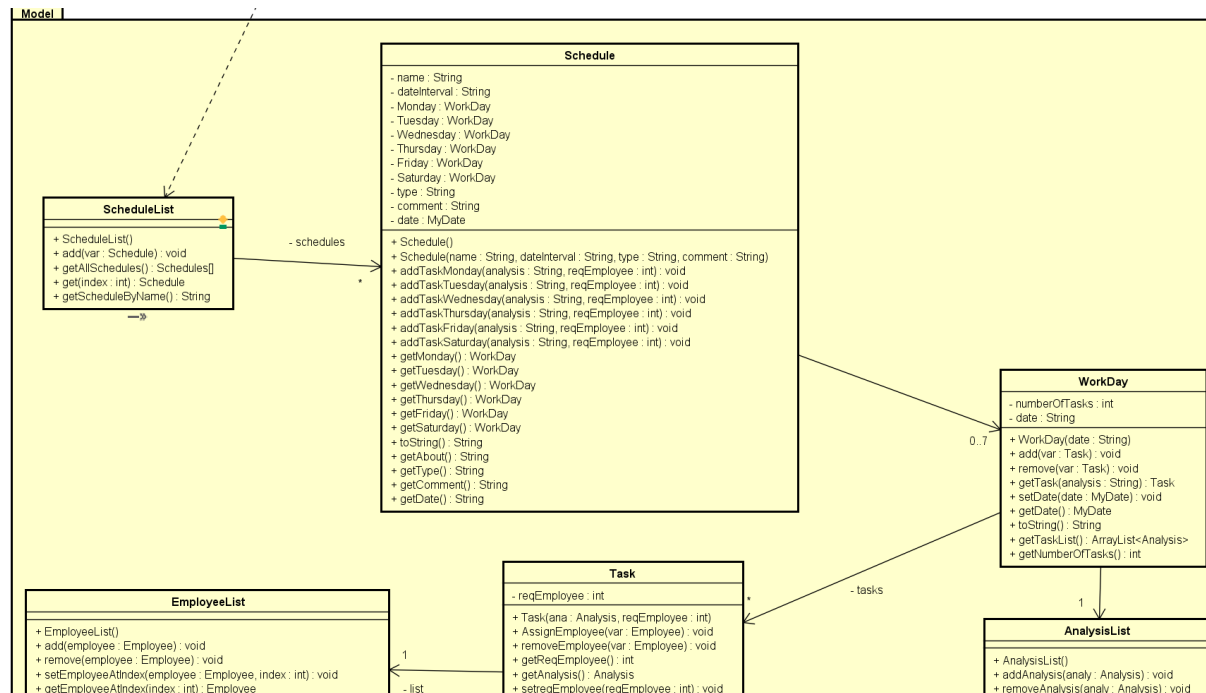


Figure 6 - Design class diagram

The remaining parts of the design class diagram can be seen in appendix C.

Sequence diagram

The sequence diagram below shows all the phases of how and where the program searches for all the schedules.

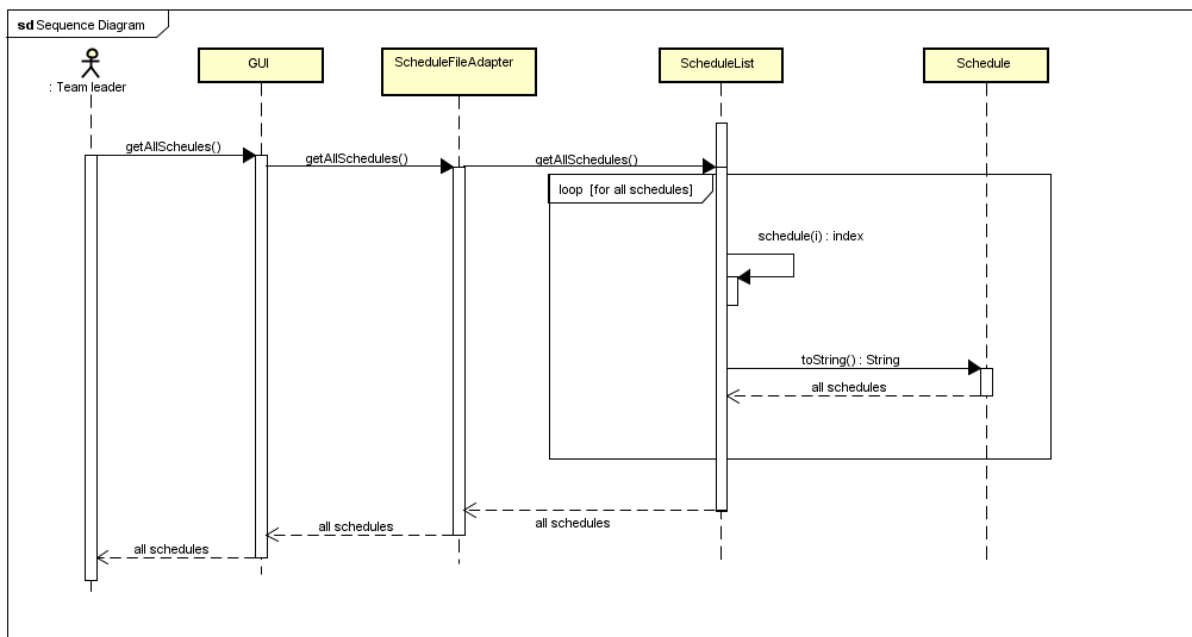


Figure 7 - Sequence diagram – `getAllSchedules()`

UI Design choices

The Graphical User Interface was written in Java using eclipse.

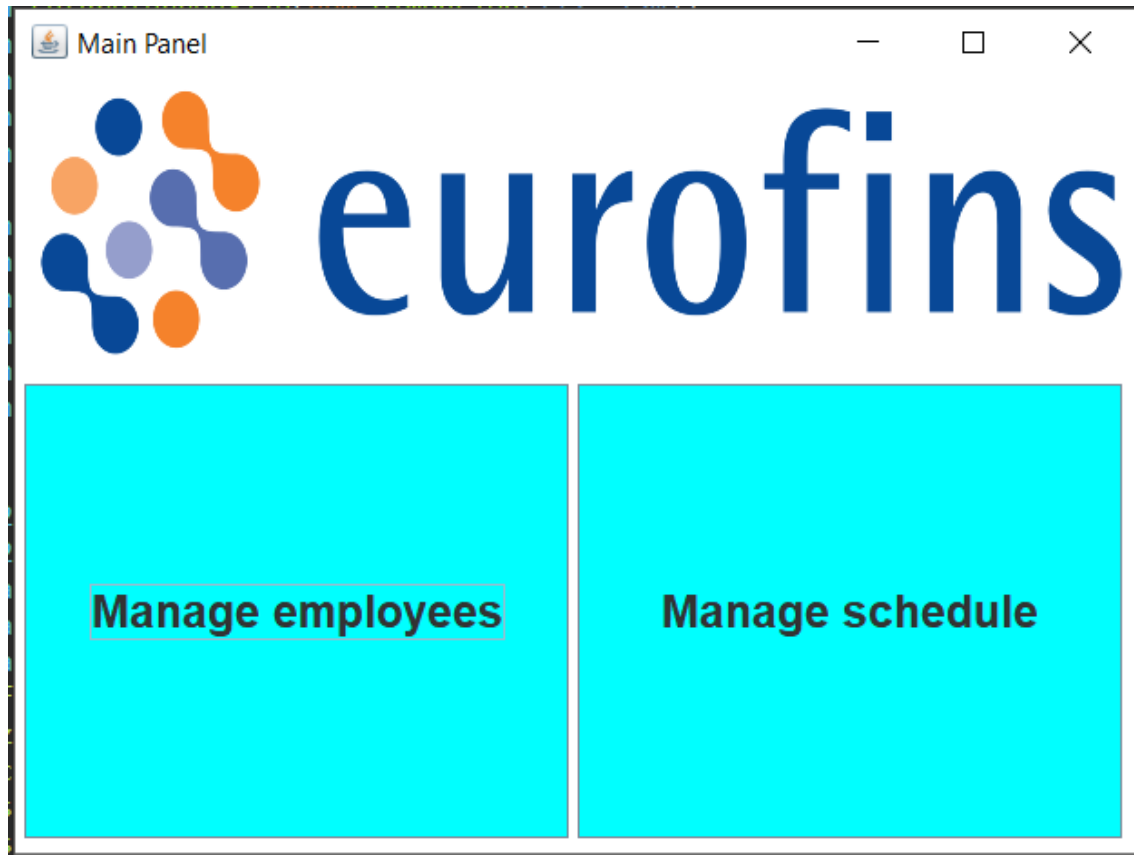


Figure 8 - Main panel GUI

The overall design for this window was chosen for its simplicity. The window should serve the user as a tool to get quick access to manage employees and manage schedules for they are our two main use cases.

Employee File Adapter GUI

File Edit About

All Employees Add Employee Remove Employee

First name:
Steve

Last name:
Stevenson

Phone:
070 934 86 00

Address:
Hybenvej 133

Schedules:

Analysis:
[Fat, Protein]

Comments:
Starts on monday!

Date hired:
15/12/2018

Edit

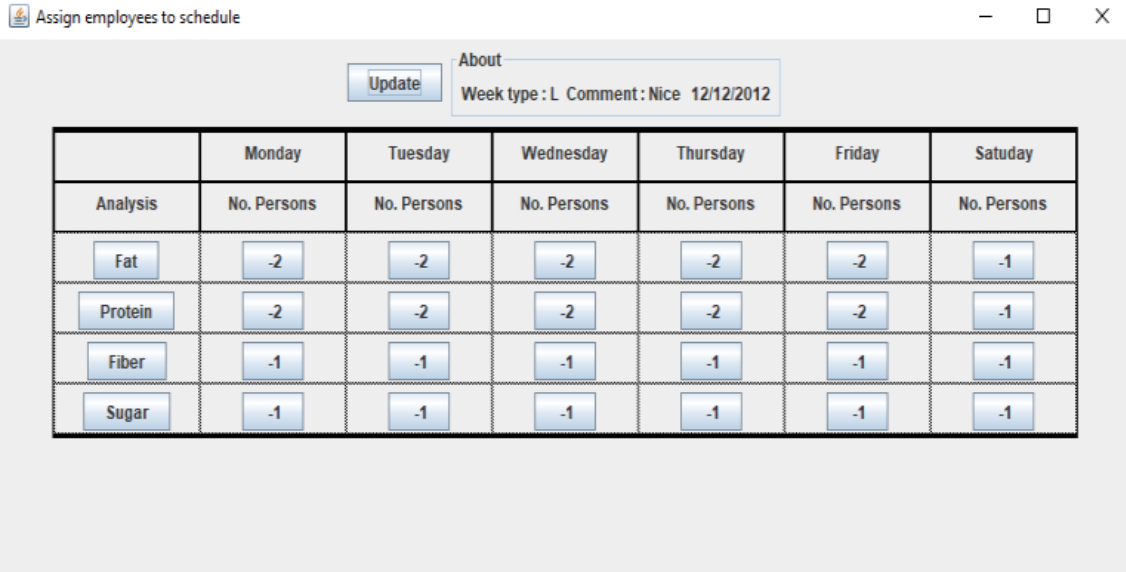
Save

Sebastien Malmberg (SM)
Steve Stevenson (SS)

Update

Figure 9 – Manage employee window (All employee tab)

Figure X displays the “Manage Employee” window which appears when the button “Manage employees” is pressed inside the main panel. All the employees are visualized inside a selectable JList. The overall UI design was chosen due to its efficiency. Because the window now covers the functional requirement 6 and partly the functional requirement 10, it allows the user an overview of the employees whilst simultaneously able to edit information regarding a selected employee.



	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Analysis	No. Persons	No. Persons	No. Persons	No. Persons	No. Persons	No. Persons
Fat	-2	-2	-2	-2	-2	-1
Protein	-2	-2	-2	-2	-2	-1
Fiber	-1	-1	-1	-1	-1	-1
Sugar	-1	-1	-1	-1	-1	-1

Figure 10 - Viewing a schedule

Figure X illustrates the Assign employee to schedule window, which opens when a schedule is selected, and the “View” button is pressed inside the View schedule panel. The schedule design is inspired by Eurofins work plan template. It displays all the days in a week, what types of analysis which needs attending and the number of employees required for each task.

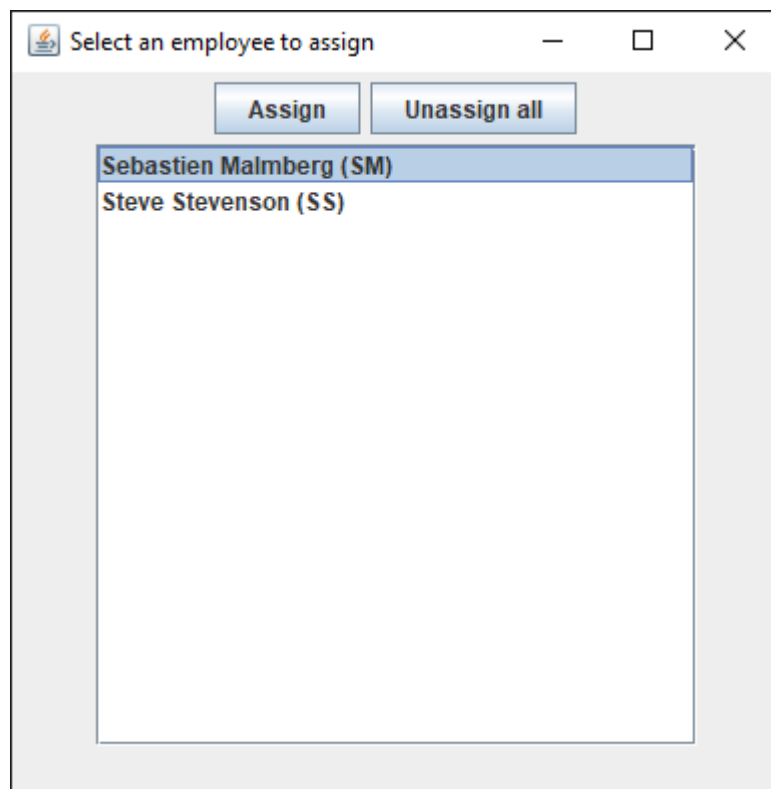


Figure 11 - Assign employee tab

Figure X illustrates the window which appears when the “No. Persons” button is pressed inside the Assign employee to schedule window. The GUI only displays employees which can conduct the desired analysis. The UI design was chosen due to its convenience. The window’s simple design raises no doubt about its function.

Implementation

The first interaction with the system is through the main panel, which has two buttons corresponding to the two use cases: Manage Schedule and Manage Employees. The two buttons generate 2 different windows that have separate file adapters: MasterFileAdapter and ScheduleFileAdapter. The first adapter serves as the database for the Manage Employees window and the second one serves as the adapter for the Manage Schedule Window.

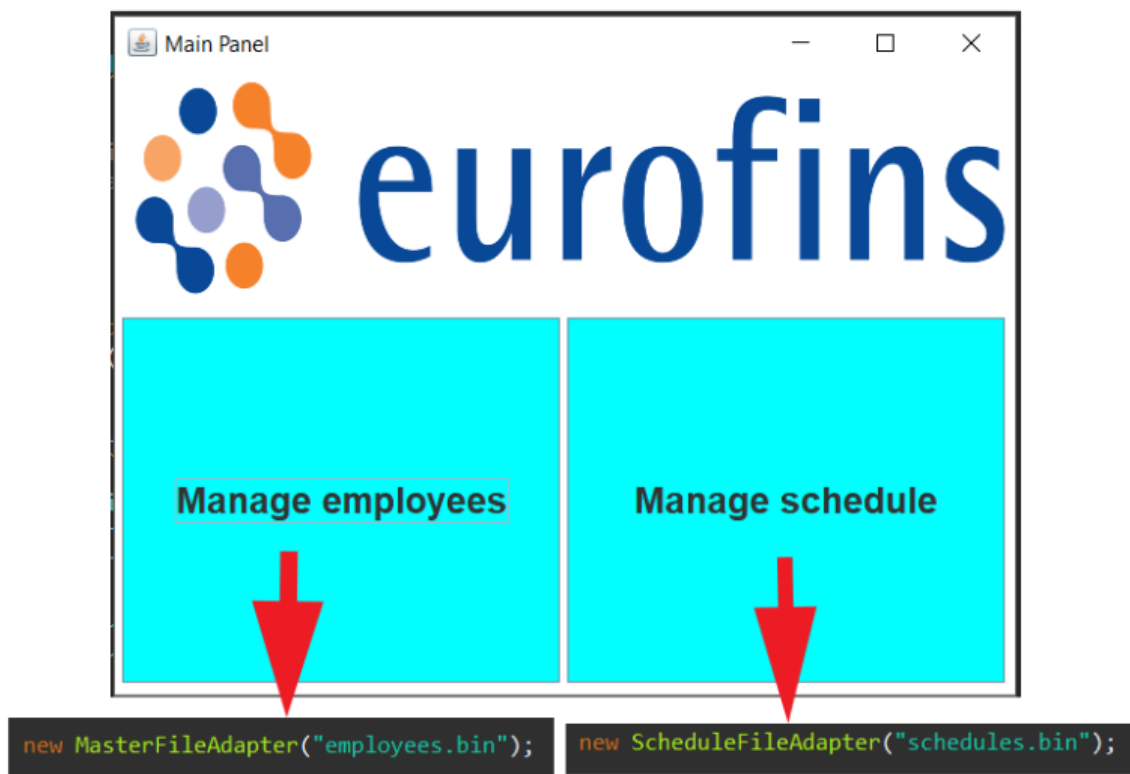


Figure 12 - Main panel

The two adapters use two standard Java libraries called MyFileIO and MyTextFileIO in order to perform operations on binary files. As the name suggests the first adapter saves all the employees in a big list, and the schedule adapter saves all the schedules in a separate file.

Employees are saved inside a single "EmployeeList" object and the "Employee" class holds various data about an employee like the name and the analysis the qualifications that person holds. The "EmployeeList" allows for the visualisation of all the employees and the "Employee" class allows for operations like changing initials and phone numbers.

Schedules are saved in the form of a schedule list which holds an ArrayList of Schedule objects. For storing the assigned employees, we created a Task class which holds an analysis and the employees assigned to that analysis. Following this the system creates a "WorkDay" class which holds a TaskList and inside the Schedule class we created six 'WorkDay' fields which correspond to the six working days of the week; Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday.

```
private WorkDay Monday;  
private WorkDay Tuesday;  
private WorkDay Wednesday;  
private WorkDay Thursday;  
private WorkDay Friday;  
private WorkDay Saturday;  
private String type;  
private String comment;
```

Figure 13 - Schedule class fields

The schedule also holds a type and a comment and corresponding to the type we create either “L” or “S” weeks trying to mimic the old way in which Eurofins managed their schedules.

```
if (type.equals("L")) {  
    Task m1 = new Task(ana.get("Fat"), 2);  
    Task m2 = new Task(ana.get("Protein"), 2);  
    Task m3 = new Task(ana.get("Fiber"), 1);  
    Task m4 = new Task(ana.get("Sugar"), 1);  
  
    Monday.add(m1);  
    Monday.add(m2);  
    Monday.add(m3);  
    Monday.add(m4);  
  
    Task tu1 = new Task(ana.get("Fat"), 2);  
    Task tu2 = new Task(ana.get("Protein"), 2);  
    Task tu3 = new Task(ana.get("Fiber"), 1);  
    Task tu4 = new Task(ana.get("Sugar"), 1);  
  
    Tuesday.add(tu1);  
    Tuesday.add(tu2);  
    Tuesday.add(tu3);  
    Tuesday.add(tu4);  
}
```

Figure 14 – Schedule constructor if-statements for types 'L' and 'S'

Inside the Schedule class the system has a couple of methods for working with the schedule but the main operation that the schedule is required to do is to assign employees to work days. The system does this by getting the desired day using “getMonday” for example and then using the “AssignEmployee” method which resides inside the Task class.

```
public void AssignEmployee(Employee var) {  
    if(reqEmployee != 0 ) {  
        list.addEmployee(var);  
        reqEmployee--;}  
}
```

Figure 15 - Assign employee method

In order to talk about the visualisation of the Schedule we must talk about the “WorkScheduleWindow” which has some interesting code choices. For instance, the system has buttons for displaying the employees assigned to the schedules and by pressing the buttons the team leader can select from a list of employees qualified in that analysis. This is all done inside a table like grid which is defined inside the “WorkScheduleTable”. Here the system has the functionality to add rows and buttons corresponding to the number of analysis and employees we must display. And every button has a listener added to it which operates on the methods inside the Schedule class.

```

JButton b1 = new JButton(analysis);
JButton b2 = new JButton(monday);
JButton b3 = new JButton(tuesday);
JButton b4 = new JButton(wednesday);
JButton b5 = new JButton(thursday);
JButton b6 = new JButton(friday);
JButton b7 = new JButton(saturday);

b2.addActionListener(new Listener(analysis,index,1));
b3.addActionListener(new Listener(analysis,index,2));
b4.addActionListener(new Listener(analysis,index,3));
b5.addActionListener(new Listener(analysis,index,4));
b6.addActionListener(new Listener(analysis,index,5));
b7.addActionListener(new Listener(analysis,index,6));

```

Figure 16 - WorkScheduleTable rows 147-161

```

private class Listener implements ActionListener {

    String analysis;
    int index;
    int day;

    /**
     * The main constructor of the listener
     * @param analysis The analysis of the employees
     * @param day The day to which the employees need to be assigned
     * @param index The Schedule index inside the ScheduleList
     */
    public Listener(String analysis, int day, int index) {
        this.analysis = analysis;
        this.index = index;
        this.day = day;
    }

    public void actionPerformed(ActionEvent e) {

        new SelectEmployeeWindow(index,analysis,day);

    }
}

```

Figure 17 - Action listener of the buttons inside the table

The schedules are created inside the “Create schedule” tab of the manage schedule window, and it can add schedules to the schedule list following the old template that Eurofins used but with some extra functionality like adding extra analysis than the default one by adding new rows. The “Add new row” button calls on a series of methods in order to add new tasks to every work day and it accepts an analysis and a type as ‘L’, ‘S’ or “all” in order to fill the required employee field of every Task. For example, if the employee would be to choose “Cattle” and “all” a new row would be added to the table with the analysis “Cattle” and with one required employee for all the tasks.

```

if(template.getAllNewAnalysis().size()>0) {
    String name = topField.getText();
    topField.setText("");
    String comment = field2.getText();
    field2.setText("");
    String date = field1.getText();
    field1.setText("");
    String type = "";

    if(L.isSelected())
        type = "L";
    else
        type = "S";

    Schedule aux = new Schedule(name,date,type,comment);
    for(int i = 0; i < template.getAllNewAnalysis().size();i++) {
        int req = 0;

        if(template.getAllNewWeekTypes().get(i).getSelectedItem().toString().equals("L")) {
            req = 2;
        }
        else if(template.getAllNewWeekTypes().get(i).getSelectedItem().toString().equals("S")) {
            req = 3;
        }
        else
        {
            req = 1;
        }
        aux.addTaskMonday(template.getAllNewAnalysis().get(i).getSelectedItem().toString(),req);
        aux.addTaskTuesday(template.getAllNewAnalysis().get(i).getSelectedItem().toString(),req);
        aux.addTaskWednesday(template.getAllNewAnalysis().get(i).getSelectedItem().toString(),req);
        aux.addTaskThursday(template.getAllNewAnalysis().get(i).getSelectedItem().toString(),req);
        aux.addTaskFriday(template.getAllNewAnalysis().get(i).getSelectedItem().toString(),req);
        aux.addTaskSaturday(template.getAllNewAnalysis().get(i).getSelectedItem().toString(),req);
    }

    list.add(aux);
}

```

Figure 18 – Code showcasing how a schedule is created when new rows are added

Test

System was tested based on the systems use-case diagrams branch sequences. Steps were followed while the program was running and as the results are shown everything is working efficiently. However, it does not mean that the program doesn't need any upgrades or fixes, it only means that the programs main functionalities are working properly.

Table 1 - Testing criteria and implementation status

Tested Criteria	Implemented / not implemented
The system must allow the creation of a new schedule	Implemented
The system must allow multiple schedules to be created and stored.	Implemented
The system must indicate whether an Employee is available, on vacation, sick, or busy.	Implemented
The system must allow Team leader to add, remove, or hide employees.	Not implemented
The system must allow a Team leader to assign available employees to any task.	Implemented
The system must allow multiple employees to be assigned to the same task.	Implemented
The system must be able to store comments and analysis.	Implemented
The system must allow team leader to change, add, and remove comments or analysis.	Not implemented
The system must allow a user to navigate between the different schedules.	Implemented
The system must store outdated schedules for minimum 1-year, current schedules and future schedules.	Implemented
The system should inform the user on what type of schedule is being viewed and the corresponding dates.	Not implemented
The system should allow the visualisation of different colours to simplify the interface.	Not implemented

Table 2 - Manage employee testing

Manage Employee	
Add an employee	Works
Remove employee	Works
Change employee data	Works
View Employee list	Works

Table 3 - Manage schedule testing

Manage Schedule	
Create schedule	Works
Remove schedule	Works
View schedule	Works
Assign employee	Works
Unassign employee	Works

Results and Discussion

The results and discussion is heavily focused on how the program manages to implement or not implement the list of requirements due to its importance for the project.

Results

Functional requirements

The list below shows what functional requirements were implemented in the program.

1. The system must allow the creation of a new schedule.
2. The system must allow multiple schedules to be created and stored.
3. The system must store and allow the visualization of all Employees.
4. The system must allow a Team leader to assign available employees to any task.
5. The system must allow multiple employees to be assigned to the same task.
6. The system must be able to store comments and analysis.
7. The system must allow a user to navigate between the different schedules.
8. The system must store outdated schedules for minimum 1-year, current schedules and future schedules.
9. The system must allow the user to view outdated schedules, current schedules, and future schedules.
10. The system should inform the user on what type of schedule is being viewed and the corresponding dates.
11. The system should allow the visualization of different colours to simplify the interface.

The list below shows what functional requirements were not implemented in the program.

1. The system must store and allow the visualisation of the information corresponding to Eurofins old schedules; workplan, training schedule, staff time overview, and preferences from annual performance review.
2. The system must allow the Team leader to add and remove/hide employees.
 - a. Hide employee was never implemented.
3. The system must allow the Team leader to change, add and remove comments or analysis.
 - a. Change analysis was never implemented.
4. The system should inform the user on what type of schedule is being viewed and the corresponding dates.
 - a. The system does not inform the user of the corresponding dates.
5. The system should allow the visualisation of different colours to simplify the interface.
 - a. Different colours were never added to the interface.

Non-functional requirements

The list below shows what non-functional requirements were implemented.

1. System must be a single-user system Java application.
2. System must be manageable with keyboard and mouse.

The list below shows what non-functional requirements were not implemented.

1. All four instances/schedules must be implemented in a single graphical user interface (GUI).

Discussion

The program is functional, and the key functions of the system has been implemented. A user is now able to create a schedule and assign appropriate employees to a specific analysis on a specific day. However, certain functions which would have contributed to the purpose of this project see appendix A definition of purpose, were never implemented.

The team leader is unable to change an employee's analysis. The "Change Analysis" feature was never completed. Currently, if the user wishes to change an employee's analysis, the employee must be deleted and added once more with the correct analysis attached, which will prove to be quiet time consuming. Therefore, although a "change analysis" feature was never specifically requested by Eurofins, it is one of the elements this program is missing. Such a feature would have made the program more flexible and less time consuming to use in the event of an employee having to change analysis.

In addition, confirmation windows were never implemented. Before an action is terminated the system should ask the user to confirm the action e.g. when deleting a schedule or an employee. Such confirmation windows could prevent the user from making mistakes. Which is why such a feature would increase the user friendliness of the program and make it less prone to mistakes

Furthermore, error-windows and notification-windows were never implemented in the system. Such informative window pop-ups could improve the system's overall effectiveness. A well-informed user will most likely not make the same mistake twice, thus saving time. Therefore, such informative window pop-ups are extensively mentioned in this project's use cases.

Moreover, multiple flaws were detected within the view schedule panel and create schedule panel. Currently, if an employee creates a schedule without giving it a name, the nameless schedule will still get created and added to the list and appear invisible. Such a flaw could have been avoided by implementing a notification window or an if else statement in which the empty text field for create schedule is replaced with a question mark. In addition, the "Add row" button inside the create schedule panel only creates rows which indicated that two employees are needed for each analysis which will not always be true.

Conclusion

Considering all the things that have been stated in the earlier parts of this paper here is the conclusion which summarizes every key aspect of our project.

Eurofins asked for a new work planning tool. The created system, that was coded in Java, aims to be the solution to this request. However, since the system fails to uphold some established requirements it fails as a complete solution. Although the system is functional and deemed ready for small scale use, and the crucial requirements have been implemented, the lack of important features such as “Change analysis”, “hide employees” and “Add analysis” sets a low limit to the system’s functionality, therefore, in conclusion its efficiency is decreased.

Furthermore, the system fails to implement exception sequences from the project use cases. Confirmation windows, error windows and notification windows were never added to the system. Therefore, it is concluded that the system is prone to user-mistakes, e.g. miss clicks. Since each click made by the user is final.

The code behind the GUI aims to fulfil the minimum requirements and it contains methods that might become obsolete in the future as they are not adaptable. For example, the system can hold data about Monday, Tuesday, Wednesday, Thursday, Friday, Saturday but not Sunday since the Task class has methods only for six days.

In the future the system could be updated, and the remaining requirements implemented which would greatly increase its functionality. Moreover, the system should be made a multi-user system since 60 employees needs access to the program. The GUI should be reconsidered since it contains a lot of redundant elements on top of having useful features missing. So, in short, the system would be greatly improved by redoing some parts of the code, revamping the GUI and then adding multi user functionality and portability to phones and websites.

The parts of the system that have been implemented are functional with some minor bugs like elements being pushed when there isn’t enough space on the screen. The database is local and does not use the SQL language but simple binary files. The program is lightweight and does not consume a lot of resources like disk space and RAM due to its low number of methods being active. The system has some leftover classes that are being used and that are exactly the same except that fact that they have different names, however this does not hinder the program in any way, but it will be pretty hard to maintain and, in the future, the system would benefit from having all the small classes merged in bigger ones.

So, to summarize, although with a few issues, a working system has been implemented so that it fulfils the requirements expressed by Eurofins Stein Laboratory. The program is ready for small scale use and it is ready to be upgraded soon.

Project future

The system covers the main requirements, but it does not have extra functionality. Also, the user experience is not streamlined at all since the user must press several buttons in order to have its schedule updated. Also, the system does not ask the user for confirmation and does not show any error messages. The windows are a bit responsive but sometimes the items are squeezed if too many elements need to be displayed. Elements are not arranged symmetrical inside the window and there are a few buttons that do not have any functionality.

The system could be therefore improved by clearing the GUI of unnecessary/redundant elements or by adding functionality to them. On top of that adding quality of life features, like the lists updating themselves without having to press a button would improve the user experience. Also, the schedules cannot be edited once created and the employees cannot have their analyses changed. For the system to be improved there needs to be a functionality for changing the template when creating a schedule and for editing employees and schedules that are already added to the database.

Besides all the listed improvements, another version of the system would benefit from having all the windows inside one single GUI instead of having all sorts of windows spawning from buttons.

Sources of Information

Viuf, A., 2018. Presentation of work planning tool. [presentation] (Public communication, 7 September 2018).

J. Bisbal, 1997. An Overview of Legacy Information System Migration. [pdf] Hong Kong. Available at: <<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=640219&isnumber=13814>> [Accessed 20 September 2018]

Appendices

Appendix A – Project Description.

Appendix B – Activity Diagrams.

Appendix C – Full Design Class Diagram

Appendix D – User Guide.

Appendix E – Source Code.

Appendix F – Website.

Appendix J – JDoc