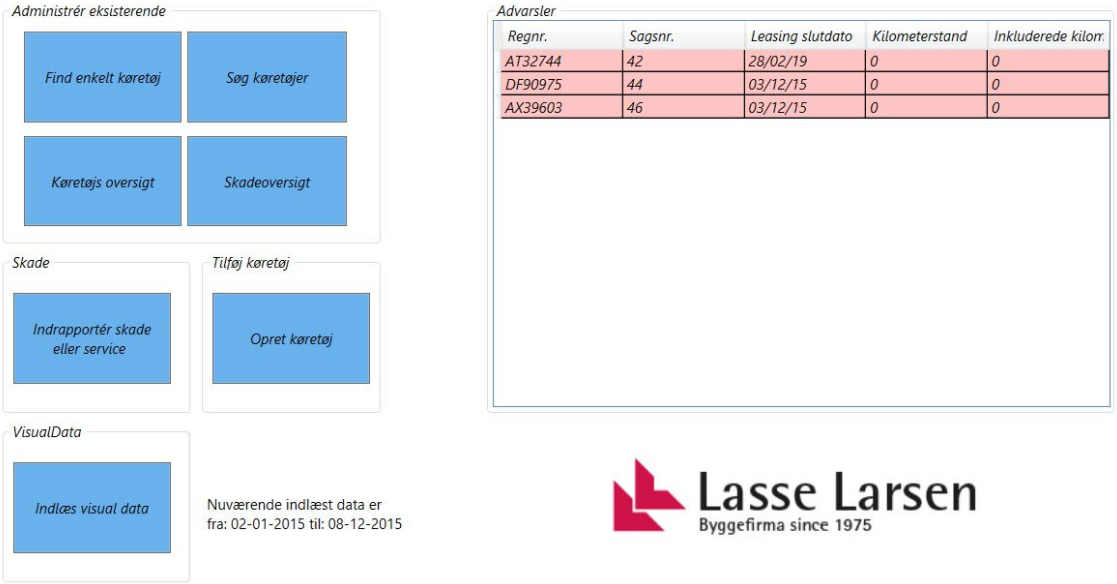
C.A.R.S.

Company Arrangeable Registry Software



P3 Project

Group ds314e15

Software

Aalborg University

December 21st, 2015

AALBORG UNIVERSITET
STUDENTERRAPPORT

Department of Computer Science
Software
Selma Lagerløfs Vej 300
9220 Aalborg Ø
http://www.tnb.aau.dk

Title:

C.A.R.S.

Project:

P3-project

Project period:

September 2015 - December 2015

Project group:

ds314e15

Participants:

Emil Bejstrup
Martin Folmer
Patrick Lynnerup Rønn Andersen
Søren Holmer Pedersen
Tanja Lind Hansen

Supervisor:

Srinivasa Raghavendra Bhuvan Gummidi

Abstract:

The purpose of this paper is to develop a stand-alone software system to handle administration of company vehicles at Lasse Larsen Byggefirma A/S, because the fleet administration today is not fulfilling the needs of the company. To examine this problem, principles from the *System Development* and the *Design and Evaluation of User Interfaces* courses has been applied. An analysis document has been created with focus on the problem domain to understand the problem together with the application domain to understand who is going to use the system. User evaluations has been conducted to test the system. The results found through this show how important human-centred design is.

Editions: 7
Number of pages: 105
Appendix: 5 + CD
Date of completion 21-12-2015

Preface

This report is written by five software engineering students during their 3^{rd} semester at Aalborg University. The theme of the semester project is *Developing applications - from users to data, algorithms and tests - and back again*, and this report focuses on a system for a type house company, which will help in managing their vehicles in different ways.

In the analysis of the current situation we would like to thank Lasse Larsen Byggefirma A/S, who has provided knowledge of how the management of the company vehicles is operated today. Specifically we would like to thank Dorte Sørensen for information and data and for being part of tests during the period.

Furthermore our appreciation also goes out to the tests subjects for **C.A.R.S.** and their feedback towards the system and our cooperation.

Last but not least we would like to thank our supervisor, Srinivasa Raghavendra Bhuvan Gummidi, for good cooperation. This includes the fresh eyes and ideas towards the project and templates for new parts, so that we had some idea of what to include in the report.

Thank you.

Emil Bejstrup Otterstrøm | Martin Folmer | Patrick Lynnerup Rønn Andersen

Søren Holmer Pedersen | Tanja Lind Hansen

Reading guide

This report is mainly divided into two parts: Development, the first part, describes the development thoughts and design of the system **C.A.R.S.** through an analysis of the problem domain and the application domain. The second part, Academic, describes development methods used in accordance with courses followed parallel to the project.

Tables and figures are numbered by the corresponding chapter. The first figure in chapter 2 will then be named 2.1, the second 2.2 and so on. Tables and figures are made by authors, unless stated otherwise by source indication in the text of the figure.

The references of the report are listed by the Vancouver method, where a reference in the report is represented by a number, e.g. [1]. Reference [1] will then be available with information of origin in the bibliography list at the end of the report.

Listing 1, shows how code listings are illustrated.

```
1 //Method for updating milage, check that new value is equal or
     higher then the old value
2 public bool YearlyOdometerReading(Vehicle vehicle, double read) {
3         if (vehicle.Milage>read) {
4                 return false;
5         }
6         vehicle.Milage = read;
7                 return true;
8         }
```

Listing 1: Code example

The full content of the CD includes a **C.A.R.S.** manual, a pdf document with the source code and a .zip file with the code as well.

The following report is a documentation of results during the work in this project.

Contents

Part I
Introduction to development

In this part of the report there will be a short introduction to the underlying problem, then the assignment chapter which introduces the reasoning behind working with an application for Lasse Larsen Byggefirma A/S company vehicles. This will be followed by the problem domain and application domain, then the system design, implementation of the system followed by the tests, a discussion, conclusion and future work.

Introduction 1

Transportation is an important part of any infrastructure in the world. Everyday transportation is a key necessity for many people around the globe. Due to the major importance of transportation, a well-developed and functional transportation system and network is of great advantages. The transportation aspect is crucial

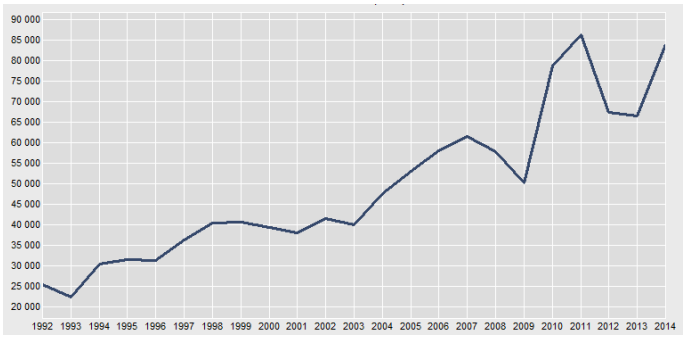


Figure 1.1: Development in company cars in denmark from 1992 to 2014

for the survival and competitiveness of many companies. The amount of company vehicles in Denmark is increasing almost every year, and has been increasing the last two decades.

In 2014, 188.416 new registrations of cars were made, 84.000 of these were cars meant for company use[1]. The previous year, 2013, the number of cars registered for company use was 66.668 and this development will only continue in the years to come[1]. The increase of company vehicles naturally adds additional workload to the administrative unit of every company, more vehicles means more administrative work. With more administrative work, comes an additional cost related to this. Administrative tasks regarding company vehicles can be numerous and the workload associated with these vehicles can quickly become quite severe. At the same time the amount of data the companies needs to store concerning their vehicles are significant, everything from simple information about each vehicle to the inspection reports has to be stored somehow. Therefore, administration is a logical choice to look for possible IT solutions and implementations to assist and help overcome the growth in work.

Assignment 2

This chapter introduces the reason for working with a system for company vehicles. Furthermore, the system definition will clarify how and what the system is supposed to do. The final section presents a rich picture for the reader to understand the scope of the problem through the eyes of the authors.

2.1 Purpose

Lasse Larsen Byggefirma A/S is a Danish type-house company with branches in Jutland and on Zealand. They construct around 100 houses each year. The Company has around 50 employees and 30 vehicles in total, where approximately a third of the vehicles are leased[2]. Currently the company handles information regarding company vehicles at several different locations. Some information is held in an excel sheet, some in an external economic software system, some in paper folders and some is currently not registered, such as information regarding damages. The company wants to consolidate the information regarding their vehicles in an IT system that supports the employees in their tasks regarding the administration of their company vehicles. It should keep track of master data and information regarding damages and present the users with different readouts that are combinations of the data. This will help the employees organize the data and to make decisions as to whether they should keep maintaining a vehicle or let it retire.

Not included

The system will not have information regarding:

- Fuel consumption
- Fuel efficiency
- Route planning
- Route optimizing

This information is not included due to the scope of the system. The system will only hold the master data to organize, and damage reports to get an overview, not information regarding optimization of driving.

2.2 System definition

A stand-alone IT-system for gathering, sorting and presenting data regarding vehicles at the type-house company, Lasse Larsen Byggefirma A/S. The system will import information about costs from an external economic system, E.G. Visual, and save the master data of vehicles in a local database. Furthermore, the system will provide an option to report damages regarding company vehicles. The system will present the user with a selection of pre-set sorting- and grouping options, and can output an excel or pdf file containing the sorted/grouped data. The system will be used by employees with either office education or similar experience and computer skills. The system must be running on an office PC that supports .NET framework, spreadsheets and pdf viewer.

FACTOR

Next the system definition is set up by the FACTOR principles[3] for the reader to get a structured overview of the system definition. No new information is added.

Functionality	System for gathering, sorting and presenting data concerning use and cost of company vehicles.
Application domain	Provides overview of information to help make decisions regarding the continued upkeep of company vehicles at Lasse Larsen Byggefirma A/S.
Conditions	Users with either office education or similar experience and computer skills.
Technology	The system will be running on a computer that supports .NET framework, spreadsheets and pdf viewer.
Objects	Vehicles, economic data, master data, damages and output file.
Realization	System for gathering, sorting and presenting data to organize and assist in making decisions regarding continued maintenance or replacement of company vehicles.

2.3 Surroundings

To present the developer's view of the situation, an informal picture has been drawn - also known as a rich picture. The rich picture focuses on relations chosen by the artist. The importance of a rich picture is, that it can lead to several different solutions and interpretations of the situation.

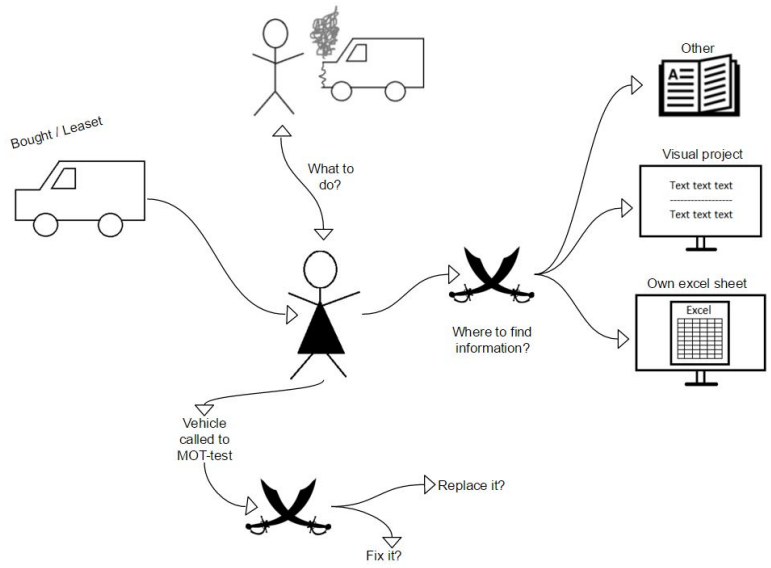


Figure 2.1: A rich picture of the situation at Lasse Larsen Byggefirma A/S

Figure 2.1 shows different conflicts at hand, in the process of handling vehicle information at Lasse Larsen Byggefirma A/S, all with the head of administration as the center. The crossed swords at the bottom of the the picture indicates a conflict, where the head of administration currently has no indications whether a vehicle is worth repairing or not. The other crossed swords to the right of the woman figure leads on to three different places in where the company currently keeps vehicle information. Therefore the conflict lies in where to find and input information about vehicles and damages.

Problem domain

The problem domain is defined as *the part of the surroundings, which is managed, monitored or controlled by a system*[3]. Figure 2.1 shows the problem domain as the information regarding company vehicles and conflicts in where to find and input this information for future reference when a vehicle must be assessed for repairs or replacements. Furthermore Lasse Larsen Byggefirma A/S currently have nowhere to register damages regarding their vehicles.

Application domain

The application domain is defined as *an organization which manage, monitor or control the problem domain*[3]. At Lasse Larsen Byggefirma A/S Dorte Sørensen is the head of administration, who manages all information about vehicles and other company resources and properties. Therefore she will be the user of the system.

2.4 Role model

The primary user of the system is, as mentioned, the head of administration at Lasse Larsen Byggefirma A/S. Currently she is doing most of her work regarding company vehicles in Excel sheets. The head of administration has expressed her fondness of the set-up and functionality in this type of software[2], based on the simplicity made by the columns and rows. Therefore Excel will be a role model for setting up information in the **C.A.R.S.** system to be developed. Furthermore, she wants the functionality in the system, that the system is able to export chosen information to excel.

Excel

Excel is a software system developed by Microsoft. It can be used for storing, organizing and manipulating data. The general use of Excel includes cell-based calculations, pivot tables and graphing tools. Excel can be used to create all sorts of budgets, and organize large amounts of data. Excel is built around columns and rows, which are made up of individual cells. These cells can be manipulated and modified as needed by changing color, fonts, layout etc.

Problem domain 3

In this chapter the problem domain will be elaborated by the structural correlation between classes. Additionally, the events in which the objects may be present will be presented. All this will end with an event table to provide an overview.

The definition of a problem domain is as follows: *the part of the surroundings, which is managed, monitored or controlled by a system*[3]. To solve a problem the problem domain must be examined by topics of interest. As explained in section 2.1, some information regarding vehicles are not an issue of the system to be developed. The analysis will solely regard relevant information within the delimited area.

3.1 Structures

The structure of the problem domain is represented with the help of the class diagram in figure 3.1.

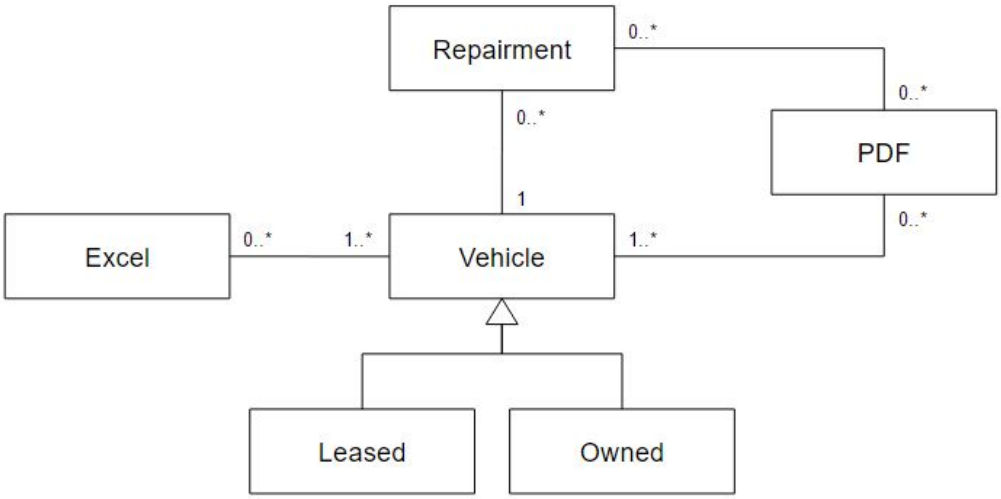


Figure 3.1: A simple class diagram of the problem domain at Lasse Larsen Byggefirma A/S

There are different relations between the classes as shown in figure 3.1. The straight line between Vehicle and Damage states that a vehicle is associated with zero to many damages. The triangle beneath Vehicle indicates that a vehicle *is a* Leased or Owned vehicle.

3.2 Classes

A class is a representation of a collection of objects having the same characteristic properties and exhibit the same common behaviour. An object is a real-world element with identity, state and behaviour. A class is a description of several objects, that has the same structure, behavioural patterns and properties. The classes shown in figure 3.1 will in this section be elaborated with conditional diagrams and related explanations.

Vehicle

Vehicle is the superclass of Owned- and Leased vehicle. A superclass has properties that the subclasses inherit. This indicates that the data and functionality held in Vehicle will be inherited by Leased and Owned and these subclasses will then expand and specialize with additional functionality.

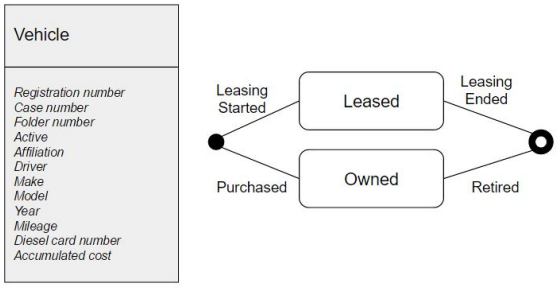


Figure 3.2: Conditional diagram for "Vehicle" superclass

The properties shown in the grey box in figure 3.2 are the master data that the two subclasses have in common. The conditional diagram to the right states that a vehicle can either be a leased or an owned vehicle. These will have further descriptions in the following two sections.

Leased Vehicle

Leased Vehicle includes five properties as shown in the grey box in figure 3.3. This class contains master data for a leased vehicle.

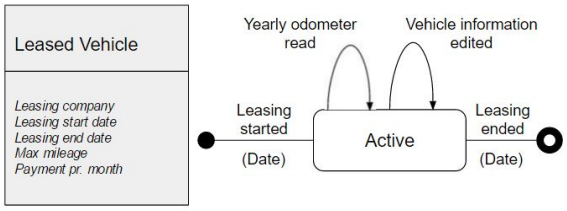


Figure 3.3: Conditional diagram for "Leased" subclass

The conditional diagram in figure 3.3 shows that an object of the Leased Vehicle class exists with the start event *Leasing started* and while it is active the event *Vehicle information edited* can be triggered as many times as needed. It terminates with the finish event *Leasing ended*.

Owned Vehicle

Owned Vehicle has five properties as shown in figure 3.4, this class contains specialized master data regarding the vehicles owned by Lasse Larsen Byggefirma A/S.

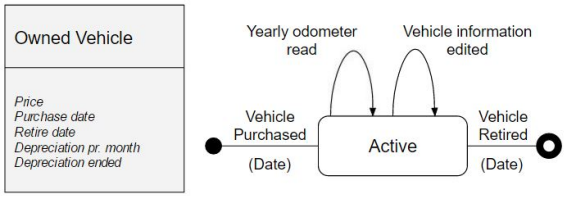


Figure 3.4: Conditional diagram for "Owned" subclass

The conditional diagram in figure 3.4 shows that an object of the owned vehicle class exists with the start event *Vehicle Purchased* and while it is active the event *Vehicle information edited* can be triggered as many times as needed. The vehicle continues to be active until the termination event *Vehicle Retired* happens.

Repairment

The class Repairment has four properties as shown in the grey box in figure 3.5. The class does not include the cost of the damage, because that is not a concern for the damage report wanted by Lasse Larsen Byggefirma A/S. The class holds information describing the nature of damage to the vehicle that requires repair.

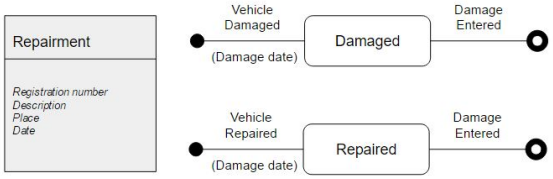


Figure 3.5: Conditional diagram for "Repair" class

The conditional diagram in figure 3.5 show that an object the class Repairment exists when a vehicle is damaged or has been repaired. The reasoning behind there being two diagrams, is that the user might enter the information regarding the damage at either point in time, depending on the circumstances. Both conditional diagrams terminates when the respective information has been entered.

Pdf

The class Pdf has two lists as shown in the grey box in figure 3.6. The lists are provided by respectively the Vehicle and the Repairment classes. Pdf will export information regarding vehicles and related damages into a pdf document.

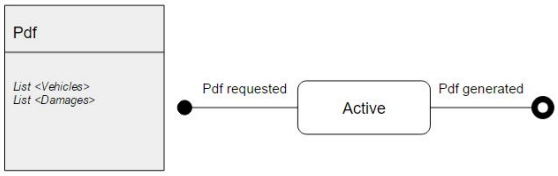


Figure 3.6: Conditional diagram for "Pdf" class

The conditional diagram in figure 3.6 shows that an object the class Pdf exists when a pdf has been requested. The termination event is *Pdf generated*.

Excel

The class Excel has only a list of vehicles provided by the Vehicle class. The class will export information regarding vehicles into an excel sheet.

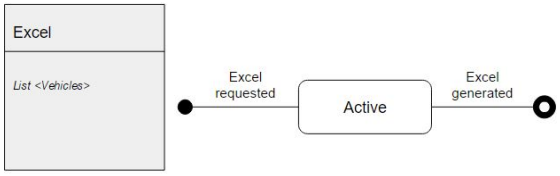


Figure 3.7: Conditional diagram for "Excel" class

The conditional diagram in figure 3.7 shows that an object of the class Excel exists between the start event *Excel requested* and the terminating event *Excel generated*.

3.3 Events

This section will provide a description of all the events presented in the explanations for the conditional diagrams of the classes. Finally, the event table will be presented to get an overview of the interactions between the events and classes.

Damage entered

The *Damage entered* indicates that the information regarding a damage to a vehicle at Lasse Larsen Byggefirma A/S has been entered.

Vehicle information edited

The *Vehicle information edited* indicates that some correction or addition to the master data of the vehicle has been made.

Vehicle purchased

The *Vehicle purchased* indicates that a new vehicle has been purchased and is added to the vehicle master data list.

Vehicle retired

The *Vehicle retired* indicates that an owned vehicle is in a state where the continued maintenance cost exceeds the cost of replacing it and the vehicle will be retired.

Leased started

The *Leased started* indicates that a new vehicle has been leased and is added to the vehicle master data list.

Leased ended

The *Leased ended* indicates that a leasing contract has ended, either by reaching its agreed end date, or by the vehicle having reached its allotted number of kilometres.

Odometer read

The *Odometer read* indicates the yearly event of reading the odometer and reporting back to the office where the head of administration updates the master data has occurred.

Pdf requested

The *Pdf requested* indicates that a report in pdf format regarding some vehicle information is needed.

Pdf generated

The *Pdf generated* indicates that a requested report in pdf format has been generated.

Excel requested

The *Excel requested* indicates that a presentation of a vehicle list is requested to be exported into an Ecxel file.

Excel generated

The *Excel generated* indicates that a requested export of a presentation has been generated into an Excel file.

Vehicle damaged

The *Vehicle damaged* indicates that a vehicle has been damaged and is ready

to by registred in the system.

Vehicle repaired

Vehicle repaired indicates that a vehicle has been repaired and is ready to by registred in the system.

Overview

An event table has been created based on the above events and conditional diagrams, to provide a better overview of the interactions between the events and classes.

Events \ Classes	Repairment	*Vehicle*	Leased vehicle	Owned vehicle	Pdf	Excel
Vehicle repaired	*					
Vehicle damaged	*					
Vehicle information edited		*	*	*		
Damage entered	*	*	*	*		
Vehicle purchased				+		
Vehicle retired				+		
Leasing started			+			
Leasing ended			+			
Odometer read		*	*	*		
Pdf requested	*	*	*	*	+	
Pdf generated					+	
Excel requested		*	*	*		+
Excel generated						+

Table 3.1: Event table

There are two different signs used in the event table: * and +. The + indicates that the event will only occur one or zero times. The * indicates that the event will be able to occur zero to many times.

Application domain 4

This chapter will elaborate on the users and the use of the system. The analysis is based on the interview in appendix B and made in cooperation with the user. The chapter will present different representations of use cases, to show the use of the system and end with a preliminary diagram/graphical presentation of the user interface navigation.

The definition of an application domain is as follows: *An organization which administrates, monitors, or controls a problem domain*[3]. Users and any external system which will interact with the system is referred to as *actors* and their interactions as *use-cases*.

4.1 Use of the system

At Lasse Larsen Byggefirma A/S only one person handles all administration regarding their vehicles. Dorte Sørensen is the head of administration, and she will mainly be the one interacting with the system.

Overview

To form an overview of the different possible interactions the system will provide for the head of administration, two actor tables has been made: one with interactions regarding vehicles and the second with interactions regarding repairs.

	Actor
Use cases	Head of administration
Deactivate vehicle	✓
Create new vehicle	✓
Edit single vehicle	✓
Delete vehicle	✓
Show single vehicle	✓
Show grouped vehicles	✓
Show all vehicles	✓
Export to excel	✓
Export to pdf Edit odometers	✓
Load data from visual	✓

Table 4.1: An actors table regarding vehicles

	Actor
Use cases	Head of administration
Report damage	✓
Show damage overview	✓
Edit damage	✓
Delete damage	✓

Table 4.2: An actors table regarding repairs

The head of administration needs to access the master data regarding the vehicle, such as make, model etc. Furthermore, she needs to be able to group and sort information by different parameters for comparisons and she needs to be able to gather all information about e.g. costs or damage history and enter new damages into the system.

Actors

To describe the main actor of the system, an actor specification will be made. This actor specification consists of three parts: purpose, characteristics and examples which will include some of the possible interactions between the actor and the system.

Head of administration	
Purpose:	A person who administrates vehicles. This person needs to be able to do all the interactions with the system.
Characteristics:	The actor of the system is only consistent with the head of administration, who has a light IT education but is used to operate different computer programs.
Example:	A vehicle has been damaged and the head of administration needs to save data for future knowledge when deciding to repair or retire the vehicle.
	A leased vehicle has been written of, so the head of administration needs to access the accumulated cost of the vehicle and kilometres driven to decide whether to buy or send back the vehicle.

Table 4.3: An actor specification

The system wanted by Lasse Larsen Byggefirma A/S is mainly connected to one person. The limited amount of actors has its advantages. The system can be designed without conflicts by user preferences.

Use cases

A use case is an interaction between the system and the actors in the application domain. A use case can be initialized by either an actor or by a system. The total number of use cases defines the use by the application domain. These cases are studied in cooperation with the actor, for her insights and demands to be available in the system. Tables 4.1 and 4.2 shows the interactions which are needed for the head of administration to get the necessary informations. How the actor will interact with the system in order to do specific tasks will in this section be described with conditional diagrams and corresponding texts. The leading action will not be included in the diagrams. The leading action is the press of a button at the start page of the system. There are seven leading buttons: *Single vehicle, Grouped vehicles, Vehicle overview, Damage overview, Report damage, Create vehicle* and *Load data*. The corresponding text descriptions will have this information included.

Two of the leading actions, Vehicle overview and Damage overview, are general buttons which lead to pages from where most functionality regarding respectively vehicles or damages is accessible. This implies that for some use cases there will be two ways of execution, either by going through the overview pages or by using the short-cut leading actions at the start page.

Create vehicle

The *Create vehicle* use case will be presented for the actor in two ways. With the leading action *Vehicle overview*, the use case is shown in figure 4.1. Figure 4.2 shows how the interaction of creating a new car into the system will be by the leading action *Create vehicle*.

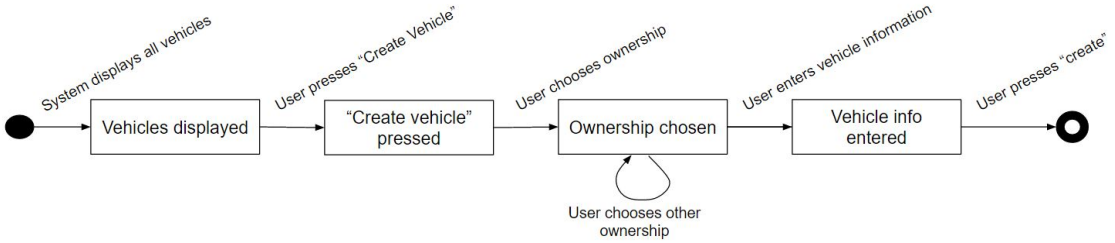


Figure 4.1: Conditional diagram for use case "Create vehicle"

In figure 4.1 the system will display all vehicles with basic information in a list. To the right of this list there will be a button assigned the creation of a vehicle. From this point the interaction will be the same as the short-cut described in figure 4.2.

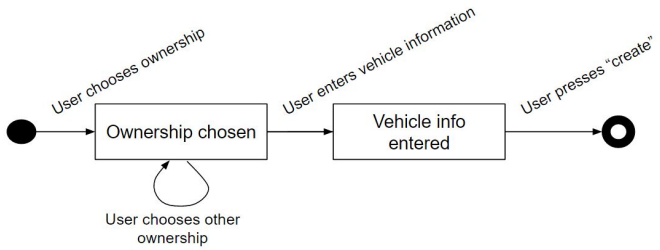


Figure 4.2: Conditional diagram for use case "Create vehicle"

The actor will firstly need to choose form of ownership before entering vehicle information. The ownership can be edited as many times as needed. When the final ownership has been chosen, the vehicle information can be entered. The vehicle will be added to the list of vehicles when the actor presses "create".

Deactivate/activate

The interaction of deactivating and the interaction of activating a vehicle are the same, therefore these two are assembled in one description. The system will provide two ways for the actor to deactivate or activate a vehicle: either by the leading action *Single vehicle* causing the actions in figure 4.3 or by *Vehicle overview* resulting in figure 4.4.

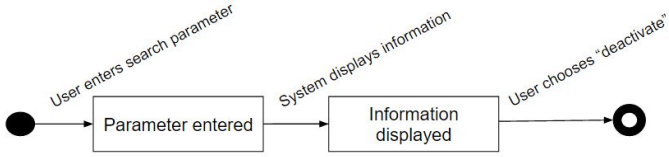


Figure 4.3: Conditional diagram for use case "Deactivate and activate vehicle"

The actor will, in this interaction shown figure 4.3, need to enter a search parameter, for the system to be able to identify the wanted vehicle. The system will then display information corresponding to the search parameter and the actor can press "deactivate" or if the vehicle is deactivated the button will say "activate".

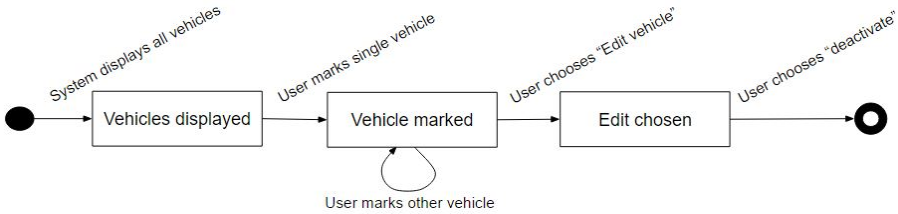


Figure 4.4: Conditional diagram for use case "Deactivate and activate vehicle"

In this interaction, figure 4.4, the actor is presented with all vehicles listed on the screen. The actor can then choose a vehicle by marking it. This marking can be changed by clicking on another vehicle, which will then be marked. When the

correct vehicle is marked, the actor can press "Edit vehicle", which will lead to the same edit page as figure 4.3 where the actor can deactivate/activate the vehicle.

Delete damage

Figure 4.5 shows the conditional diagram for the interaction between system and actor when deleting a damage. The leading action is *Damage overview*.

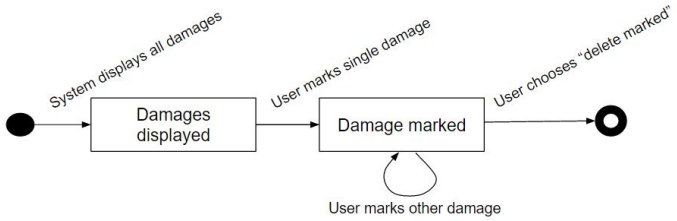


Figure 4.5: Conditional diagram for use case "Delete damage"

The system will display all damages to the actor. When clicking on a damage, the damage be marked. This selection can be changed by marking another damage. When the wanted damage is found and marked, the actor can delete the damage by pressing "delete marked".

Delete vehicle

Figure 4.6 shows the conditional diagram for the interaction between system and actor when deleting a vehicle. The leading action is *Vehicle list*.

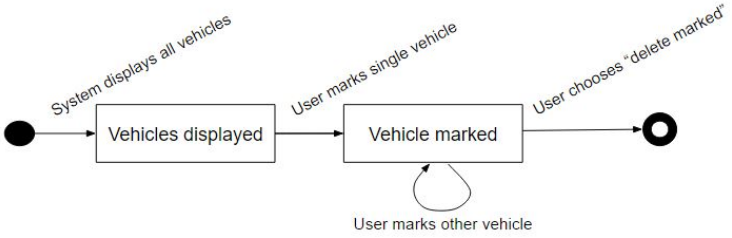


Figure 4.6: Conditional diagram for use case "Delete vehicle"

The system will display all vehicle to the actor. When clicking on a vehicle, the vehicle be marked. This selection can be changed by marking another vehicle. When the wanted vehicle is found and marked, the actor can delete the vehicle by pressing "delete marked".

Edit damage

The conditional diagram in figure 4.7 shows the interactions when the actor wants to edit or add information to a damage. This interaction has the leading action *Vehicle overview*.

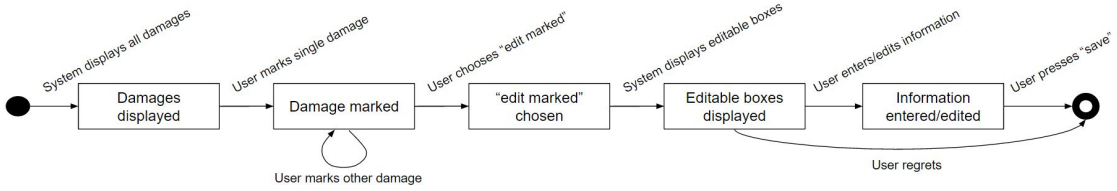


Figure 4.7: Conditional diagram for use case "Edit damage"

The system will display all damages and the actor can highlight one by clicking on it. If the wrong damage has been marked, this can be changed by marking on another. When the right damage has been marked the actor can press "edit marked" which will make the system display all information about the chosen damage in editable boxes. The actor can then edit or add information and then save changes and if not, just close the window.

Edit odometer

Figure 4.8 is the conditional diagram for the interaction where the actor changes odometer readings. This interaction will happen approximately once a year. The leading action is *Vehicle overview*.

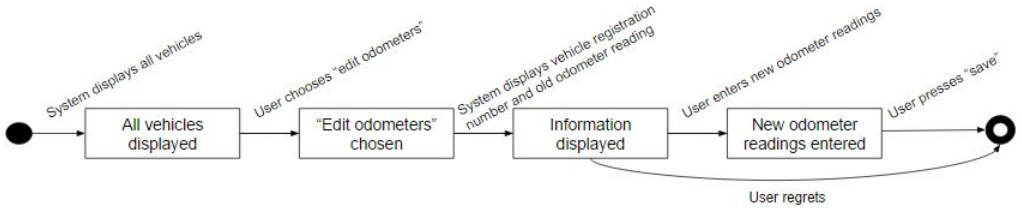


Figure 4.8: Conditional diagram for use case "Edit odometer"

The system will display all vehicles whereupon the actor can choose "edit odometers". This will make the system display all vehicles with registration number and currently registered odometer readings together with text boxes where the actor can enter new readings. When pressing "save" the old readings will be overwritten by the new ones. If the actor has made a mistake, the actor can close the window without any changes having been made.

Edit single vehicle

There are two ways for the actor to either edit or add information to a vehicle. The leading action for figure 4.9 is *Single vehicle* and the leading action for figure 4.10 is *Vehicle overview*.

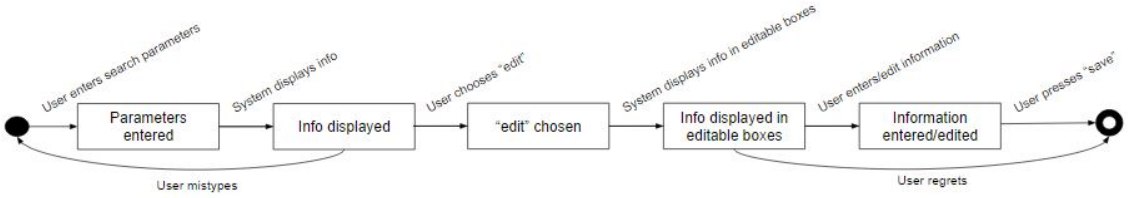


Figure 4.9: Conditional diagram for use case "Edit single vehicle"

In this interaction the actor will need to enter a search parameter for the system to identify the vehicle and the current information will be displayed. If a mistype has occurred the actor can close the window and re-enter the correct parameter. By choosing "edit" the same information will be displayed, but in editable boxes for the actor to either edit or add information about the vehicle. No new information will be saved until the actor presses "save", thereby if closing the window in earlier stages of interaction, nothing will change.

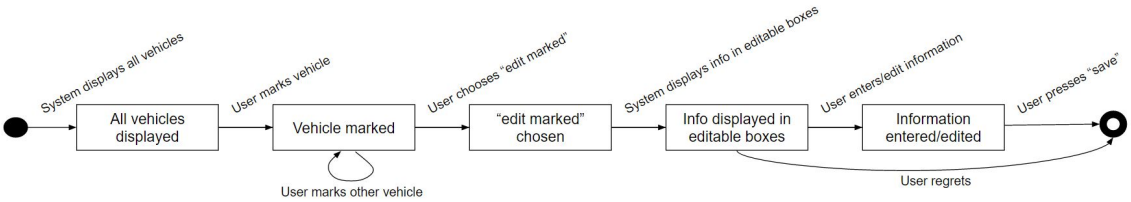


Figure 4.10: Conditional diagram for use case "Edit single vehicle"

In this interaction the system will display all vehicles and the actor can thereby mark the intended vehicle. If the wrong vehicle has been marked this can be changed by marking the correct vehicle. When the right vehicle is marked the actor can press "edit marked", which will make the system display current information about the vehicle in editable boxes like the procedure in figure 4.9. The following actions correspond to figure 4.9 as well.

Report damage

The Report damage also has two different interactions. The first, figure 4.11, has the leading action *Report damage* and the second, figure 4.12 has *Damage overview* as its leading action.

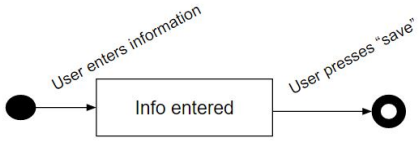


Figure 4.11: Conditional diagram for use case "Report damage"

Figure 4.11 shows one of the possible interactions when the actor wants to report a damage to the system. It is a short interaction where the actor enters information and presses "save".

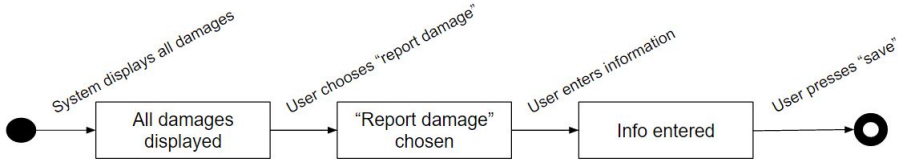


Figure 4.12: Conditional diagram for use case "Report damage"

Figure 4.12 shows another way to report a damage to the system. In this case the system will display all damages to the actor. The actor will not use any current damages, but instead press "Report damage" at the page. This will lead her to the same window whereas figure 4.11 starts its interaction and the following procedure will be the same.

Show all damages

Figure 4.13 shows how the actor will interact with the system when viewing all damages. The leading action is *Damage overview*.

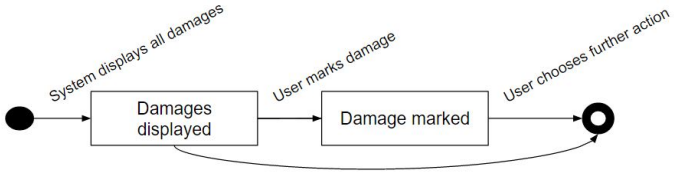


Figure 4.13: Conditional diagram for use case "Show all damages"

The system will display all damages in a list sorted by last added. From this point the actor can go and do further actions which are described in the *Export* use case. If the actor marks a damage, more functionality becomes available, which are described in respectively *Delete damage* and *Edit damage* use cases.

Show all vehicles

The use case of *Show all vehicles* is similar to the use case of *Show all damages*, as shown by comparison of figure 4.13 and 4.14.

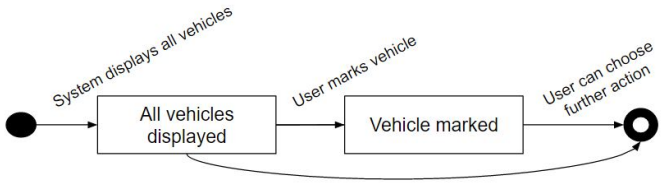


Figure 4.14: Conditional diagram for use case "Show all vehicles"

When marking a vehicle the actions, which are available, are explained in the *Delete vehicle*, *Edit vehicle* and *Deactivate vehicle* use cases.

Show grouped

Figure 4.15 shows the conditional diagram for the interaction *Show grouped*, where a specific list of vehicles is presented. The leading action is *Grouped vehicles*.

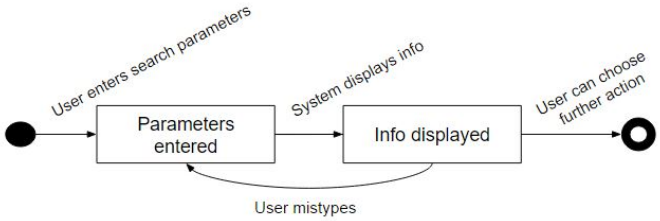


Figure 4.15: Conditional diagram for use case "Show grouped"

The actor will start by choosing which information she wants to have presented. The system will then display vehicles corresponding to all previously entered parameters. The actor can then close the window or continue as described in the use cases *Export pdf* and *Export excel*.

Show single vehicle

This use case, shown as a conditional diagram in figure 4.16, describes the interactions of presenting a single vehicle. The leading action is *Single vehicle*.

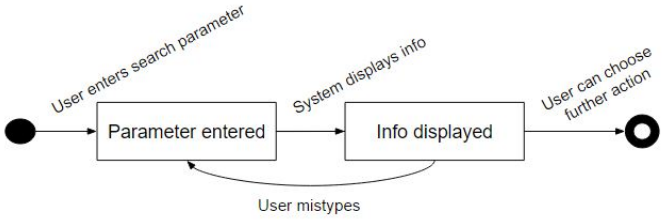


Figure 4.16: Conditional diagram for use case "Show single vehicle"

The actor will start by entering a search parameter, so that the system can identify which vehicle to display. If the wrong parameter has been entered, the actor can close the window and start over. When information is displayed, the actor can close the windows or take further actions, which are described in the use cases *Export pdf, Export excel, Edit single vehicle* or *Deactivate/activate*.

Export excel

Export excel is a general use case that is an extension of two other use cases. *Show all vehicles* and *Show grouped*. Therefore the leading action is not one of the seven as mentioned in the introduction, but instead one of the two of which this is an extension of.

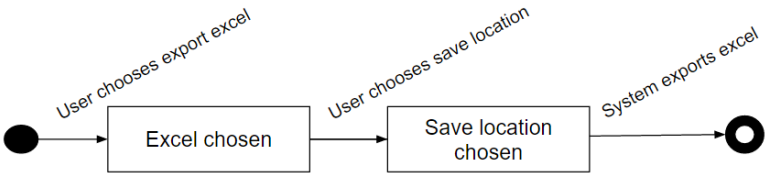


Figure 4.17: Conditional diagram for use case "Export excel"

The interaction starts with the actor choosing to export the presentation as excel. The system then prompts the actor to select a location to save the excel presentation. The actor selects a location and the system exports the presentation.

Export pdf

Export PDF is likewise a general use case that is an extension of three other use cases. *Show single*, *Show grouped* and *Show all vehicles*.

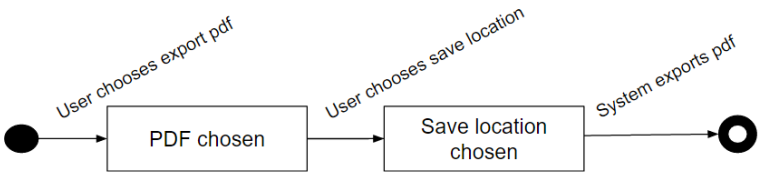


Figure 4.18: Conditional diagram for use case "Export PDF"

The interaction starts with the actor choosing to export the presentation as a PDF. Then selecting the location where the PDF should be saved.

4.2 Requirement specification

The following requirement specification is a list made in cooperation with the user, and built by different elements. First, a general description of the system will be made. Second the requirement list will be presented. The requirement specification in this case consists of functional and non-functional requirements. The functional requirements describe the actions which happens automatically, and the non-functional requirements are guidelines for the project, where no data will be influenced, such as for example graphics and limits.

General description

This system will be developed for a personal computer. The system will store current data from excel sheets into a database along with new data and it will consist of group placed tasks, concerning company vehicle data, from which the user can choose the task to perform. If the task includes addition of data, the system will update the vehicle or damage database. If the task is to present some data regarding a vehicle the it will provide export actions for the user - excel or pdf.

Requirement specification

The requirement specification is made in cooperation with the user. The specifications will be listed in points to provide an overview and serve as a check-list for developers to use. The functional requirements of the system will be listed first, followed by the non-functional requirements.

Functional requirements

Vehicle	-	Identify vehicles by registration number
	-	Place input information in most used order *(list given by the user - Appendix C)*
Damage	-	Identify damage by registration number of the vehicle involved
	-	Insert copied text from mail into damage description
	-	Place input information in most used order *(list given by the user - Appendix C)*
General	-	Information will be stored in the database
	-	Show and print graphical presentation
	-	Export data to excel
	-	Read from excel containing economic data

Non-functional requirements

Limits	-	Danish user interface incl. æ, ø, å
Design	-	Developed as a windows application
	-	High usability

4.3 Functions

A function is a *facility which makes a model usable for an actor*[3]. A function has three stages: it is activated, it is executed and it gives a result. To be able to establish the functionality of the system, a complete list of functions will be presented.

Complete list of functions

Every function is described with its name, its complexity and its type. The complexity is scaled in three stages: simple, medium and complex. The type is likewise segmented into four possible definitions: update, read, signal or calculate.

Function	**Complexity**	**Type**
- Yearly odometer reading	Simple	Update
- Find vehicle by parameters	Simple	Read
- Deactivate vehicle	Simple	Update
- Accumulate cost	Simple	Calculate
- Create owned	Simple	Update
- Create leased	Simple	Update
- Show single vehicle	Simple	Read
- Show all vehicles	Simple	Read
- Edit vehicle	Simple	Update
- Warning	Simple	Signal
- Read visual data	Complex	Read
- Export to ecxel	Complex	Read, Update
- Export to pdf	Complex	Read, Update
- Serializer	Medium	Update
- Deserializer	Medium	Read
- Add repairment	Simple	Update
- Delete repairment	Simple	Update
- Edit repaiment	Simple	Update
- Find repairment by reg.no.	Simple	Update

Table 4.4: Complete list of functions

All of the aforementioned function types are used to define functions for the **C.A.R.S.** system. To understand their relations a brief definition will be presented in the following.

Update results in a mode swift in the model, and are activated by an event in the problem domain

Read functions are activated when the actor needs information. This type of function results in a display of the requested part of the model.

Calculate functions have similarities to read functions. This type is also activated when the actor needs information, but the result is a calculation of information given by the actor and model.

Signal functions is activated when a change in the model occurs and results in a reaction to the surroundings, which can be a displaying of the changes to the actors of the application domain or a direct procedure in the problem domain.

Specification of functions

In this section the three complex functions *Read visual data, Export to excel* and *Export to pdf* will be described with relevant details.

Read visual data

This function will read in an exported excel sheet from E.G. Visual. The actor will choose the file from a destination of her preferences and the data will be loaded into the system, where different parts of the system can make use of the economic data.

Export to excel

This function will export information chosen by the actor. The information will be set up and divided into cells in columns and rows, as current excel sheets used by the head of administration at Lasse Larsen Byggefirma A/S.

Export to pdf

This function will export master data regarding vehicles into a pdf document. Along with a vehicle, the corresponding damages will in addition be shown. Furthermore, all economic data will be presented with a finishing diagram for a visual overview. If more than one vehicle are being exported, they will be listed one at a time.

4.4 User interface

The system will have a graphical user interface for the user to quickly have a transparent overview. Furthermore, the system will run on a computer supporting .NET Framework. The following describes the components of the graphical user interface:

Start page

When the client is run, the start page will appear, from which seven options are presented to the user along with warnings about upcoming events, such as odometer readings or when a leased vehicle is close to end-date according to the leasing agreement. The start page will have an easy access from any point in the system.

Find single vehicle

This component appears when the user clicks *"Enkelt køretøj"*(Single vehicle) at the start page, and will ask the user to enter one of three parameters for the system to be able to find the desired vehicle. A vehicle will either be identified by registration number, case number or diesel cards attached to the vehicle.

Show single vehicle

This component appears when the user has entered search parameter and pressed "Find" on the previous page *Find single vehicle*. It will show the vehicle information corresponding to the search parameters of the vehicle.

Create new vehicle

This component enables the user to create and add a new vehicle to the existing list. When the user clicks *"Opret køretøj"*(Create vehicle) the system will ask the user to fill in information regarding the vehicle requested for creation. When sufficient information is filled in the user can click the *"Opret"*(Create).

Vehicle overview

By clicking *"Køretøjsoversigt"*(Vehicle overview) the system will display all vehicles. Most interactions regarding vehicles will be available from this page. The user can be directed to create a vehicle by clicking *"Opret køretøj"*(Create vehicle). The user can be directed to edit vehicle information by marking a vehicle in the list and clicking *"Rediger markeret"*(Edit marked). Finally, the user can be directed to edit odometers by clicking *"Rediger km"*(Edit odometer), *"Generer rapport"*(Generate report), *"Generer excel"* (Generate

excel) and *"Slet køretøj"* (Delete vehicle).

Edit Odometer

This component enables the user to edit the odometer data of all vehicles at once. This component appears when the user clicks *"Rediger km"*(Edit odometer) at the Vehicle overview page as explained in Vehicle overview description. The system will display all vehicles by registration number listed with their previous entered odometer readings together with a text box in which the user can enter the new readings and save by pressing *"Gem"*.

Report damage

Report damage will also be available from the start page and will ask the user to input specific information like date, description, workshop and registration number corresponding to the damage and then save the information.

Search by parameters

This component enables the user to have a list presented with desired parameters. It is available from the start page by clicking *"Søg køretøjer"*(Search vehicles). The user can choose parameters in check boxes and press "Find", which will make the system display all vehicles corresponding only to those parameters.

Damage overview

Similar to the Vehicle overview component, just by listing damages instead of vehicles. This component will likewise direct the user to all functionality regarding damages by button clicks, such as "Opret skade"(Create vehicle), "Rediger markeret"(Edit marked) and "Slet markeret"(Delete marked).

Presentation of data

The presentation is not a component showed in the navigation diagram, it is a general component which will be available from almost every point in the system. *"Show single vehicle"*, "Vehicle overview" and "Search by parameters" will all have the functionality of being exported into a presentation which the user can print and use for reports.

Overview

A navigation diagram serves as the skeleton of the interface that the user will interact with. It aims at providing an overview for the user to easily find the information needed. Below is a navigation diagram for the user interface needed at Lasse Larsen Byggefirma A/S, where every "screenshot" is a component explained in the previous section 4.4.

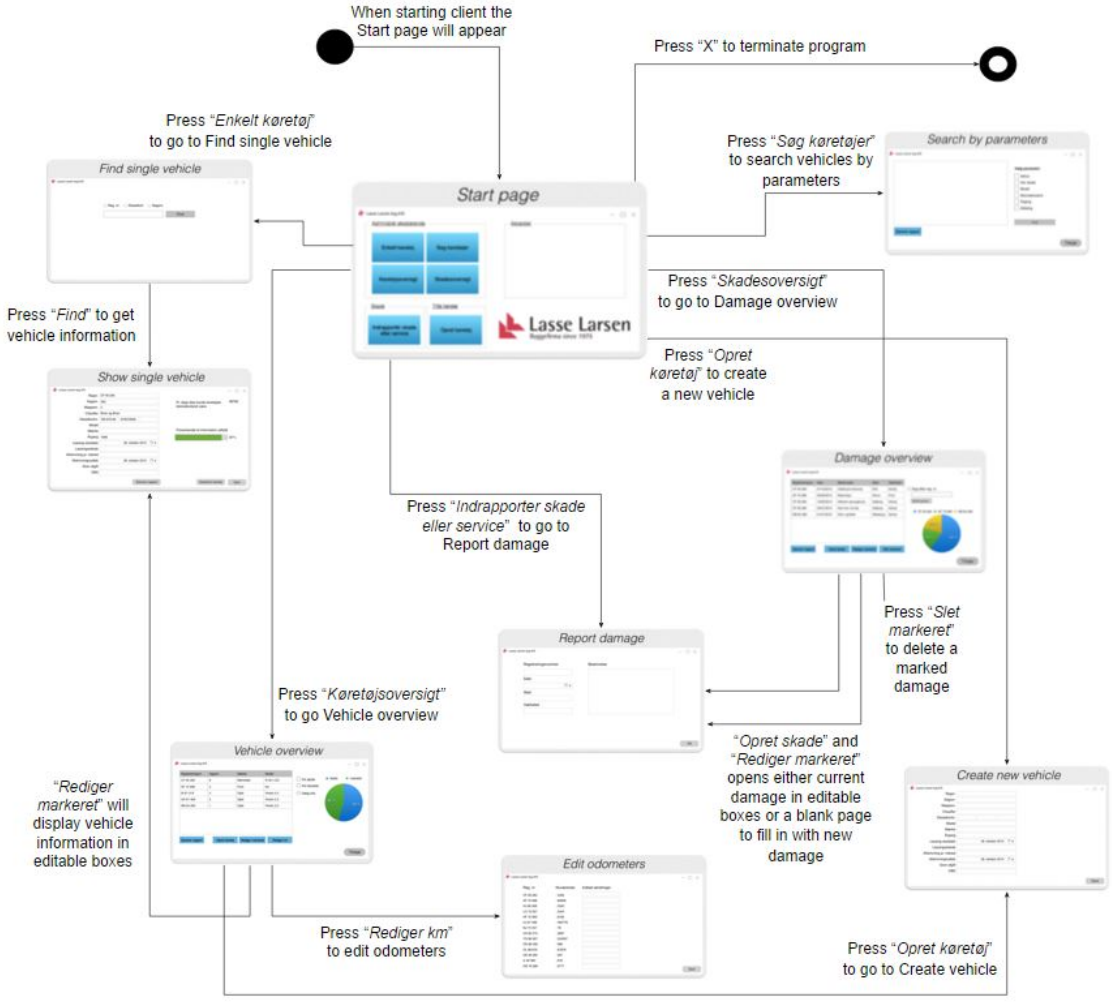


Figure 4.19: Full navigation diagram

The illustrations in figure 4.19 is not the final appearance for the system to be developed, but the navigation and interactions are final. A larger scaling of the figure can be found in Appendix A.

Example

The full navigation diagram can be overwhelming due to the many components with additionally many buttons, which are not all explained. Therefore the interactions with *Køretøjsoversigt* (Vehicle overview) will be presented below with detailing description as an example.

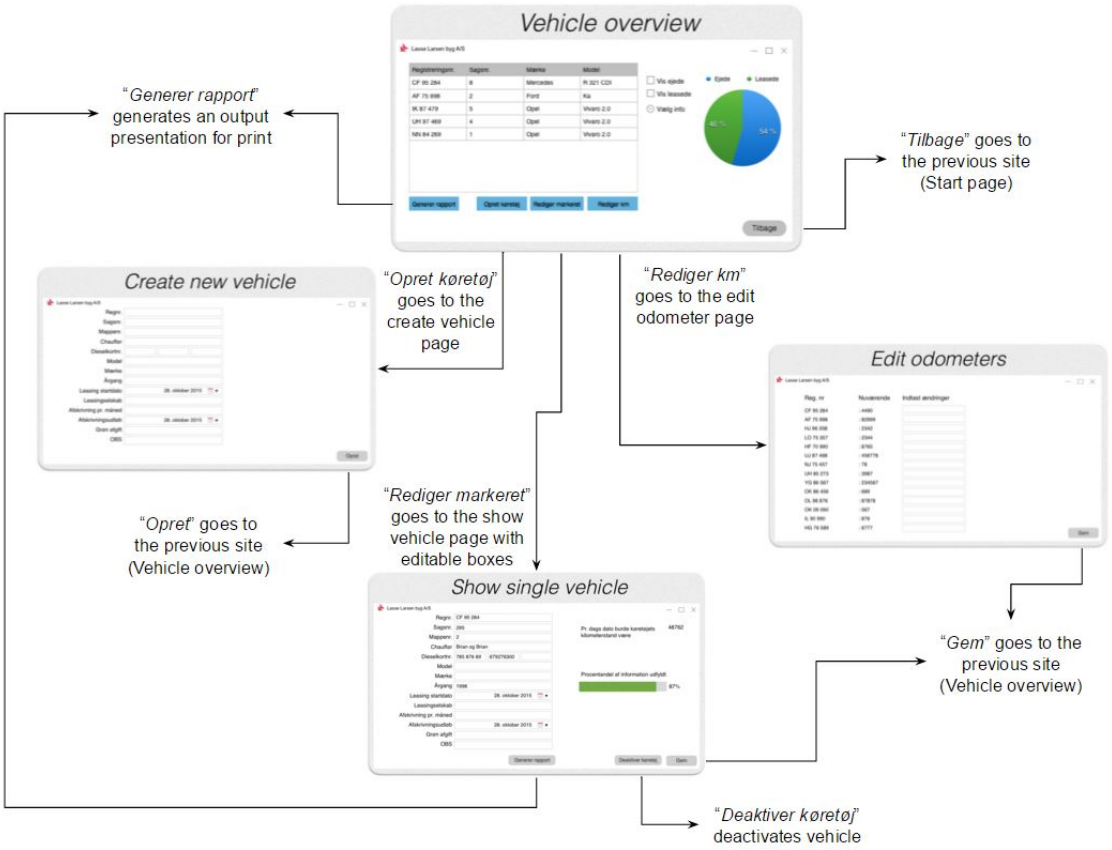


Figure 4.20: Navigation for Show or edit single vehicle information

The navigations in figure 4.20 starts at the *Køretøjsoversigt* (Vehicle overview) where the user can choose from different tasks. To the right of the vehicle list there will be a graphical presentation of some data of the vehicles together with check boxes where the user can choose which information to be shown. There will also be a graphical presentation together with the single vehicle at the "Show single vehicle" page.

System design 5

This chapter will present and describe the importance of different criteria to the system and show the layers and architecture for the reader to get a graphical overview of the design.

5.1 Criteria

The quality of a system is based on its lack of weaknesses. A faulty system can be disastrous in the late developing phase or after release. Object oriented design is a way to eliminate all significant insurgencies and it helps to set up criteria for the design. The criteria are weighted in a five point scale by the designers for simplicity and overview.

Criteria	Very important	Important	Less important	Irrelevant	Trivial fulfilled
Useful	✓				
Secure			✓		
Effectively			✓		
Correct	✓				
Reliable		✓			
Maintainable				✓	
Testable	✓				
Flexible				✓	
Comprehensible		✓			
Reusable				✓	
Portable				✓	
Interoperable				✓	

Figure 5.1: Criteria prioritizing of C.A.R.S.

In the following the criteria will be described to elaborate on choices of prioritizing regarding check marks in figure 5.1.

Useful (Very important)

The usefulness of this system has been categorized as very important. A useful system is a system, which requires no or very little adaptation from the users.

It is important that this system enables the users to easily navigate and make full use of it. In addition, the system should be able to satisfy the users to the expected extent.

Secure (Less important)

The security of this system is categorized to be less important due to the fact that the system will not be handling any safety-critical information. At the same time the system will mainly be used by one user, the head of administration, and the system will at no time require connection to the internet and therefore the access to the system is limited.

Efficient (Less important)

It is less important that the system is effective, because the goal of this system is not to achieve perfect running times and this is not a requirement. In addition, the system will only be handling small amounts of data, therefore optimizing the performance of the system is less important.

Correct (Very important)

Is it very important that the system fulfil the requirements set for it, as the overall functionality depends on whether the requirements are fulfilled or not. If the system lacks fulfilment of the requirements, the system could in some aspects be inadequate or useless.

Reliable (Important)

For the system to work properly, the search and sort functions will have to be reliable for the user, which makes the systems reliability important.

Maintainable (Irrelevant)

The system will be very testable, which makes the ability to maintain the system less important. After the system has been tested, it will not need to be maintained as often because the system should already work properly.

Testable (Very important)

The system will have to be well tested before a user will be allowed to access it. Therefore, the system should be easy to test, to avoid errors after release.

Flexible (Irrelevant)

The system will have no need for flexibility due to the limited scope of the project.

Comprehensible (Important)

A good comprehension of the system is important for the development. This comprehension will help the development team to ensure reliability of the system.

Reusable (Irrelevant)

The reusability of the system, the ability to be easily implemented into another system, is deemed irrelevant. As it is irrelevant for the success and development of the system. In addition, the system will mainly be used by a single user and is intended as a stand-alone system.

Portable (Irrelevant)

The ability to change the platform of the system is marked as irrelevant, as the system will achieve no major advantages from being moved.

Interoperable (Irrelevant)

As the system has no interaction with external systems or servers, the ability to be interoperable is deemed irrelevant for the system, as it is a stand-alone system. Furthermore, the system is designed to be used mainly by a single user on the stand-alone system.

5.2 Architecture

C.A.R.S. is a small system requested by Lasse Larsen Byggefirma A/S. It is designed by developers through user preferences and requirements. The architecture of the system is set by different components. A component solves a specific sub-problem of the system. A general architecture is shown in figure 5.2. This consists of a model-component at the bottom, which has a model of the system corresponding to the problem domain. The middle layer is the function component which consists of the functionality of the system. This makes the user able to update and use the model component. The top layer is the system interface component, which ties the system to the surroundings. The surroundings include hardware such as IO devices.

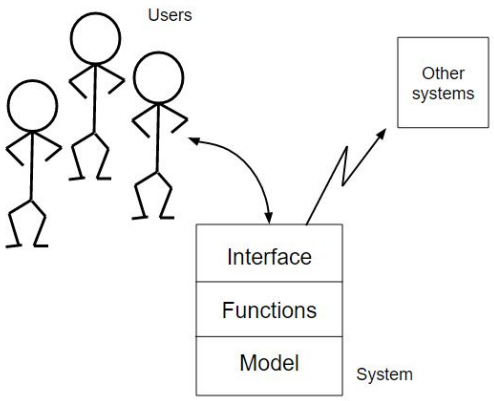


Figure 5.2: General system architecture[3]

The architecture of **C.A.R.S.** is shown in figure 5.3. In addition to the general system architecture this has a command parser component. The command parser serves as a translator between the UI and functionality, and it makes future replacements of the user interface a possibility. The system functions by the client-server relationship. The graphical user interface servers as the client, while remaining functionality is handled by the server.

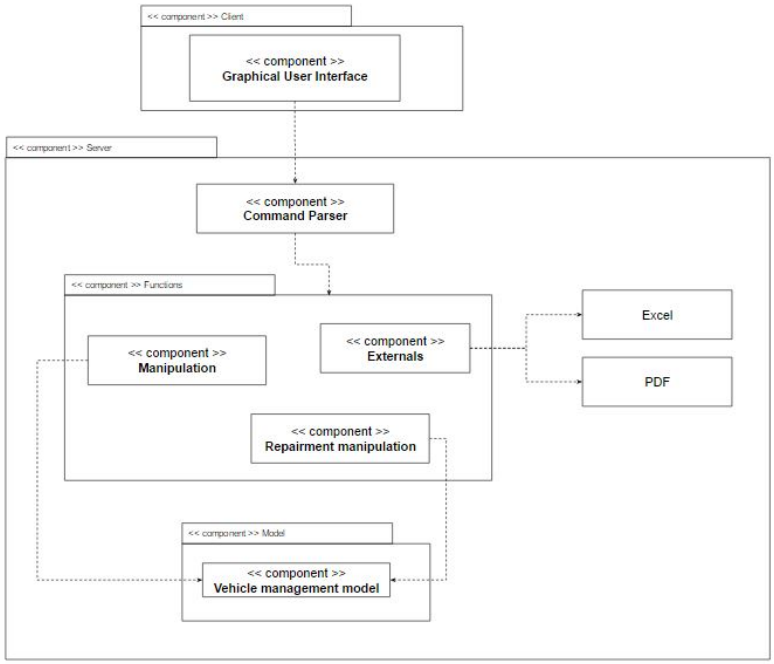


Figure 5.3: Architecture of C.A.R.S.

As shown in figure 5.3 the function component is divided into three sub-function components: *Vehicle manipulation, Repairment manipulation* and *Externals*. Each with different responsibilities. *Vehicle manipulation* and *Repairment manipulation* are able to manipulate respectively vehicle or repairment information in the *Vehicle management model* in the model component of the system, while *External* is responsible for exporting presentations into different formats of output and read from a spreadsheet exported by the economic software system EG Visual.

5.3 Components

In this section the components: *Graphical User Interface*, *Functions*, *Vehicle management model* and *Command parser* will be elaborated with detailed descriptions. The section will finish with an expanded class diagram to show each component with its dependencies.

Graphical user interface

The component *Graphical user interface* consists of the ten different classes as shown in figure 5.4. The different classes represent different windows/usercontrols in the **C.A.R.S.** system.

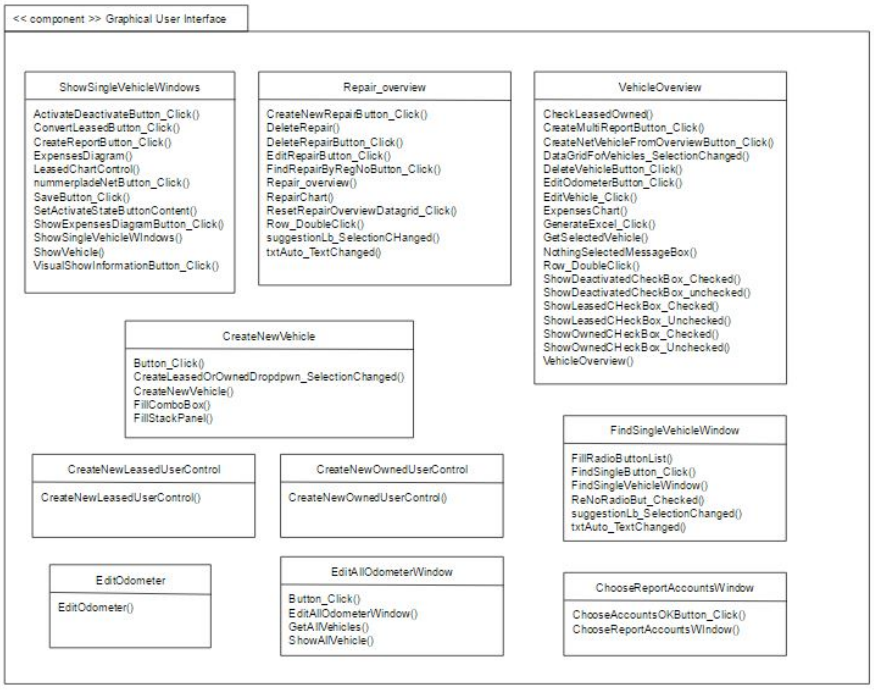


Figure 5.4: Architecture of the Graphical user interface

Not all classes will be explained, but the one with most responsibility, the VehicleOverview class, will as an example be elaborated. The VehicleOverview in the top right position of figure 5.4 holds the functionality of the Vehicle overview page in the system. The functionality handled is the events triggered when a user presses a button, double clicks a vehicle or checks off check boxes.

Functions

The function component in figure 5.5 contains three sub-components, as mentioned in the architecture description. The *Vehicle manipulation* component and the *Repairment manipulation* component each consists of one class to respectively manipulate or access information in either the *Vehicle* or the *Repairment* classes in the model of the system.
The final sub-component of the functions component, *External*, contains three different classes. *External* has the functionality to either read from, or write to different external file formats. The *External* component has three jobs, hence the

division into further three sub-components. The component *Export to pdf* will have the functionality to export the different layouts of presentation into a pdf format. This format of output file is requested by the head of administration for her to print out informations to use as reports for presentation meetings. The class *Export to excel* will, as the name suggests, export chosen information by the user into a spreadsheet. This is also an output file format desired by the head of administration, for her to be able to further manipulate data. The last sub-component *External* is *Read visual data* and it is the class responsible of reading in the economic data from EG Visual.

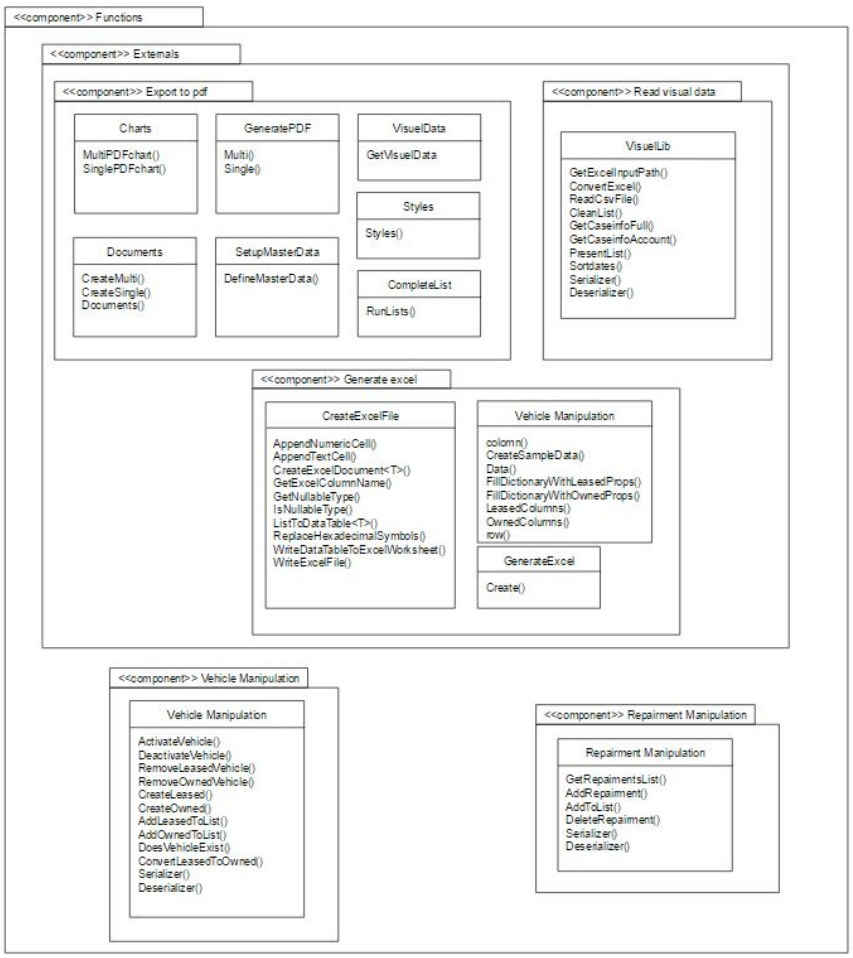


Figure 5.5: Architecture of the Function component

The three sub-components described are shown figure 5.5 with associated method names to list functionality of the classes and components.

Vehicle management model

The Vehicle management model contains the model of the problem domain. In the problem domain it was found that information regarding a vehicle and its repairs were essential. Therefore the model is divided primarily into two parts a vehicle and its repairments. Lasse Larsen Byggefirma A/S' vehicles are not all owned, two sub-classes are assigned to vehicle. The vehicle class sets the properties the two types of vehicles has in common, and the derived classes sets the specifications for exactly that type of vehicle. Likewise, the Repairment class in the Repairments model sets the properties for repairs.

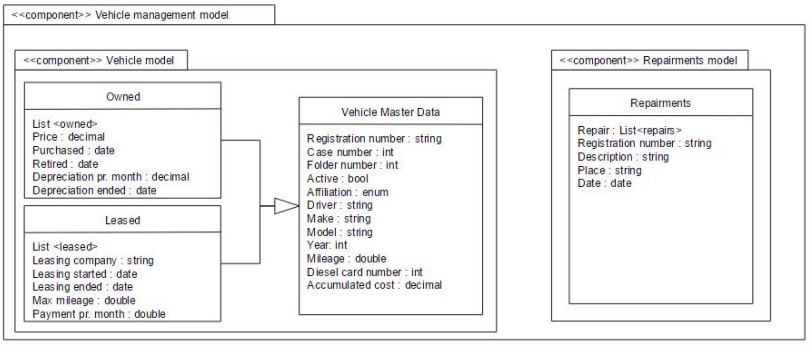


Figure 5.6: Architecture of the Model component

The Vehicle management model is shown in figure 5.6, where it is illustrated how the division into two sub-components of the model looks like and what they each include.

Command parser

Tying everything together is the CommandParser. A snippit of the CommandParser component is shown in figure 5.7. It is responsible for connecting the UI to the model, and acts as a node point for all communication. All method calls go through the CommandParser, which gives some advantages regarding privacy and control flow. Another advantage of having the CommandParser is that it becomes easier to change the UI or port the system to other device types. The choice to have a central communication Class also meant that the other system classes could easily be divided into separate Class-librarys that could be tested and worked on individually.

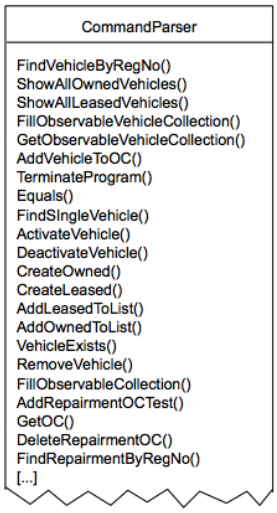


Figure 5.7: CommandParser

Overview

Below is the parts of the architecture explained in the Component section put together with dependencies. The classes are collapsed, which means it is only the class names that are shown in the diagram.

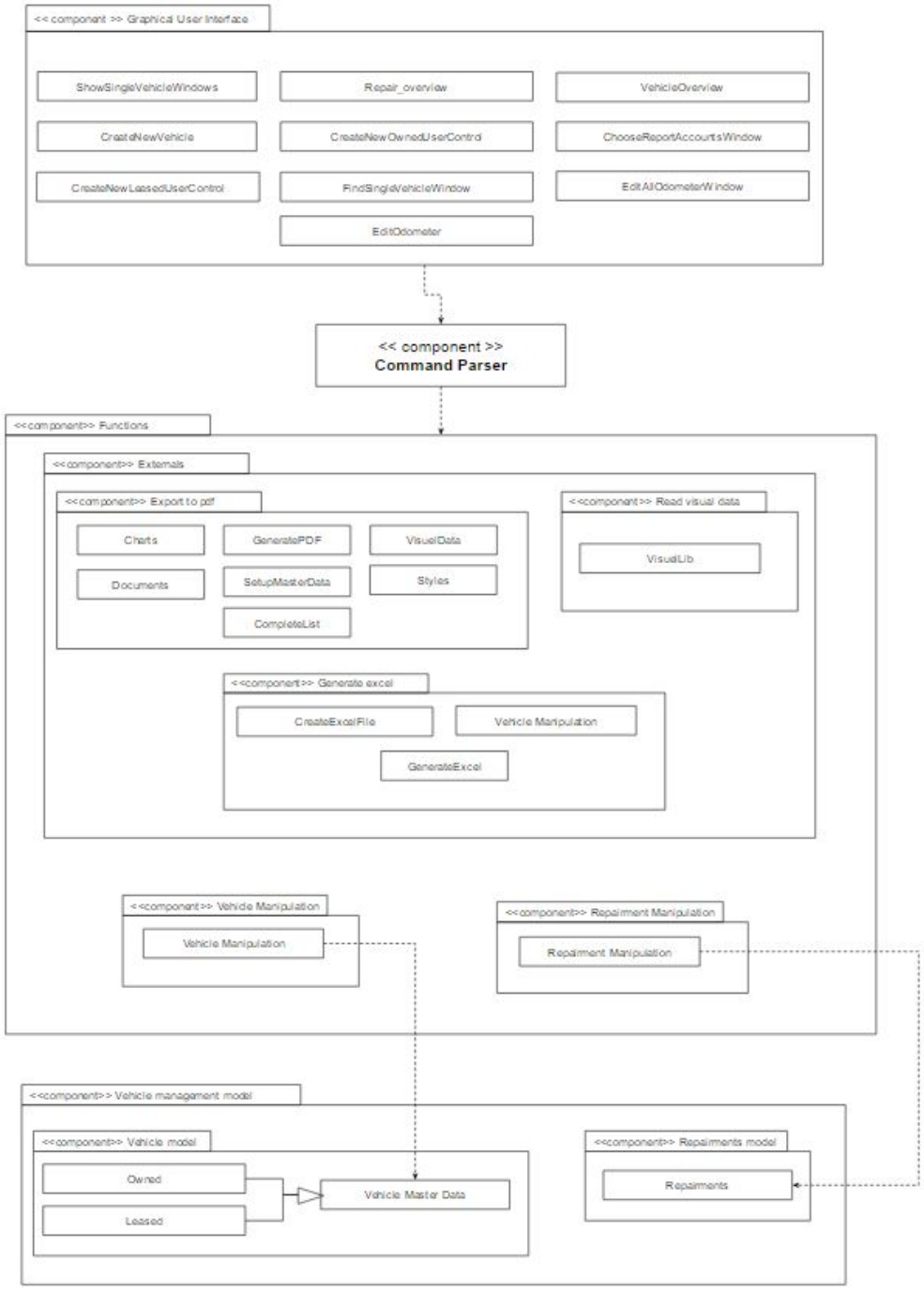


Figure 5.8: Component architecture of **C.A.R.S.** system

5.4 Future Scope of the Report

Due to limited time resources the External component will be developed with inspirations from works others have done and made work. It will then be adapted to be used by the system, but not completely written by the developers of the **C.A.R.S.** system. Furthermore, the user will only be a part of testing the first prototype, a late prototype and the final system.

Implementation 6

C.A.R.S. *is implemented using C# 5.0 and .NET 4.5/4.6. This chapter will describe the development method and main points of the implementation, including the process of reading and exporting the data.*

6.1 Test Driven Development

As a result of the failures and successes throughout development history Test Driven Development (TDD) has become one of the most important concepts in modern software development. TDD helps to ensure developers that they understand the requirements of the system by engaging the user in the process to define the logic that is being created and it helps developers to understand when the project is done. TDD is simple as a development practice. By using TDD you start by writing a test, also known as test first development, and not by creating a class or similar approaches. This method, by writing the test first, gives the requirements for which you are designing in code.

Benefits of TDD

Test Driven Development has shown to be advantageous in many ways. Below is a list of some of the benefits of doing TDD [**?**] :

- TDD ensures quality code. Developers are encouraged to write only the code needed to make the test pass and thus fulfil the requirements. Less code has fewer opportunities for error.
- TDD ensures a high degree of fidelity between the code and the user requirements. If requirements are written as tests, and all tests pass, the code meets the needs of the user.
- TDD encourages communication with the user to make sure that the input and output combinations make sense.
- TDD provides built-in regression testing. When changes are made, the tests will ensure that today's changes do not damage yesterday's functionality.

Writing tests

There are several steps in writing TDD code, and also steps which are predicted to fail at first try. When writing the first test it will fail due to the fact that the application does not compile. The test will attempt to use a method that does not yet exist or attempt to instantiate a class that has not been defined.

1. First step: to create the class and method in the class you want to test. This step will fail due to fact that the class and method just created do not do anything yet.
2. Second step: write just enough code for the test to pass, and no more. This keeps the code simple and easy to understand.
3. Third step: when the first test passes - write more tests. There should be enough tests to ensure all the requirements are being tested with multiple inputs both in- and outside the approved range.

6.2 Source code descriptions

In this section code descriptions will be made, to explain essential functionality of how the system works. The functionality to be examined will be some of the requirements made by the user, such as storing data, exporting to excel and pdf and the code for loading economic data from the EG Visual excel export. Furthermore, parts of the graphical user interface will be described.

De/Serializer

Below are the two serialization methods contained within the `VehicleManipulation` class. These methods serialize and de-serialize the data regarding the owned- and leased vehicles. The serialization is binary, using the `BinaryFormatter` class.

```
1  public void Serializer()
2  {
3      BinaryFormatter formatter1 = new BinaryFormatter();
4      Stream streams1 = new FileStream(("ownedvehicles.bin"),
           FileMode.OpenOrCreate, FileAccess.Write, FileShare.None);
5
6      formatter1.Serialize(streams1, OwnedVehicle.ownedVehicles);
7      streams1.Close();
8
9      BinaryFormatter formatter2 = new BinaryFormatter();
10     Stream streams2 = new FileStream(("leasedvehicles.bin"),
           FileMode.OpenOrCreate, FileAccess.Write, FileShare.None);
11
12     formatter2.Serialize(streams2, LeasedVehicle.leasedVehicles);
13     streams2.Close();
14 }
```

Listing 6.1: The serializer methode from source code

The `Serializer()` writes the `OwnedVehicle` and `LeasedVehicle` lists into their respective files. This adds a small amount of security as the files are not as easily read by third parties as a regular text file. The serialization process for `OwnedVehicles` is done in lines 4-8, as shown in listing 6.1. The `BinaryFormatter` is being instantiated in line 4, and the stream is being instantiated in line 5. The stream is then serialized and written to the designated file in line 7, with the file being closed in line 8 with the call `streams1.Close()`. The same applies for `LeasedVehicles`, which starts in line 10 by creating a new instance and all the way down to the close of the stream in line 14.

```
1  public void DeSerializer()
2  {
3      BinaryFormatter formatter1 = new BinaryFormatter();
4      Stream stream1 = new FileStream("ownedvehicles.bin",
           FileMode.Open, FileAccess.Read,
5      FileShare.Read);
6
7      OwnedVehicle.ownedVehicles =
           (List<OwnedVehicle>)formatter1.Deserialize(stream1);
8      stream1.Close();
9
10     BinaryFormatter formatter2 = new BinaryFormatter();
11     Stream stream2 = new FileStream("leasedvehicles.bin",
           FileMode.Open, FileAccess.Read,
12     FileShare.Read);
13
14     LeasedVehicle.leasedVehicles =
           (List<LeasedVehicle>)formatter2.Deserialize(stream2);
15     stream2.Close();
16 }
```

Listing 6.2: the deserializer method from source code

The `DeSerializer()` does, as the name implies, the opposite of the `Serializer()`. It reads the binary data from the designated files, and writes them to their respective lists and finally closing the files.

Reading Visual Data

One of the important features the user requested was the ability to import economic data from their economy management system. The system used is EG Visual[4]. From this system it is requested to use an exported Excel file with the system and have the relevant data presented alongside the vehicles, both in the GUI and more importantly in the generated reports. An example of the Visual data shown in the GUI is presented in figure 6.1 where the diagram to the right shows the economic data corresponding to the vehicle shown on the left.

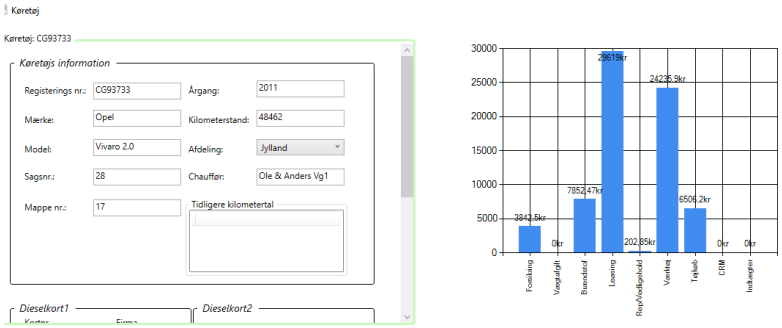


Figure 6.1: Illustrations of Visual data in the system

The `Visuallib` class is responsible for reading the Excel file containing the exported Visual data, converting it into a list of strings, cleaning those strings of excess characters, sorting the information by case and account number making the data easily accessible by the system.

```
1  //Returns a list containing economic data concerning the
       selected vehicle and accounts
2  public List<string> GetCaseInfoAccount(int caseNo, int accountNo)
3          {
4              GetCaseInfoFull(caseNo);
5              List<string> AccountResults= new List<string>();
6              bool activeAccount=false;
7              if (results == null) { return results; }
8              foreach (string line in results) {
9          //Identifies the lines beginning the selected account
10                 if (line.StartsWith(accountNo.ToString())) {
11                     activeAccount = true;
12                 }
13         //Identifies the lines beginning an account
14         else if (line.StartsWith("100")
15     || line.StartsWith("200") || line.StartsWith("210")
16     || line.StartsWith("850") || line.StartsWith("851")
17     || line.StartsWith("852") || line.StartsWith("853")
18     || line.StartsWith("854") || line.StartsWith("855")
19     || line.StartsWith("870")) {
20                     activeAccount = false;
21                 }
22                 if (activeAccount) {
23                     AccountResults.Add(line);
24                 }
25             }
26             return AccountResults;
27         }
```

Listing 6.3: Account data connected to vehicle

Listing 6.3 shows a method that returns selected account information regarding a vehicle.

The `Visuallib` class likewise has a `Serializer()` and a `Deserializer()`. The `Serializer()` is used on shut-down of the system to serialize current loaded data and on start-up the `Deserializer()` allows the system to maintain the latest set of loaded data between sessions.

Generate PDF document

The user has the option to choose between two different styles of an auto generated PDF document through the system. One, which contains information about a single vehicle, and one for several vehicles. The `GeneratePDF` class contains two methods `Single()` and `Multi()` which determines if the user wants a PDF report for a single vehicle or multiple vehicles. Two methods are presented in the `Documents` class, which are named `CreateSingle()` and `CreateMulti()`. Both the methods create a new document and apply the different text styles, like fonts and sizes for the texture in the document using the method `styles()` presented in the class `styles`. When the document has been created and styled, the `CreateMulti()` or `CreateSingle()` method calls the method `DefineMasterData()` located in the class `SetupMasterData` which add a page with a table of master data for a single vehicle.

```
1  foreach (PropertyInfo item in
       vehicle.GetType().GetProperties())
2  {
3      if (item.PropertyType == typeof(DateTime))
4      {
5          DateTime date =
               (DateTime)item.GetValue(vehicle);
6          propnames.Add(propList.ElementAt(i),
               date.ToShortDateString());
7      }
8      else if (item.GetValue(vehicle) != null)
9      {
10         propnames.Add(propList.ElementAt(i),
               item.GetValue(vehicle).ToString());
11     }
12     else
13     {
14         propnames.Add(propList.ElementAt(i), " ");
15     }
16     i++;
17 }
```

Listing 6.4: Adds the appropriate name and value to a dictionary

The `foreach` loop represented in listing 6.4 is located in `DefineMasterData()`. It loops through each property of the vehicle and adds their value, if not null as seen in line 8, and the appropriated Danish name of the property, to a dictionary. Addition of the Danish name, was a requirement made by the head of administration af Lasse Larsen Byggefirma A/S.

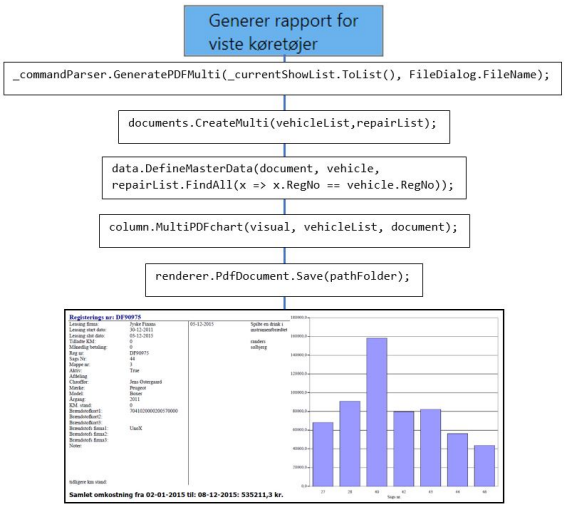


Figure 6.2: Flow to generate pdf

When the dictionary is complete, it is added to the document as illustrated in figure 6.2, where the user has pressed the button "Generer rapport for viste køretøjer"(Generate report for shown vehicles), which generates a report for several vehicles listed one by one with Danish names.
Plugins as Migradoc and PdfSharp[5] has been used to create documents, charts, and converting the finished document to a PDF file.

Generate Excel

The user also has the option to auto generate a spreadsheet file for the vehicles in the system. The class `GenerateExcel` contains one method named `Create()`. The `Create()` method uses the method `CreateSampleData()` in the `Data` class to receive the data for the vehicles. The `CreateSampleData()` calls six private methods in the `Data` class to set-up the data table with columns and rows, as well as dividing the leased and owned vehicles into categories. Two of these methods have been selected and will in the following section be elaborated.

```
1 private void column(string type, string columnName)
2         {
3
4             DataColumn column = new DataColumn();
5             column.DataType = Type.GetType(type);
6             column.ColumnName = columnName;
7             currentDataTable.Columns.Add(column);
8         }
```

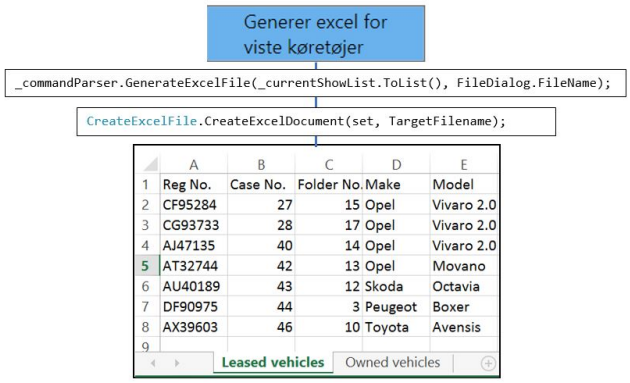
Listing 6.5: Set up columns

The `column()` method shown in listing 6.5 creates a single column with the name of the string parameter in the data table. It is called by the methods `LeasedColumns()` and `OwnedColumns()` to generate a sheet for each of the methods.

```
1  private void row(Vehicle vehicle)
2          {
3              DataRow row = currentDataTable.NewRow();
4
5              foreach (PropertyInfo prop in
                   vehicle.GetType().GetProperties())
6              {
7                  if (propDict.ContainsKey(prop.Name))
8                  {
9                      string columnName;
10                     propDict.TryGetValue(prop.Name, out
                           columnName);
11                     if (currentDataTable.Columns
12                     .Contains(columnName))
13                     {
14                         if (prop.GetValue(vehicle) == null)
15                         {
16                             row[columnName] = "";
17                         }
18                         else
19                         {
20                             row[columnName] =
                                   prop.GetValue(vehicle).ToString();
21                         }
22                         [...]
```

Listing 6.6: Set up rows

The `row()` method shown in listing 6.6 creates the rows for the data table. It is called by the method `CreateSampleData()` which also calls the `LeasedColumns()` and `OwnedColumns()` to combine the columns and rows. It returns the complete data table to `Create()` which will call the method `CreateExcelDocument()` in the class `CreateExcelFile` and convert the data table to a spreadsheet document. This class, including all its methods are an open source project created by Mike Gledhill[6].



	A	B	C	D	E
1	Reg No.	Case No.	Folder No.	Make	Model
2	CF95284	27	15	Opel	Vivaro 2.0
3	CG93733	28	17	Opel	Vivaro 2.0
4	AJ47135	40	14	Opel	Vivaro 2.0
5	AT32744	42	13	Opel	Movano
6	AU40189	43	12	Skoda	Octavia
7	DF90975	44	3	Peugeot	Boxer
8	AX39603	46	10	Toyota	Avensis
9					

Figure 6.3: Flow to generate spreadsheet

Figure 6.3 shows the flow when the user have pressed the button "Generer excel for viste køretøjer" (Generate excel for shown vehicles), which generates the spreadsheet document.

Graphical User Interface

The Graphical User Interface (GUI) was implemented using Windows Presentation Foundation (WPF). The WPF GUI was implemented using the markup language XAML (eXtensible Application Markup Language). Windows Presentation Foundation has a lot more customizability and clear separation between the User Interface and the logic. Knowing that the User Interface is strictly separated from the logic, it is common to implement the MVVM pattern (Model – View – ViewModel) in WPF applications. WPF is meant to be a unified platform for modern User Interfaces, as seen in figure 6.4 [7].

	Windows Forms	PDF	Windows Forms/ GDI+	Windows Media Player	Direct3D	WPF
Graphical interface, e.g., forms and controls	✗					✗
On-screen documents	✗					✗
Fixed-format documents		✗				✗
Images			✗			✗
Video and audio				✗		✗
Two-dimensional graphics			✗			✗
Three-dimensional graphics					✗	✗

Figure 6.4: Overview of WPF compared to User Interfaces [7]

An example of the XAML code used in the GUI of the **C.A.R.S.** system can be seen in listing 6.7. The code is for the GroupBox section "Administrér eksisterende" (administrate existing [vehicles]). The box contains four buttons, which are all defined as seen in line 3 - 6. These buttons are given a name that is shown when the system is run.

```
1 <GroupBox Header="Administrer eksisterende"
2         [...]
3         <Button x:Name="AllVehiclesButton" Content="Koretojs
              oversigt" HorizontalAlignment="Left"
              VerticalAlignment="Top" Width="130" Height="75"
              Margin="11,96,0,0" Click="AllVehiclesButton_Click"
              Background="#FF69B2EE"/>
4         <Button x:Name="SingleVehicleButton" Content="Find
              enkelt koretoj" HorizontalAlignment="Left"
              VerticalAlignment="Top" Width="130" Height="75"
              Margin="11,10,0,0" Click="SingleVehicleButton_Click"
              Background="#FF69B2EE"/>
5         <Button x:Name="RepairOverviewButton"
              Content="Skadeoversigt" HorizontalAlignment="Left"
              VerticalAlignment="Top" Width="132" Height="75"
              Margin="146,96,0,0"
              Click="RepairOverviewButton_Click"
              Background="#FF69B2EE"/>
6         <Button x:Name="SearchButton" Content="Sog koretojer"
              HorizontalAlignment="Left" VerticalAlignment="Top"
              Width="132" Height="75" Margin="146,10,0,0"
              Background="#FF69B2EE" Click="SearchButton_Click"/>
7         </Grid>
```

Listing 6.7: Example of XAML

The Graphical User Interface was designed to accommodate the user's needs, being simple in design and contain all the necessary information. The start page gives the user access to various tasks, as described in Section 4.4.

Vehicle overview

As specified in the requirement specification in section 4.2 the user explicitly emphasized that the system had to enable her to get a quick overview of their complete fleet of vehicles. Therefore, an important feature of the system is the vehicle overview section. The overviews main responsibility is to give the user short and trimmed information regarding every single vehicle.

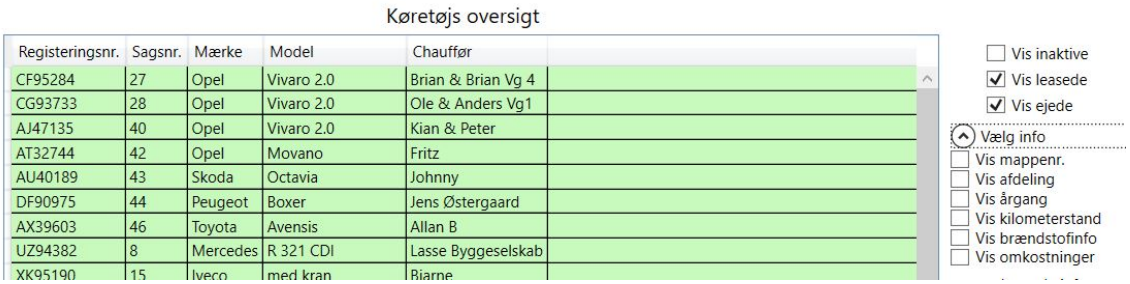


Figure 6.5: Vehicle overview clip

Figure 6.5 shows, which pieces of information regarding vehicle master data the system will present to the user upon accessing the vehicle overview. In order to access even more master data the system enables the user to choose whatever information might be relevant through a listing of check boxes. In addition, the user can choose to filter the presented vehicles by vehicle types. Some of these filter options can be seen to the right in figure 6.5. This functionality was crucial to the user because of the desire to almost instantly get an overview over their fleet and identify vehicles.

```
1 <CheckBox x:Name="ShowFolderNo" Content="Vis mappenr."/>
2 <DataGridTextColumn Header="Mappenr." Binding="{Binding
    FolderNr}" Visibility="{Binding Source={x:Reference
    Name=ShowFolderNo}, Path=IsChecked, Converter={StaticResource
    BooleanToVisibilityConverter}}"/>
```

Listing 6.8: Column binding

These master data filter options is provided to the user by accessing the visibility property of each column and binding that to the state, checked and unchecked, of the matching checkbox. Due to the fact that the states of a checkbox are Boolean values, this type of visibility binding requires the use of a class called BooleanToVisibilityConverter, which, as the name implies, can convert Boolean values to and from visibility enumeration values. Listing 6.8 shows an example of this visibility binding.

```
1 foreach (OwnedVehicle vehicle in
      _commandParser.ShowAllOwnedVehicles().Where(x=> x.Active ==
      true))
2             {
3                 _currentShowList.Add(vehicle);
4
5             }
```

Listing 6.9: Show type

The filter option to only show some types of vehicles is done programmatically by removing and adding the desired types from the collection currently being displayed to the user. By default when the user access the vehicle overview both types of vehicles are present. As figure 6.5 shows the user has the possibility to check or uncheck "Vis Leasede" (Show leased), "Vis ejede" (Show owned) and "Vis inaktive" (Show inactive). By changing the check-state of these check boxes the system removes or adds the corresponding types of vehicles from the collection currently showed. Listing 6.9 shows the piece of code that is responsible for adding every active owned vehicle to the collection, when the check-state is changed.

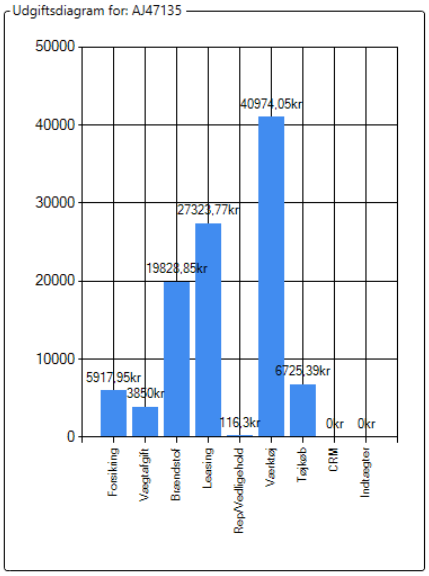


Figure 6.6: Expenses chart

Accompanying the master data in the vehicle overview is a column chart, see figure 6.6, displaying economically data for whatever vehicle that is currently selected in the list of presented vehicles. The economically data is from the users accounting system, EG Visual. Every column represents an expense account and their height represents the amount of DKK used and credited the respective account number. In collaboration, the master data and expenses chart enables the user to get an overview of a desired vehicle without having to browse multiple documents or systems.

The vehicle overview also provides some functionality, which allows the user to export information, delete, create and edit vehicles through a series of controls. These controls and their following action is shown in figure 6.7. When the vehicle overview is accessed the user has the following ways to interact with the system.

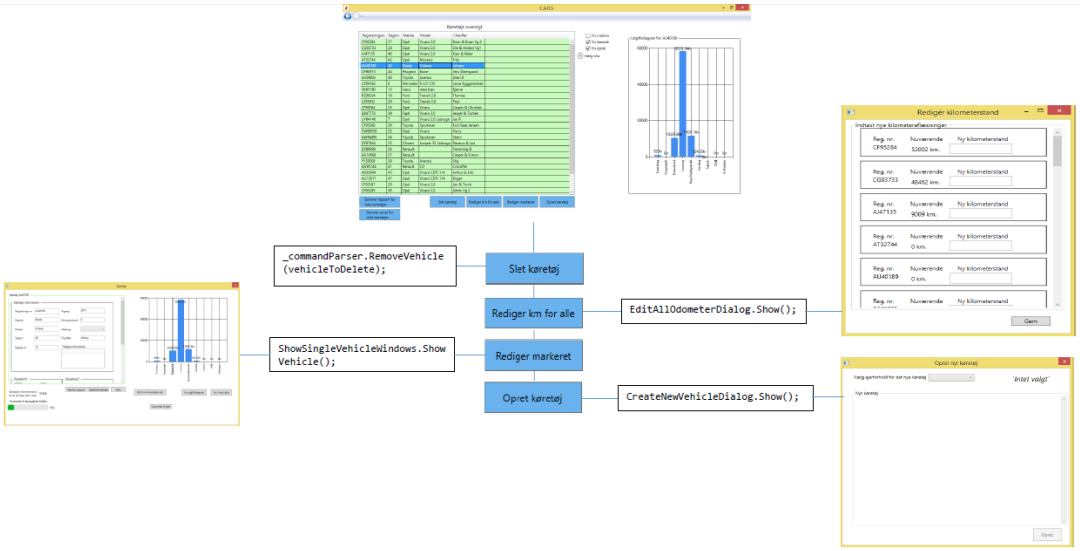


Figure 6.7: Flow of vehicle overview

- *"Slet køretøj"*(Delete vehicle): When clicked, this button calls the method _commandParser.RemoveVehicle with the vehicle selected in the list. This vehicle will then be removed from the list of vehicles in the system.
- *"Rediger km for alle"*(Edit odometer for all vehicle): When the user clicks this button, the system will show a dialogue with all active vehicles, their current mileage and a possibility to input new ones.
- *"Rediger markeret"* (Edit marked): When the user clicks this button, the system will open a windows with all the master data, economic data and some options for editing information regarding the selected vehicle.
- *"Opret køretøj"*(Create vehicle): This button enables the user to create a new vehicle and save it into the system.

Repairments

The repairment part of the system was a completely new addition to the fleet management system currently used. At the same time, it was highly desired to include this functionality into the system. Therefore, this part quite naturally became an important part of the development process. As the requirement specification in section 4.2 also shows there were some well-defined requirements to this part of the system. The system enables the user to create and store information regarding repairments of all types. The user specified what information a repairment would include, and at the same time what information the system had to be able to store.

Figure 6.8 shows how the system, when accessing the component, allows the user to input the information regarding a repairment through a series of UI elements, for example a textbox as see in listing 6.10. The system creates an instance of the class repairment where the properties will be bound to the aforementioned UI elements when the user presses the save button. The system will dynamically show a list of registration numbers matching the so far inputted registration associated with the repairment under creation. By giving the user the ability to see and choose the desired registration number, the user can easily associate any new repairment with any vehicle currently stored and active in the system.

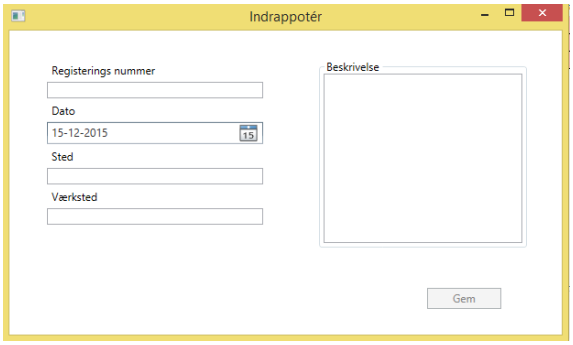


Figure 6.8: Create repairment

```
1 <TextBox x:Name="RegNoTextBox" Text="{Binding RegNo}"
    TabIndex="1"/>
```

Listing 6.10: Textbox

When the user has inputted all information, the user can save the new repairment in the system. With repairments stored in the system, the user has the possibility to see and find these. Figure 6.9 shows how the system presents the user with a complete overview over repairments.

When accessing the overview, as shown in figure 6.9 the user has the possibility to filter the list by searching repairments by registration number. By inputting a registration number and pressing the button find, the system will search through the list of stored repairments and display the matching ones to the user. The code block responsible for finding the matching repairments can be seen in listing 6.11.

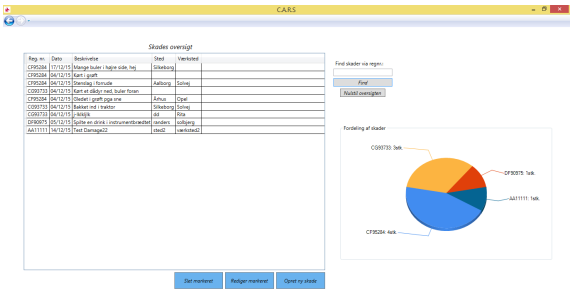


Figure 6.9: Repair overview

```
1 ObservableCollection<Repairment> foundRepairs =
      _commandParser.FindRepairmentsByRegNo
2 (FindRepairByRegNoTextbox.Text.ToUpper().Replace(" ", ""));
3 DataGrid.ItemsSource = foundRepairs;
```

Listing 6.11: Find repair code snippet

At any time, the user can reset the currently shown collection by pressing the "nulstil oversigt" (Reset overview) button, which will display a full list of repairments to the user again. The repairment component of the system gives the user a completely new possibility to quickly obtain an overview over repairments of each vehicle. This enables the user to evaluate upon the vehicles performance compared to others, and hopefully in the future save the company money by identifying patterns in some vehicle groups or brands.

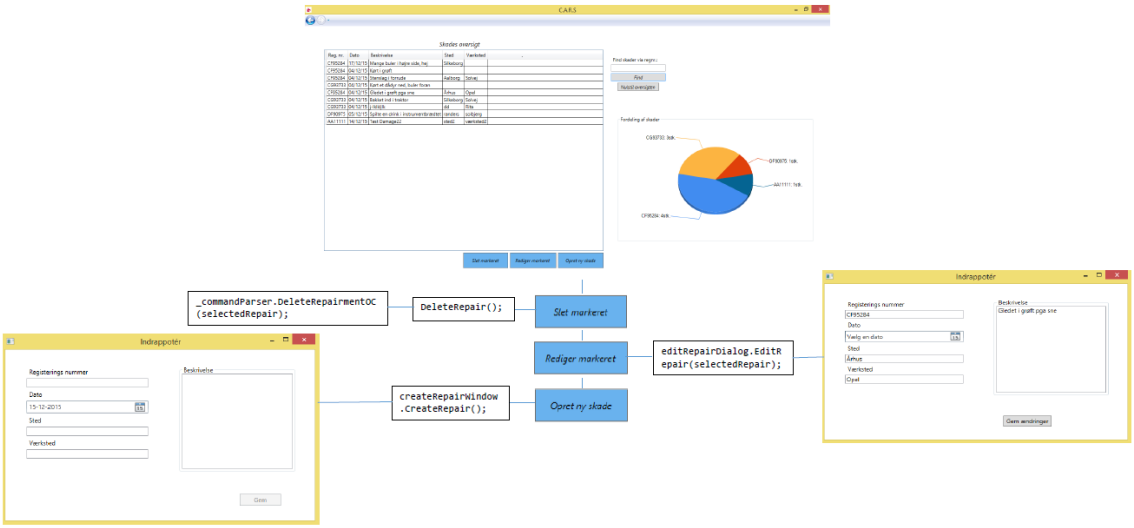


Figure 6.10: Flow of repairment overview

In addition the repairment component offers some functionality to edit or add repairment as shown in figure 6.10, the user has the following ways to interact with the system.

- *"Slet markeret"* (Delete marked): When clicked the selected repairment from the repairment overview is removed from the list of all repairments.
- *"Rediger markeret"* (Edit marked): When user enables this component, this system will show a window with all the information regarding the marked repairment.
- *"Opret ny skade"* (Create new repairment): This button enables the user to create a new repairment with the associated information, and save it into the system.

Tests 7

*This chapter will describe the various ways in which the **C.A.R.S.** system has been tested. The chapter will describe how to set up unit tests and in addition show an example from the project. Then a short presentation of intergration test will be introduced, and finally the two methods for usability testing will be presented with findings.*

7.1 Unit Testing

Unit testing is a testing method that tests small parts of the system individually and isolated. Exactly this individuality and small tests are the biggest advantage of unit testing. It enables the developer to easily identify where a possible bug is located, which can save time used on debugging. To achieve the full effect on code quality, unit testing has been an integral part of the developments process. Either a unit test can be carried out successfully or it can fail.
A good unit test is characterized by the following qualities[8]

- It should be automated and repeatable
- It should be easy to implement
- Once written, it should remain for future use
- Anyone should be able to run it
- It should run at the push of a button
- It should run quickly

Microsoft Visual Studio integrated unit testing framework has been used to create the tests, run them and reporting the results. Unit tests are commonly created and written with focus on the AAA (Arrange, Act, Assert) pattern[9]. These three parts are the core elements.

- The Arrange section of a unit test method initializes objects and sets the value of the data that is passed to the method under test.
- The Act section invokes the method under test with the arranged parameters.

- The Assert section verifies that the action of the method under test behaves as expected.

The framework provides a very suitable collection of statements or methods to indicate passed or failed result of every test, these are part of the assert class. Examples are shown in section 7.1 in the Test chapter of this report.

Example of Unit testing

Figure 7.1 shows an example of the unit testing done on the source code for **C.A.R.S.** The test tests whether a vehicle is actually set to active.

```
[TestMethod]
public void ActivateVehicle_ActivateStateTrue_True()
{
    //Arrange..
    testVehicle.Active = false;
    //Act..
    vehicleManipulator.ActivateVehicle(testVehicle);
    //Assert..
    Assert.IsTrue(testVehicle.Active);

}
```

Figure 7.1: Example Unit testing

Prior to the tests two methods are declared used to ensure isolation. In these methods objects shared by the tests are initialize or set to null. The unit test in figure 7.1 follows the triple A pattern as described in section 7.1. The arrange part sets the active property of the test vehicle to false. Afterwards the method under test, that is responsible for setting active to true is called. The assert verifies that the output is as expected, and thereby that the method is worked properly.

7.2 Intergration test

White- and black-box testing are two software-testing methods widely used. Both methods are used to inspect and validate that a system works as expected. Due to the time constraints of this project, and white-box tests many similarities to unit-tests, whitebox testing has not been done. The following section will describe black-box testing.

Black-box testing

Black-box testing is a test method where the tester does not know/care about the internal code. This is always user or client based testing where testing is done based on the requirements provided. This testing is done by testers only. It is used to test the overall functionality of a system, and only focuses on input and output therefore one might say that, black-box testing ignores the internal mechanism that is under test. The tester should consider the code, as a black-box, which can not be seen through. The only information the tester needs, is what kind of information can be put into the system and that the black-box will send something back in return. Much of the black-box testing for this system was done by having the end-user complete use-cases during testing sessions[10].

7.3 Usability

This section will describe the two ways in which the system has been tested by people. First, heuristic analysis will be explained with listings and focus areas. Second, the results of user evaluations will be presented.

Usability can be tested in many different ways. Over time many different forms of analysis has been developed to identify problems in development of computer software. Usability involves evaluation of the user interface. To create the best possible user interface for **C.A.R.S.** two forms of usability analysis have been used through the project: Heuristic analysis and User evaluations.

Heuristic analysis

One way of doing usability evaluation is to do heuristic analysis. Heuristic analysis is an expert based inspection of the UI, where the main goal is to identify problems associated with the design throughout recognized usability principles, also called "heuristics"[11]. This type of analysis has been beneficial in the span of the project due to lack of time and limited amount of users.
The heuristic analysis has been made by a member of the group and the heuristics used to evaluate the system **C.A.R.S.** derives from some basic characteristics[11], which have been customized for this specific UI:

- Simple and natural dialogue
- Speak the user's language
- Minimize user memory load

- Consistency
- Feedback
- Clearly marked exits
- Shortcuts
- Good error messages
- Prevent errors
- Help and documentation

Most people automatically perform some sort of heuristic evaluation when looking at computer software by using intuition and common sense. The evaluation done in the project has, however, been a systematic inspection by using the heuristics. The focus has been placed especially around communication and consistency.
Studies has shown that a single evaluation has a problem-found-rate at 35% of the usability problems in the UI[11], therefore the evaluation has been made several times with different focus views. The first inspection had focus on preventing errors by finding places where errors could appear and handle mistakes in the source code and at the same time take measures in the UI such that the user would not end up in such situations. The second inspection was primarily focused around communication in different aspects like the natural dialogue where, in this case, less is considered more. Other aspects were feedback, messages and similar. All had to be in the user's language, so that the system adapts to the user and not the other way around and the user feels secure in using the system. The third and last inspection done by developers were the consistency and especially consistency involving the appearance, where to find specific buttons, what colors different parts of system has and this to correspond to what the color means to the user.

CG92424	31	Opel	Vivaro 2,0	Kurt & Jesper Vg 5
CG92425	32	Opel	Vivaro 2,0	Karl Vg 7
CG93733	28	Opel	Vivaro 2.0	Ole & Anders Vg1

Figure 7.2: Example of color use

An example could be the vehicle list, as shown in figure 7.2, where red indicates an inactive vehicle and green indicates a vehicle is active.

User evaluation

The heuristic analysis was made by an expert and in this case the expert had been a part of the development, hence some functionality may seem trivial to that person, meanwhile others may think differently. Therefore the inspection of the system has

been expanded to involve the user of the system and additionally other suspects with similar computer and work experience. This also enhances the problem-found-rate. By having six people testing the system, studies shows that approximately 85% of usability problems will have been found[11].

#	**Gender**	**Age**	**Profession**	**IT experience**	**C.A.R.S. experience**
1	Female	49	Head of administration at Jels Waterworks	Experienced	New
2	Male	50	Former administrator at Allans Mad	Practised	New
3	Female	51	Head of administration at Lasse Larsen Byggefirma A/S	Experienced	New
4	Female	49	Account manager at Lasse Larsen Byggefirma A/S	Experienced	New
5	Female	58	Subject director at Viborg municipality	Experienced	New
6	Female	20	Student	Experienced	New

Table 7.1: Subjects which have tested the **C.A.R.S.** program

Table 7.1 is the list of subjects, which have been testing the **C.A.R.S.** software. The subjects are matched close to the user, the head of administration at Lasse Larsen A/S, in computer experiences and work, so that they have a mental model not far from the user. The sixth user is included, because Dorte Sørensen from January will have an intern, at approximately that age.

The next table, in table 7.2, shows the list of usability problems found by having users evaluate the program through pre-prepared tasks, so that the evaluations had the same basis. The tasks are attached in Appendix D in Danish and translated into English in Appendix E. The first column of the table holds a simple count of the problems. The second holds a description of the problem. The third column holds a categorization and the last indicates which subjects have experienced the problem during the evaluations. The categorizations of the problems are divided into three levels of critical importance:

- Cosmetic
- Serious
- Critical

#	**Usability problem**	**Category**	**Experienced by**					
			1	2	3	4	5	6
1	Back button was not placed where user expected	Cosmetic	x	x				
2	User clicked on back button on startpage when they wanted to close pop-up window because the pop-up did not fill the whole screen	Cosmetic			x			
3	User double clicks on buttons	Cosmetic			x			
4	User pressed "enter" on keyboard instead of using "OK" buttons	Cosmetic	x		x	x		x
5	User tried to "tab" to next text box when inputting data	Cosmetic			x	x		x
6	It was hard to find damages about a vehicle, when the user had to remember the whole registration number found on another page in the program	Critical	x	x	x		x	x
7	User did not know whether a vehicle was created when they have pressed "create" and went to the list on another page to be sure	Cosmetic			x	x	x	
8	User did not know whether a damage was registered or not and went to check on the list after having created the damage	Cosmetic			x	x	x	
9	User was in doubt if data had been loaded after a while because check-mark disappears after navigation around in the program	Cosmetic		x				
10	User did not understand how to search for specific vehicles by parameters	Serious		x			x	
11	User did not know how to get more information to the list of vehicles presented	Serious					x	

Table 7.2: Usability problems found by subjects

The categorizing in table 7.2 is categorized by worst case. This means that the problem may not have been in the critical area for all subjects, but if at least one experienced a problem as critical it would have been categorized as so. Further more the critical problem in #6 are experienced by a multiple users and will then be considered in another category not mentioned before: catastrophe!

	Delay	Irritation level	Expectation vs reality
Cosmetic	< 1 minute	Low	Small difference
Serious	Several minutes	Medium	Significant difference
Critical	Total User stops	High	Critical difference

Figure 7.3: Table of categorization

The categorization is made from the model in figure 7.3, where time, user's irritation level and expectations vs. reality are the key pointers.

Besides the problems found by the user inspections of the program, the test subjects also filled out a questionnaire, where the last question was if they had any further suggestions about functionality or appearance of the program. The suggestions added to **C.A.R.S.** were:

- *"Could it be possible to stretch out listed vehicles, likewise damages to fill the whole screen instead of being in a scrollable box filling half the screen?"*
  - said by Rita Birkmann
- "I would like the diagrams of the costs in the report to be shown on the screen, when I mark a vehicle in the vehicle list"
  - said by Dorte Sørensen
- "I would very much like an indication on the front screen of the start-date and end-date of the loaded visual project data" - said by Dorte Sørensen

8 Discussion

8.1 Analysis vs. Implementation

In the analysis prior to the implementation, it was specified that the system had to have a function responsible for reading in economic data. When the developers had to implement this function, it was clear that it required more than a simple function. Therefore, a class was created to have this responsible. The class was given the responsibility to read in economic data from an exported file, generated by EG Visual, and trim the data making it readable and presentable for the users. In addition, the initial idea was to implement one class with responsibility for reading in data and for exporting a spreadsheet. As the developers discovered that the extent of this responsibility was to major for a single class to contain, two separate classes was implemented. As mentioned above one class should be in charge of the read in part. The other class was implemented to create and output a spreadsheet file with the desired user input.

8.2 Development

As mentioned in section 6.1 the software development method Test Driven Development (TDD) was used throughout development of **C.A.R.S.** but not to the full extend. In the ideal TDD process every single piece of code-logic contained in the system, is tested as an individual block. This near-hundred percent code coverage would be ideal and insure the quality of the system in many ways, some of them are listed in section6.1. Early in the development process of **C.A.R.S.** the TDD way of writing code was followed, first creating the test and then implementing the code block to make the test run successful. However, due to time constraints, the strict test-first nature of TDD had to be omitted. The time required for programming, surprised the developers and forced this omitting of TDD, if the deadline was to be reached. Unit testing for this project is, for these reasons, restricted to some components of the system.

8.3 Requirements

Fulfilling the user requirements is a crucial part of any software developing. Without fulfilling these, the system could in worst case be useless to the user. **C.A.R.S.** almost fulfils every requirement set by the user. Table 8.1 gives an overview of how and to what extend the requirements has been fulfilled.

Requirement	**Fulfilled by**	**Fulfilled**	**Parcial fulfilled**	**Not fulfilled**
Identify vehicles by registration number	The system enables the user to find vehicles by registration number	X		
Place input information, regarding vehicles, in most used order	The system do not follow the exact same order, when displaying information, as specified by the main user. However the system partly follows this order, but some changes has been made		X	
Identify damage by registration number of the vehicle involved	The system enables the user to find all damages related to a vehicle through a search option	X		
Insert copied text from mail into damage description	The system is design in a way, that allows any information to be edited with copy and paste commands	X		
Place input information, regarding damages, in most used order	The system orders damage information as specified by the user	X		
Information will be stored in the database	All information handle by the system is updated, read and saved in binary files	X		
Show and print graphical presentation	The system gives the user the possibilities to show and print reports, regarding vehicles, in printable pdf format	X		
Export data to excel	The system enables the user to export vehicle information to an excel file	X		
Read from excel containing economic data	The user can choose exported economic files to load into the system	X		
Danish user interface incl. æ, ø, å	All information the user will interact is in Danish	X		
Developed as a windows application	The system is developed as an windows application	X		
High usability	This will be discussed in the following section :-)	X		

Table 8.1: Requirement fulfilment overview

8.4 User evaluations

The participants in the user evaluations in section 7.3, found several usability problems. Some where cosmetic, some serious and one was critical. The first three cosmetic problems in table 7.2 was considered not important and easy to learn in time. Furthermore, these problems were only experienced by few subjects, hence a solution to these problems was not implemented. Problem # 4 and #5 where the subjects wanted to use keyboard short-cuts, like "tab" and "enter", instead of navigating with the mouse had solutions implemented. These were also cosmetic problems, but many of the subjects found it natural to use the keyboard, which became the reason for the implementation. The last of the cosmetic problems: #7, #8 and #9, where the subjects was in doubt if they had performed the tasks, were also corrected by implementing messages telling them, that they had succeeded or not. For #9 an indication of the date-span for the currently loaded data, was created, as Dorte Sørensen requested in the questionnaire. The two serious problems, #10 and #11, was not corrected in the system, but a manual was made to explain the functionality of **C.A.R.S.** . Last but not least, there was a critical problem, problem #6.

In the requirement, from the head of administration, she explained that she wanted damages to be identified by registration number. This turned out, in the evaluation, to be too hard for the participants to remember. The solution that was implemented for this problem, was a drop-down suggestion menu as shown in figure 8.1. This list of registration numbers decreased as the subject typed more of the registration number. This helped the subjects in entering the registration number, as they no longer had to remember it entirely. Another solution and perhaps more fulfilling solution would have been create a button on the *Vehicle Overview* page, named "Show damages of marked vehicle". In this way the user would not have to remember anything just click a button. Likewise have a button on the *Damage Overview* "Show vehicle corresponding to the damage".

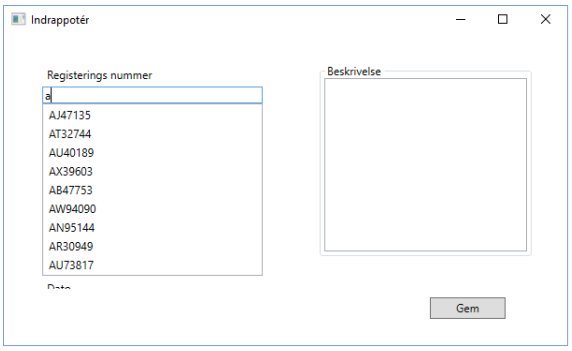


Figure 8.1: Designed solution to a usability problem

Usability lab

A usability lab is a place set up for usability testing, where developers can study users during user evaluations of a system. During this project a usability lab has not been used, due to the distance of the participants. The **C.A.R.S.** system is meant for the user to be sedentary in a office environment. Therefore, the use of an usability lab would have been beneficial to set up similar environments for the participants. To compensate, the usability has been done in small conference rooms at the participants offices. This was perceived as the best possible setup. A less beneficial outcome from this solution was, that some participants got rather nervous due to the fact that three other people were in the room watching. If the usability had been done in a usability lab, this would not have been an issue, as everyone besides the facilitator, would have been looking from the control room, behind a one-way mirror and the participant would not feel their presence.

Conclusion 9

As mentioned in section 8.2 Test Driven Development had to be discharged due to time constraints. This was clearly a good choice for the overall system. If the development process had continued with focus on TDD, the system would have lacked crucial functionality and could not to same extent fulfil the requirements from the user.

OOA&D was a new toolset for developers to work with, use to analyse a problem, and then to develop a system based on the analysis. The result has been a more fluent process of development, because developers did not each have different visions for the solution, but instead clear guidelines from the user. The analysis of the problem domain helped developers to understand the problem and the analysis of the application domain established clear guidelines for the interactions the user would have with the system and which functionality was necessary to meet the requirements.

The initial requirements from the head of administration has been mostly met. The changes made through out the project, has been discussed with, and approved by, the user. A good communication and mutual understanding has been in place, a necessity for this to be possible.

Heuristic analysis was a good way for developers to find flaws in the **C.A.R.S.** system and prevent many errors before presenting the final prototype to the user. As shown in chapter 7.3 heuristic analysis did not find all the flaws in the system. The user evaluations has been an important part of this project, which is shown in the additional amount of usability problems found by subjects compared to developers. If user evaluations had not been conducted, the final system would have had irritation elements and may not have been useful to the head of administration, because of extra workload. Small thing, like being able to press "enter" on the keyboard, made the system more attractive.

The **C.A.R.S.** system meets the requirement of the user and exceeds the

expectations according to simplicity. According to the main user of the system, Dorthe Sørensen, the system will be very useful in assisting her with the workload regarding administration of their fleet. She has mainly expressed optimism about getting to use the system.

Future work 10

The **C.A.R.S.** system has flaws. When the user tries to create a vehicle or edit information about a vehicle, the system will ignore some values if they have not been entered correct. An example is the box where the user will input the monthly payment. This box is only set to register numbers, so if the user enters 1000kr, the input will not be saved without warnings to the user. Furthermore, additional functionality could be implemented in the system, like saving old odometer readings into a list to have historical data preserved.

Another issue for the future work would be to do the optimized solution to the usability problem found in the user evaluations, where the participants could not remember the registration number and had to browse from one page to another to compare a damage to a vehicle. The solution would be to make a connection from a vehicle to its damages and from a damage to the corresponding vehicle information.

Lastly, an important upgrade would be for the system to run on multiple computers and likewise be used by multiple users at the same time.

Part II
Introduction to Academics

In this part of the report the "how" will be answered. How the project has been planed, how different methods has been used and how the cooperation with stakeholders has taken place. This will be answered through the following chapters: development method, cooperation with stakeholders and lastly design of user interface. The Academics will finish up with explanations and illustrations of how the design of the user interface ended with the appearance it did in the final system.

Development method 11

This chapter describes the development methods used during the project. Furthermore, it will describe how the project was planned and how often meetings took place and what they concerned, for the reader to understand the work methods as well.

Throughout the project the group has utilized the system development method *Object Oriented Analysis & Design* (OOA&D). The work method for this project has primarily been the waterfall method. The waterfall method is a process where the development is considered to be constantly flowing downwards through the different phases.

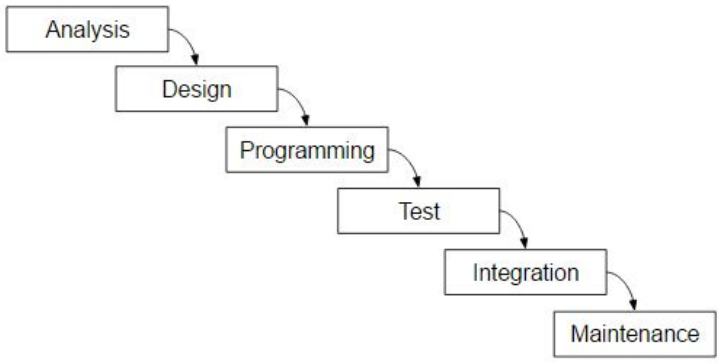


Figure 11.1: Illustation of the waterfall method

Figure 11.1 shows the different phases of the development process. By using the waterfall method, the analysis must be "carved in stone" so it would not be changeable after moving to later phases. The two final phases in 11.1 is not a concern of this project and will not be explained further.

The work done in each step or phase has been iterative in nature. The analysis phase was divided into two activities, which each had a number of sub-activities attached:

- Analysis of the problem domain
- Analysis of the application domain.

Each of the two activities had a number of sub-activities attached as the report states. For the problem domain it was *Structure, Classes and Behaviour*. The sub-activities of the application domain was *Actors, Use cases, Requirement specification, Functions and Interfaces*. When changes were made or new knowledge was gained in the application domain, the model of the problem domain was revised and vice versa.
The next phase according to figure 11.1 was the design. This was likewise divided into different activities:

- Setting up criteria
- The design of architecture
- The design of components

All of these activities were initially done in sequence, as the waterfall method states, but in reality the development became a mix of both the waterfall and iterative methods. When doing the analysis and the design phases, the waterfall method was used. However, in the final stages, the iterative cycle became a part of the remaining two activities for this project: *Programming* and *Tests*. The actual development methods used is illustrated in figure 11.2.

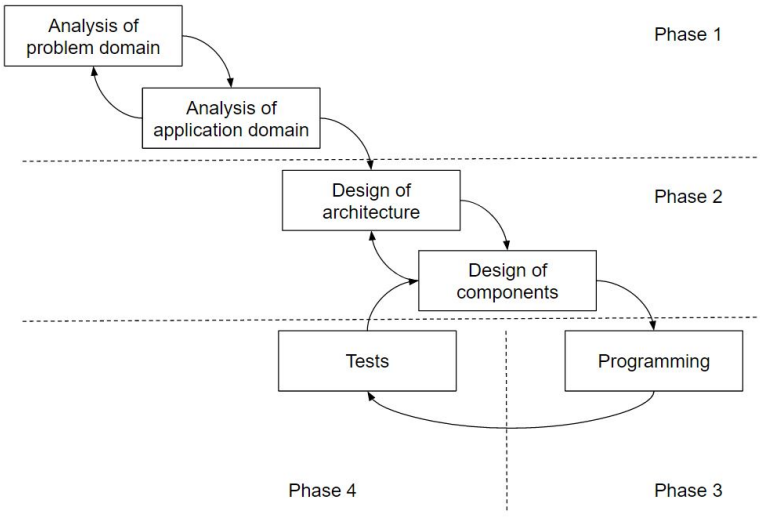


Figure 11.2: The actual development method used during the project

Figure 11.2 illustrates that the two first steps have been completed by the waterfall method in sequence, but that each step individually has been iterative. The final steps has been loosely done by using the iterative method. The reasoning for not choosing a strict waterfall method, was the fact that it would restrict developers from going back to change previous activities.

11.1 Project Scheduling

The service used to make the time schedule for this project, was a simple excel sheet used as a calendar. This was very helpful due to the pedagogical approach with colors. The schedule included lectures, which each had a color and project timelines and deadlines.

Uge 42	12/10	13/10	14/10	15/10	16/10	17/10	18/10
8.15-10.00				G	S		
10.15-12.00				F	S		
12.30-14.15		G	F		S meeting 14.30		
14.30-16.15		F	F				
16.30-18.15							
Note							
Uge 43	19/10	20/10	21/10	22/10	23/10	24/10	25/10
8.15-10.00	Analysis done		Corrections	G	S		
10.15-12.00			to analysis	F	S		
12.30-14.15	F	G	F				
14.30-16.15	G	F	G				
16.30-18.15							
Note							

Figure 11.3: Clip of excel sheet used for planning

Figure 11.3 shows a piece of the schedule from the weeks 42 and 43. The three different colors are each assigned to one of the three courses followed during this semester project. The text in white cells are project related, either deadlines or supervisor meetings. This gave a good overview by also being able to see remaining work needed for each chapter in the report. Homework was assigned on the social media Facebook and so was notions of absence and delays. This media was chosen due to the frequent use by all group members and easy access along with the feature that Facebook gives notifications if someone posted anything on the wall.

11.2 Meetings

In the beginning of the semester the days were filled up with courses. This left little time to work on the project, but nevertheless the group met every day to attend the courses and were able to discuss and apply content to the project. When the courses were done the group continued to meet each day to discuss what had been done and what needed to be done. The daily meetings included discussions of what figures, tables and content needed to be done and was then assigned to group members. Each Friday the schedule was held up against the work of the week and the work that was not done, was assigned to be done by the following Monday. The weekly supervisor meeting, mostly on Tuesdays, worked as an extra follow up according to the plan and the supervisor pointed out features which, at this point, the group had not yet thought about. Due to the different stages of the project not all weeks had one meeting. Some had two and some had non dependent on the state of the work.

Cooperation with stakeholders 12

This chapter describes stakeholders and cooperation with them in the course of the project. Furthermore, a short description of the user-evaluations will be presented for the reader to see when the stakeholder was included in the project.

The stakeholders for this project are people controlling and administrating company vehicles. Throughout this project the stakeholder for cooperation has been the head of administration at Lasse Larsen Byggefirma A/S. There has also been contact with other informants, to research the scope of the problem domain, whether other firms likewise could be interested in such a software system.

Ongoing talks and cooperation with stakeholders clearly had its benefits. The development became more dynamic as she expressed her wishes and developers tried in the best possible way to meet these requirements. The prototype testing then became a test of whether the stakeholder and developers saw eye to eye, and if needed, mistakes could be corrected early in the process.

12.1 Stakeholders

As mentioned the stakeholder in this project is Lasse Larsen Byggefirma A/S, with the main user Dorte Sørensen, the head of administration. She provided the group with requirements for the system and background knowledge along with current data about the company vehicles for the project.

12.2 User-based evaluations

After the analysis a prototype was developed and user-based evaluation was conducted with Dorte Sørensen. The evaluation started with an introduction to the program, thereon after Dorte Sørensen got different tasks to fulfil. The tasks was in decreasing order, so that they started out at low difficulty to make the subject comfortable. The test was done by the think-aloud protocol.

The prototype was only tested by the user, but the finishing user-based evaluation was conducted on six subjects. Due to the limited amount of users at Lasse Larsen Byggefirma A/S subjects was searched for elsewhere. The small firms, where the problem also had been researched in the beginning, together with others who had similar computer experience and workload as the head of administration were involved in the finishing user-based evaluation. The user-based evaluation was conducted in the same way as the prototype, with tasks to fulfil while the subject was asked to think-aloud. Because of the distance from the evaluators to Aalborg, it was decided to do usability in the field, which was the current offices of each subject. There are pros and cons for both doing tests in the lab and in the field. In this case, because the field was an office it would have been most beneficial for the group to do the evaluations in the lab and set up an office environment, but the distance and busy days of the subjects made it more desirable for them, that the usability tests were conducted there.

The subsequent analysis was performed with Instant Data Analysis (IDA), which requires few resources to perform, but still uncovers many usability problems. IDA mainly consists of a brainstorm over problems observed by participants right after the user evaluation has taken place, so IDA allows usability evaluations to be conducted, analysed and documented in a day[12]. The aim in this process is to find the most critical problems. The findings of the usability evaluations are the ones presented in chapter 7.3 under User evaluation.

Design of user interface 13

*This chapter describes the principles of designing the user interface for the **C.A.R.S.** system. Topics to be examined are Human-centred design, Conceptual -and Physical design together with the use of Universal design principles, for the reader to see the range of design principles considered in the project.*

Mitch Kapor once said "*What is design? It's where you stand with a foot in two worlds - the world of technology and the world of people and human purposes - and you try to bring the two together[13].*" Design is a mixture of artistic expression together with engineering skills. A designer must know the "fabric" in which the work is done and know the people for whom it is done equally to get a good result.

13.1 Human-centred design

The parts of a system in which the user comes into contact, both physical, perceptual and conceptual is a user interface.

- Physical contact is the press of a button or other physical influence.
- Perceptual is the things displayed to see or noises to hear for the user.
- Conceptual is to interact with a device by trying to work out what it does and the device will help by providing messages of how to do it.

Since it is the user who is interacting with the system, the human-centred principle is important. The human-centred approach is where the system is designed to support people and it is designed for people to enjoy and not the other way around. When doing a human-centred design, the design will become a safe, effective, ethical and sustainable design[14] and for this, subjects are required, which in this case primary is the head of administration at Lasse Larsen Byggefirma A/S.
There are different important principles when understanding how and where a system is to be used. The human-centred design derives from the best combination of the PACT (People, Activities, Context, Technology) analysis domain. The most essential elements in this project from the PACT analysis will in this section be described.

Mental Model

The mental model is a part of the People element in the PACT analysis, and have been one of the more important principles of this project. It has been essential for developers to understand the mental model of the user, the head of administration, to do the conceptual design of the system. The conceptual design is, among other things, about finding a good conceptualization of a design and getting it communicated to the user.

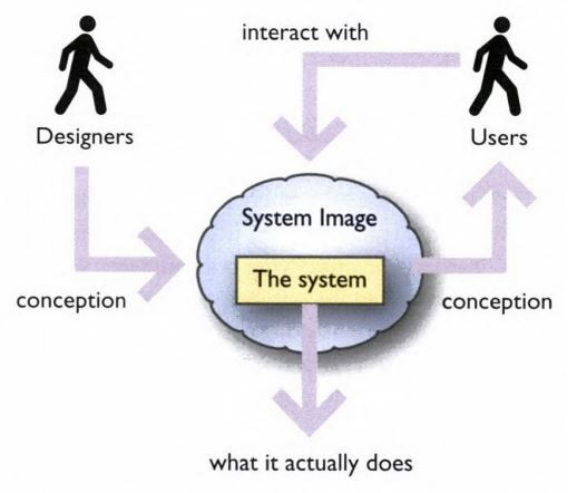


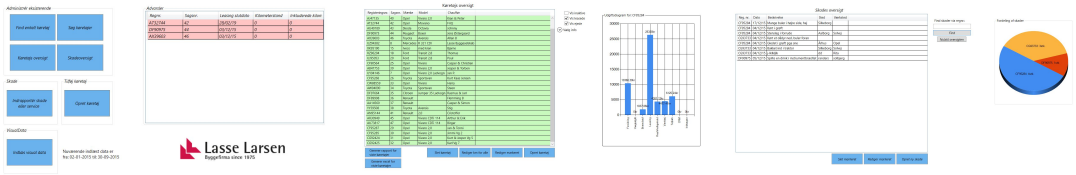
Figure 13.1: Illustration of the Mental model principle [14]

Figure 13.1 shows the principle of the mental model. The designers and the user may both have conceptions of what the system does and how to interact with it. This hopefully corresponds to what the system actually does, if not, a bad mental model exists. When a good mental model is not in place, the user may not be able to recover from situations where something goes wrong. To ensure a good mental model in the **C.A.R.S.** system continuous conversations with the user has taken place and tests of user interface prototypes has been performed. Furthermore, to prevent the user from ending in uncertain situations, message boxes with explaining texts has been implemented.

Temporal aspects and Complexity

Temporal aspects and complexity are both sub-parts of the Activity element in the PACT analysis. There are many aspects in Activity to consider, but those to be highlighted, in this project, has been the two mentioned.
The temporal aspects refers among other things to the regular and infrequent use of functions in the system. The regular use should be easy to do, meanwhile the infrequent tasks should be easy to learn and remember.

Figure 13.2a shows the start page of the **C.A.R.S.** system where frequent tasks are placed and accessible at the press of a button. All tasks together with infrequent tasks regarding vehicles will be placed under Vehicle Overview (Køretøjsoversigt), and likewise will all tasks including infrequent tasks regarding repairs be placed under the Repairments Overview (Skadesoversigt) as illustrated in respectively figure 13.2b and figure 13.2c. By placing all functionality in few places, the user will know where to look when doing infrequent tasks.



(a) Start page (b) Vehicle overview (c) Repairment overview

Figure 13.2: Pages of **C.A.R.S.** system

The complexity is slightly connected to the explained temporal aspect in the design. It has been an important issue for the project to make tasks well-defined. This indicates a step-by-step design where the user is not forced to browse around. To do the well-defined step-by-step tasks it requires a good mental model and knowledge of the system or else the user could do the task as a vague activity where she will find different information in different places and move from one place to another, which is not the intended idea with the button design of the start page.
These two aspects were essential to the physical design of the system, how buttons were placed and how to structure interactions into logical sequences.

13.2 Conceptual vs Physical design

In the previous sections the terms conceptual -and physical design has been shortly mentioned. In this section these will be elaborated with explanations and examples of use through the **C.A.R.S.** system.

Conceptual design

Conceptual design refers to the purpose of the overall system to be developed. There are many ways and techniques to help develop a conceptual design, such as use cases and rich pictures presented in the Assignment and Application domain chapters. The main key is to keep things in an abstract manner, by focusing on the "what" rather than the "how" to create a good mental model.

Physical design

Physical design is concerned with taking the abstraction from the conceptual design and transform it into concrete designs. There are three components to the physical design to consider: Operational, Interaction and Representational design.

Operational design

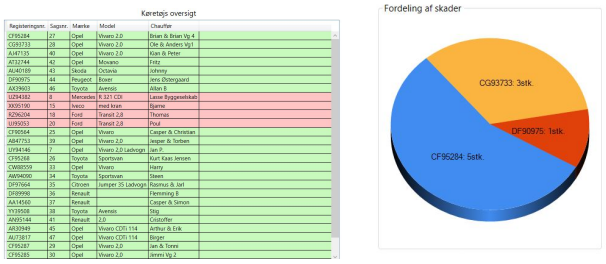
This is where the focus is on how the flow of the system will be and how information is stored and structured. In the **C.A.R.S.** system this is present as a functional view when the warning events are triggered by a underlying function which will make warnings about vehicles appear when they are in a specific span from the end-date of their leasing period and likewise if they are exceeding their maximum mileage.

Interaction design

This is concerned with the allocation of functionality along with the sequencing. As explained and shown, the analysis of the temporal aspects has an important effect on the interactions of the system. The allocation will not be described again but the sequencing of the **C.A.R.S.** system is made by user preferences. It has a simple design with few different pages. An example could be the functionality of viewing vehicle information and editing. Both interactions will lead to the same page: a display of vehicle information with editable boxes.

Representational design

This part of the design is focused on the appearance: the colors, shapes and sizes of buttons in the system along with presentation of information. The graphical presentations in the final system is chosen by the user, so that she has the presentation most informative to her. The button layout is likewise made in cooperation with the user. The calm colors of buttons in blue and gray, gives an appearance which is easy on the eyes, but is still clear on the white background.



(a) Vehicle overview (b) Diagram

Figure 13.3: Illustration for color comparison

In addition to calm colors the red and green colors for illustration of whether a vehicle is active or not has been toned down. The only sharp colors used in the

system are in the pie-chart diagrams for a clear indication of change of vehicle. The difference between sharp and toned down colors can be seen when comparing the red colors in figure 13.3a and figure 13.3b.

(a) Grouped functionality (b) Consistency (c) Consistency

Figure 13.4: Illustrations of User Interface

The grouping of buttons has also been a focus in the system, for the user to easily learn where to look for information. This is shown in figure 13.4a where the functionality regarding existing information is grouped in the four upper boxes. Functionality regarding creating a new vehicle or a new damage is boxed in line three, and finally the last line handles loading of EG Visual data and presenting the user with an overview of dates for which the data has been loaded. Furthermore, an important issue has been the consistency of location. The consistency of location is shown by an example figures 13.4b and 13.4c where the "generate report" button is placed in the same position whether the user is at the Vehicle overview page or the user is at the Search vehicles page. Icons has deliberately been deselected to keep a clean and calm appearance.

13.3 Universal design

The system has mainly been made by the main user's preferences, but the principle of a universal design has been applied, to secure future users of the system also will be able to incorporate the design. The principles of the universal design are as follows[14]:

- Equitable use
- Flexibility in use
- Simple, intuitive use
- Perceptible information
- Tolerance for error
- Low physical effort
- Size and space for approach and use

The only requirement made by the client, which is a contrary to the listed points, are the language specification. The point *Simple, intuitive use* refers among other

things to the language. The designers has kept dialogues in a simple language, but it is still essential to understand the Danish language to use the **C.A.R.S.** system, because icons has been deselected.

Bibliography

[1] Danmarks statestik. nyregistrerede-og-brugte-biler. `https://www.dst.dk/da/Statistik/emner/biler` (18/10/2015).

[2] Dorte Soerensen. The head of administration at Lasse Larsen A/S.

[3] Peter Axel Nielsen Jan Stage Lars Mathiassen, Andreas Munk-Madsen. *Objekt Orienteret Analyse & Design*. Marko ApS, Aalborg, 2001.

[4] EG. Styr på dit regnskab. `http://eg.dk/brancher/bygge-anlaeg/haandvaerkere/regnskab` (12/12/2015).

[5] Software GmpH empira. MigraDoc Overview. `www.pdfsharp.net/migradocoverview.ashx` (15/11/2015).

[6] Mike Gledhill. Create an excel file. `http://mikesknowledgebase.azurewebsites.net/pages/Home/index.htm` (18/11/2015).

[7] Introducing Windows Presentation Foundation. Msdn. `https://msdn.microsoft.com/en-us/library/aa663364.aspx` (10/12/2015).

[8] Roy Osherove. *The Art of Unit testing*. Manning publications co, Sound View Court 3B, Greenwich, CT 06830, 2009.

[9] MSDN. Unit test basics. `https://msdn.microsoft.com/en-us/library/hh694602.aspx` (15/11/2015).

[10] Software Testing Fundamentals. Blackbox testing. `http://softwaretestingfundamentals.com/black-box-testing/` (15/11/2015).

[11] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1994.

[12] Jan Stage Jesper Kjeldskov, Mikael B. Skov. *Instant Data Analysis: Conducting Usability Evaluations in a Day.* Association for Computing Machinery, Department of Computer Science, Aalborg University, 2004.

[13] Mitchell Kapor. A software design manifesto. *Dr. Dobb's J.*, 16(1):62–67, November 1990.

[14] David Benyon. *Designing Interactive Systems.* Pearson, Edinburgh Gate Harlow CM20 2JE United Kingdom, 3rd edition, 2014.

Navigation diagram A

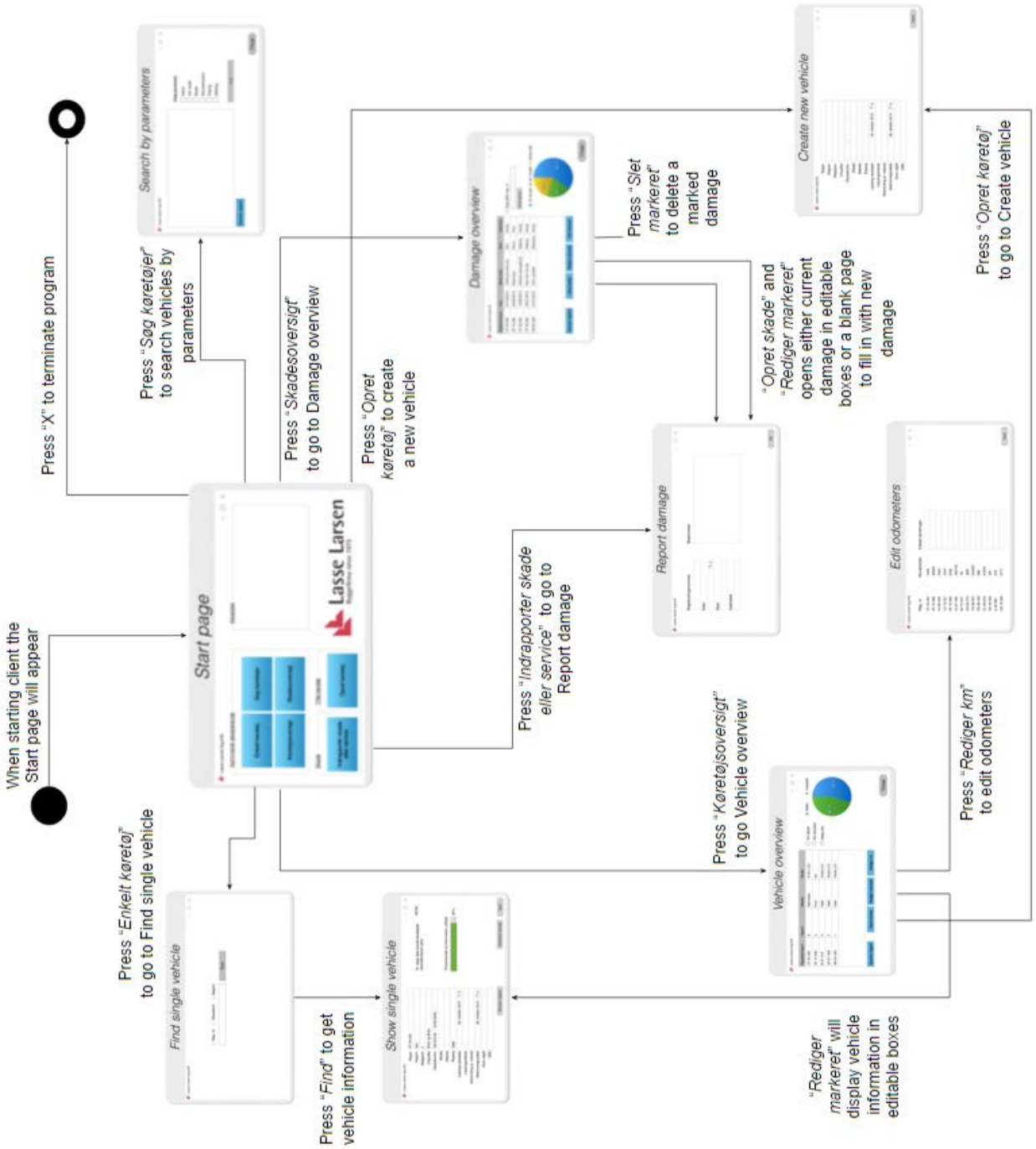


Figure A.1: Full navigation diagram of the user interface in chapter 4

Interview 1 B

Interview conducted at Lasse Larsen Byggefirma A/S 20/10/2015. The interviewee was the economy and administration director Dorte Sørensen.

Søren: The purpose today is to get an overview of how you handle the data today, and then potentially get some pointers as to what elements you would like to see in the final solution. For instance like we talked about earlier, it might be the ability to pull data on a vehicle showing its cost per day throughout its lifetime. So you would be able to see whether one model is better than another.

Dorte: Well if I start by telling what we have now, and where we keep it located, and what I think could be interesting to have.
We lease some of our vehicles and we buy some of our vehicles, the ones we lease are leased through 2 different companies , and when we lease a vehicle there is a lot of master data accompanying the vehicle. We access that data by logging on to the specific leasing company's website. Its information regarding age, fuel type, how many kilometres we have reported for the vehicle, what services have been performed on the vehicle. Not what the services have cost us. So if we have the leased car that is like one box where we have some data.

Søren: And the leased vehicles made up around half of your vehicles right?

Dorte: Ehh its actually less, we have just bought 4 of the vehicles we have been leasing, so it is closer to a third that we lease. And it is with 2 different leasing companies, which doesn't make it (the data) any more accessible. Ehm, then we have our economy system, it is called "Visual", ehm this is where we have all the costs regarding vehicles gathered, but since the economy system is project based we have each vehicle as a separate project. This means that when I get data in and when I get data out it is for one vehicle only. What we have in these projects are the bills that regard vehicles, so everything from service costs to fuel expenses, purchase price

and road tax. Then I have an excel file containing a list of all the vehicles containing master data, who is driving the vehicle, how old is it, how many kilometres have it travelled. So this is the one I use mostly, because this is where I can group the information I need about each vehicle, but it is input manually.

Søren: That is a lot of work to maintain then.

Dorte: Well I can export data from Visual in a excel format and the copy what I need over, but I don't use it that much, I usually just type in the data I need. And that is how we do it today.
One thing I would like, is to have access to all the data from just one location, and excel would suit me fine because I am good at working with it, and it is the application i use most. But how you choose to display the information is of course up to you, you might come up with an even better way, I just like excel because it is so easy to get an overview and most importantly I can calculate things in it once I have the data put in.

Søren: So would your primary need be to get a detailed view of an individual vehicle, or to be able to do calculations with a list of vehicles at once?

Dorte: Ehh... well its depends on the task I am performing, if for instance I am trying to decide whether to keep repairing on a vehicle or to buy a new one, I would be looking on a specific vehicle, recently we were looking into 5 of the leased vehicles that had expended their allotted kilometres, in that case I had to inspect the 5 cars together to get a overview. So it really depends on the situation, both would definitely have its uses.
For me it would be nice to have all the master data gathered in one place so I don't have to go searching for it in different systems. Also it would be a great help if I could easily access information like "how much has this vehicle cost in the last year", the economy system can do a part of it, but since it does not contain master data i have no way of knowing how old the current vehicle is for instance, it does not have information about write offs on the individual vehicle since we do that combined for all our vehicles, so in order to figure out what a vehicle is worth I have to find information regarding the period of time we have had the vehicle, and then search the master data to find the amount we are deducting from its value each year.

Søren: How do you collect data from the odometers?

Dorte: Once each year I will ask our people to report back their vehicle's odometer value, we have tried having them report it each time they refuelled but to be honest they just couldn't be bothered to do it.

Søren: As I understand, the reason you lease some and own some of your vehicles is to spread the risk, what benefits are there to leasing?

Dorte: Price, if you buy you have to put up the entire price of the vehicle, furthermore, when you lease there is a service package included so that you won't get any unforeseen expenses regarding that vehicle.
As the rate of interest has gone down it, it is becoming more and more a good idea to purchase instead, since the savings on service agreements offset the money spent on loans.

Søren: How do you keep track of when to service vehicles?

Dorte: Well it is a fixed interval, since we can't keep accurate track of odometers in our vehicles.

Søren: How about M.O.T. tests?

Dorte: We get a mail in our "E-Boks" every two years with a date the test has to be done by, and then we look at the vehicle to see if it is worth keeping the vehicle, or if it would make better sense economically to replace it.

Søren: The vehicles you own, do you have a service package for them?

Dorte: No

Søren: So they get serviced and repaired only as needed?

Dorte: Yes it depends on the vehicle, whether we use a authorised repair shop or not, if the vehicle is getting old we have a place we take them in Solbjerg that keeps them running.

Søren: How about roadside assistance? Do you have any subscriptions?

Dorte: No

Søren: Knowing that damages will occur, how do you handle when a vehicle gets damaged?

Dorte: Uhm.. that would be a really interesting event to register, we really are not that good at it now, that information should be somewhere, but we don't have it today. For the leased vehicles the repairs are part of the leasing agreement, we of course have to make a damage report, there are two paths, one is the repair the other is the insurance where we have to submit the report. This is because we get as much as possible fixed through the insurance, assuming the cost is less than our deductibles, for instance a stone crack would be less than the 6000 kr. that we have as deductibles so that would just be repaired at our own cost.

Søren: So you don't have a damage history for each vehicle?

Dorte: No, but that is something that would be nice to have.

Søren: Is it the same person that is responsible for a vehicle for its entire lifetime?

Dorte: Yes, it is actually hard for us to swap vehicles between people, because they feel that they each take better care of their vehicle than other people do. We have tried to swap if one vehicle is getting close to the allotted number of kilometres in the leasing agreement, but as I said, it is very hard.

Søren: Okay, having branches both here (Jutland) and on Zealand, are the vehicles handled centrally or in each branch?

Dorte: I handle all the vehicles, otherwise they (the other branch) would need their own administrative personnel, so it is more cost efficient this way.

Søren: Yes, so costs would increase by quite a lot.

Dorte: Yes, the way we do it is really the best way.

Søren: Okay, well I think that was what we had for now. You have covered most questions already.

Søren: If you could maybe try telling me what information you use most regarding the vehicles?

Dorte: Well I typically group the vehicles for instance by leased or not leased, managers or craftsmen. That is because they have different driving needs, so it is groups like that I will use, other than that it is the number of driven kilometres and the total cost of the vehicle I look at.

Søren: Okay..

Dorte: And of course if I had a way to see a damage history that would be great.

Søren: Yes, that would be a good thing to have access to... Well this definitely gives us something to start working with, we will email you about things that need clarification and if questions come up, the next time we meet, we should have a prototype ready for you to try out and give some feedback on.

Dorte: That is fine with me

List of input information C

This is three lists given by the head of administration. The lists represents in which order the user wants the input information to be placed.

Owned vehicle

1 Registration number
2 Case number
3 Folder number
4 Make
5 Model
6 Year
7 Affiliation
8 Driver
9 Diesel-cards (up to three)
10 Purchase date
11 Price
12 Depreciation per. month
13 Depreciation end date
14 Mileage

Leased vehicle

1 Registration number
2 Case number
3 Folder number
4 Make
5 Model
6 Year
7 Affiliation
8 Driver
9 Diesel-cards (up to three)

10 Leasing company
11 Leasing start date
12 Payment per. month
13 Payment end date
14 Mileage
15 Max mileage

Damage

1 Registration number
2 Date
3 Place
4 Workshop
5 Description

Tasks in Danish D

Indledende opgave : Indlæsning af Visual data

Du skal have indlæst data fra visual project til programmet.

a Åben programmet og indlæs data

Opgave 1 : Skade

Det er d. 17 december. Du er mødt ind på arbejde trods den store mængde sne på vejene. Brian, som kører i CF 95 284, kørte i går i grøften pga. islag i et sving på vejen hjem fra et projekt ved Silkeborg. Brian rapporterer at bilen har betydelige buler langs højre side, men han er okay.

a Indrapporter skaden til systemet, så den er gemt i databasen.

Omfanget af skaden var mere omfattende end Brians forklaring. Værkstedet rapporterer at der yderligere er skade på hjullejet.

b Rediger skadens beskrivelse, så den svarer til værkstedets forklaring.

Det er nu d. 20. december og du kan ikke huske skadens omfang præcist, da dit hoved er fyldt op med juleforberedelse.

c Hvordan vil du finde information omkring en skade?

Opgave 2 : Køretøj

Der er netop blevet skrevet under på en leasingkontrakt. Du vil gerne oprette køretøjet i systemet med det samme. De oplysninger du har indtil videre er:
Reg. nr: AF 75 998
Leasing firma : FLEET
Model : Fiesta
Mærke : Ford
Leasing startdato : 01/02/2016
Afskrivning pr. måned : 2798,-

Forventet afslutning : 01/02/2021
Max kilometer : 40.000km

a Hvordan vil du fortsætte for at oprette køretøjet i systemet?

Du har nu fået yderligere information om køretøj AF 75 998
Chauffør : Sigurd
Dieselkort : 875 398 090 fra Statoil og 908826 87 fra Q8

b Hvordan vil du tilføje information til køretøjet?

Sigurd, som er fører af køretøj AF 75 998 er gået på pension og køretøjet videregives derfor til Poul

c Hvordan vil du ændre chauffør?

Opgave 3 : Rapport

Du skal til møde med Brian og ønsker derfor at kunne diskutere de mange skader han har påført sit køretøj CF 95 284 .

a Hvordan vil du finde alle Brians skader?

Du har nu talt med Brian og skal videre til den næste i rækken, Ole.

b Hvordan vil du finde Oles skader?

Opgave 4 : Overblik

Det er dagen efter du har haft møde med Brian og Ole og du har siddet hele dagen og kigget rapporter igennem omkring køretøjer og økonomi. Du sidder lige nu inde under "Skadesoversigt" (Tryk på "Skadesoversigt"), da telefonen ringer. Det er Brian der igen har lavet en skade på sit køretøj CF 95 284, da han er bakket ind i et træ. Brian rapporterer at han er på vej til Solbjergs værksted med køretøjet.

a Hvordan vil du indrapportere skaden?

5 minutter efter møder Brian og Brian op med kage og fortæller at det var en spøg og at de ikke har lavet nogen skade på bilen.

b Hvad gør du?

Opgave 5 : km aflæsning

Det er nu den tid på året hvor du har sendt en besked ud til alle chaufførerne om, at de skal tilbage melde deres køretøjs kilometerstand. Dine svar er som følger : CF 95 284 : 52002 CG 93 733 : 48462 AJ 47 135 : 9009

a Hvordan vil du registrere dette i systemet?

Der er et køretøj der har været lidt sløv på aftrækkeren og du rykker derfor efter svar, som du også får efter kort tid. Køretøj UZ 94 382 har kilometerstand på 10087

b Hvordan vil du tilføje den forsinkede kilometerstand?
c Kan du gøre det på andre måder?

Opgave 6 : Skadesomkostninger

Du vil gerne finde de samlede reperationsomkrostninger for CF 95 284. Du vil gerne bruge dette til at se om hvorvidt det kan svare sig at købe et nyt køretøj i stedet for.

a Hvor vil du finde disse oplysninger?

Du har fundet ud af det ikke kan svare sig at beholde køretøjet og at det er bedre at købe et nyt. Så du vil nu gerne deaktivere det nuværende køretøj.

b Hvordan vil du fortsætte for at deaktivere køretøjet?

Opgave 7 : Fra leaset til ejet

Et af jeres køretøjer er meget tæt på det maksimale antal km i forhold til leasingkontrakten. I har i dette tilfælde valgt at købe dette køretøj, da dette bedre kan svare sig.

a Hvordan vil du ændre køretøjet fra at været leaset til ejet?

Tasks translated into English

E

Assignment 1 : Damage

It is December the 17, you just meet in at work despite the large amount of snow on the roads. Brian that drives in the vehicle with the registration number CF 95 284, drove off the road yesterday because of the icy roads on his way home from a job in Silkeborg. Brian reports that he is okay, but the vehicle obtained several dents along the right side of the vehicle.

a Save report of the damage obtained on the vehicle in the system.

The damage was more comprehensive than Brian had told you yesterday. The repair shop reports that the vehicle's wheel bearing has been damaged.

b Edit the damage description, so it matches the report for the repair shop.

It is now December the 20. and you're busy preparing for Christmas. Unfortunately you have forgotten the precise extent of the damage.

c How would you find the information about the damage in the system?

Assignment 2 : Vehicle

A leasing contract have just been signed. You would like to create the vehicle in the system right away; the information available at the moment is following:
Reg. No.: AF 75 998
Leasing Company: Fleet
Model: Fiesta
Make: Ford
Leasing start date: 01/02/2016
Depreciation pr. month: 2798,-
Expected end date: 01/02/2021
Max km: 40.000km

a You will have to create the vehicle in the system, how would you proceed?

You have just gained a new set of information for the same vehicle with reg.no. AF 75 998.
Driver: Sigurd
Fuel card: 875 398 090 from Statoil and 908826 87 from Q8.

b How would you add the new information to the vehicle?

Sigurd, the driver of the vehicle with registration number AF 75 998 are retiring, and the vehicle are passed to Poul.

c How would change the driver for the specific vehicle?

Assignment 3 : Report

You are having a meeting with Brian about the many damages he has caused on his vehicle CF 95 284.

a How would you locate all the damages Brian has caused on his vehicle?

The meeting is over with Brian, and you are now having a meeting with Ole.

b How would you locate all the damages Ole has caused on his vehicle?

Assignment 4 : Overview

Yesterday you had the meetings with Brian and Ole and you have spent the whole day looking through reports about vehicles and economy. Right now, you are looking under "Skadesoversigt" (press the "Skadesoversigt" button) when Brian calls you. He reversed into a tree and made another damage to his vehicle CF 95 284. Brian reports that he is on his way to Solbjergs repair shop with the vehicle.

a How would you report the damage in the system?

Five minutes later Brian and Brian shows up with cake laughing, telling you that it was a prank all along and they had not caused any injuries to the vehicle.

b What do you do?

Assignment 5 : Odometer reading

You have asked all drivers to report the odometer reading of their vehicles. The drivers responses are as followed:
CF 95 284: 52002
CG 93 733: 48462
AJ 47 135: 9009

a How will you register these readings into the system?

One driver has reported back later than the others, and you ask him once again. Shortly after you get the reading you wanted, vehicle UZ 94 382 has following reading: UZ 94 382: 10087.

b How will you register this into the system?
c Can this be done in any other way?

Assignment 6 : Repairment cost

You would like to know the overall cost of the vehicle with the registration number CF 95 284, to evaluate weather it is worth repairing the vehicle or replacing it instead.

a How will you find these information?

You have decided that the vehicle is not worth keeping and replacing it is the only option. Now you would like to deactivate the vehicle.

b How will you continue to deactivate the vehicle?

Assignment 7 : From leased to owned

One of your vehicles is close to the maximum allow mileage according to the lease contract. You have chosen to buy this vehicle instead of extending the lease contract, of economic reasons.

a How will you change this vehicle from being leased to owned?