Content

Appendices that support the project

Appendix

Appendix - JavaDoc

Appendix - Code

Appendix - Video[1]

[1] The video appendix can be found on the DVD or on Youtube - http://www.youtube.com/user/TeanTalc

Preface – Nyvang & Lars

The project you are about to read is divided into 2 parts. The parts aren´t visually separated and are therefore written like one single project; however they differ in their specific goals.

The School-part of this project will focus on the use of UML and the Unified Process (UP), spiced up with some object oriented programming in the Java language. Throughout the project we will touch some of the most essential parts in the UP and we will implement some of the newest technologies within the Java language. The overall goal is to master the UP, UML and Java on a higher level than we do at this point. We hope, that our curiosity towards exploring new features and trying to implement these in our project will be a good approach to learn as much as possible - guess only time will tell…

The company-part is normally to be a shell for the learning project, so that we have a "real-life" situation to work with, the same applies to this project. However we have agreed that if we can make a full system, the company is to use our system in "real-life". This will of course set our relatively small designing /programming skills on a great test, as we only have attended this education for 1 semester (at the start of the project).

As this project is an exam's project, the primary recipient group is of cause our teachers and the respective sensor. If others may find this project useful; it could be other computer science students, they could use it for inspiration or a different point of view in relation to their education and/or project.

In the report you´ll find a lot of diagrams according to the UML standard, some of these however, are very big and can therefore be difficult to read[2]. We are sorry for this, but yet we hope that our documentation will give you, as a reader, an insight in some of the choices we have made upon our discussions during the project.

[2] All diagrams can be viewed electronically on the attached

1.0 Project establishment

1.1 Team Members in TeaN Talc

TeaN TALC[3] consists of 5 members

Torben 'Tvupper' Hansen
Tobaksvejen 29 1. tv
2860 Søborg
Phone: 20 15 46 63
E-mail: tvupper@gmail.com

Adam Mukiibi
Rebæk Søpark 5 - 432
2650 Hvidovre
Phone: 27 44 66 58
Email: Katngire@hotmail.com

Lars 'NegoZiatoR' Schou
Knolden 164A
4000 Roskilde
Phone: 22 64 87 14
Email: Lars.sc@gmail.com

Claus Beck
Terslev Bygade 13
4690 Haslev
Phone: 31 24 87 20
Email: beck@welovefailing.com

Nicolaj Nyvang
Møllehusene 8 2nd Right
4000 Roskilde
Phone: 40 14 25 55
Email: nicolajnyvang@gmail.com

[3] The capital letters is the first character in our first names.

1.1.1 TeaN Talc Pros and Cons

In this project, different tasks is assigned to different team members and the weaknesses and strength of different members, as shown below are of great importance in doing this. It is because it helps the team to find out the kind of resources it has, this helps in the distribution of tasks.

For example some people are good in programming, some in designing, and so on and we have of course some weaknesses as shown in the table below. Though we have some weaknesses, we are a team so this will not hinder the progress of the project.

Claus

Pro:	**Cons:**
Java Programming UML Overview (in programming and planning) Open Workbench (planning) Knowing KJ Motorcycles inside out WILLING TO MAKE COFFEE – Added by Nyvang :)	Databases GUI design(what it should look like, not programming) Lack of patience (please kick me sometimes ☺)

Nicolaj

Pro:	**Cons:**
Writing reports Project management Finalizing and layout Programming Patience **Love beer and baking cake for the group**	UML Open Workbench – don't know it @ all The grand overview (programming) Will often try solutions that are too complicated…

Lars

Pro:	**Cons:**
Spend much time to Understand things to the end have the desire and courage to do this project to its end Have a good grip on concepts in both UML and Programming	Trying to learn too many new things at once find it difficult to survey a larger project with full consistency Know nothing about databases and their revivals Can make a small problem into a huge problem.

Adams

Pro:	**Cons:**
I like working in a team I always like socializing English is not a problem to me and the project is in English I always look forward to accomplish my task	Not so good in programming Have a lot of things to do in a week I do not have any of my group members living near me

Torben (TVUP)

Pro:	**Cons:**
A good overview of code, able to see systems in complex systems. Basic understanding of C++, Visual Basic, PHP & ASP Have a lot of experience in SQL-databases. Have worked with ACCESS-databases.	Can get frustrated when I'm not able to spot compile-errors. Often forgets simple syntax in JAVA UML… Forgets to comment on code Open Workbench Windows

1.1.2 Roles -

We have agreed that all team members must take the different roles[4] (Analyst, Designer, Implementer, Integrator, tester, Change Control Manager, Configuration Manager, Project Manager, Reviewer, Any other, and Stakeholder) throughout the different iterations of the project. We talked about handing out the roles, but it didn't make sense that one team member didn't get to use all the roles in this project which primarily objective is learning. The only role that will rotate among the team members is the project management role. This is chosen because every project should have a manager who is on top of the risks, project planning etc. Besides this, the role is somewhat distant for most of the team members and therefore there will be a great learning potential.

1.1.3 Methodology - Nyvang

We are, of course, using different theory throughout this project. Mainly, the project is built around the Unified Process, which is described at UPEDU.ORG (United Process Education). To focus on the personal learning, we have agreed that different team members must code and draw the diagrams. In short, no team member may code his own diagrams. By doing this, we are making sure that the code will reflect our design. Fx. If the same person is both drawing and coding, we cannot make sure that the diagrams are actually used in the right way, because this person will have a general idea of how the code should look like and therefore there is a possibility that the diagrams and the code doesn't match.

As we define in the problem definition, we have chosen to follow the UP in the project. However some places we might agree to do things in a slightly different way.

[4] Source www.upedu.org

1.1.3.1 - Xtreme programming/agile development - Lars, Claus, Nyvang

As we attended a course in agile development/xtreme programming, we are thinking of the pros and cons of trying the theory on a use case in our project.
Very short, the technique is that we split the use case in as many small user stories as possible. Each story must present a value for the company (visually, like GUI features they can see and evaluate) and will be created in different sprints. These are rapidly released to review for the company and feedback for us. The technique can be related to recursion in programming, where you split the problem up in more simple problems that are easier to deal with.

The pros of this technique (as we see it) are

- Fast feedback – we get feedback at each release, this could be several times a week. By doing this we can correct errors "on the fly" and we always know if we are on track or if any features are missing.
- Quick closure – each user story will be done in "n"-amount of hours/days and therefore we have a great overview of the backlog. When a story is released, and feedback and/or errors are corrected, the part is done, implemented, and tested.
- Fast error correction – when errors are discovered at a early point, the possible impact on the next story/use case etc. are minimized, and are therefore cheaper (time-wise in this school project) than discovering them when you are further in the process.

The cons of this technique (again, as we see it)

- A very large number of very small iterations, which often makes you focus only on the specific functions of the user story on which you are working. Then it's easy to oversee if the functions will work in the bigger picture.
- As this is a new technique for all of us, it´s possible that we cannot use it right and therefore the pros might turn into cons.

When we hold the pros and cons up against each other, we see that the pros clearly exceed the cons. Therefore we decide to try the technique in a single iteration, so that we get an idea about which techniques/activities might be better for what.

Another process we are using in the project is from the extreme programming event, this is called pair-programming, and is based on the old saying "Two heads are better than one".

1.1.3.2 – SCRUM – Lars, Nyvang

A few of us were at a seminar about game production, where we heard of the SCRUM technique. This technique seemed also like something we could use. And we decided to try it. Each of us will use SCRUM in a light edition, which mean that every morning, we ask ourselves: What have I done, what should I do, and what is keeping me from doing it. This way we can optimize our time because we minimize the things that are keeping us from moving on.

1.2 TeaN TALC Agreements -

To help maintain focus and consistency throughout the project, we have agreed on some "rules" that will serve as a basic common foundation for the teamwork. In addition, the rules should help all team members to know what is expected of them, and what they can expect from the others. In case of disagreement in such a degree where the only solution is that the group will split up, the agreement should help defining the copyrights on the already completed iterations.

The rules, in random order, are as follows:

- Meeting well prepared –*this includes reading the theory and doing the assignments. All team members are of course trusted to take care of their individual learning, and inform the team if he/she needs help on a chapter in the book etc.*
- Meeting every day – *if a team member for any reason are late or absent the rest of the team must be informed before 08.30. This could be done by text message, e-mail, or phone etc.*
- Work 8:20-15, ready for "overtime" –*the normal school hours is from 08:20 to 13:00, the remaining hours is primarily used for the project. If* for *some reason the team needs to finish fx. Iteration or other work. The team can agree on extending the hours however this must be proclaimed <u>at least</u> the day before. The other way around, a team member that cannot attend the "overtime" should inform the team <u>no later</u> than the day before.*
- Would it / believe it / could it – *the team motto "if u really want it, and believe it, you can do it"*
- Be a team player – *each team members is obligated in participating in discussions and plain generally work while the team is together. In addition, the teams members are committed to listen to all suggestion, speak their mind, and be honest about critics (good or bad).*
- Like the coffee and cake and a trip to George - *The teams members are obligated to participate[5] in the social arrangements to help achieve the "jelled team spirit" as described in the team members section.*
- Team disbands– *if the team for some reason disband, the already written material must be copied and shared so that all members can continue working on the project.*

1.3 Company Description -

KJ Motorcycles is a repair shop that is specialized in tuning motorcycles of any type. They have a niche business, that is, they don't really have any competition in the Danish market.

The workshop has 2 mechanics and sometimes a part time helper or 2. The 2 mechanics have different roles in the workshop. To explain this we need to mention the different stations in the shop.

When a bike arrives to the shop, its performance is measured in a big device called a Dyno – this is station 1.
After this the bike goes to the workshop where the parts are separated from the bike. This is because the engine is to be moved to the next room, but also because some parts are permanently removed or replaced with lighter and stronger parts. The weight is also an issue when tuning a bike. This is the 1st workshop visit (station 2).
Then the separated engine is moved to the engine room, where it's being tuned after the customers' demands. Here the measurement of the parts are recorded on a whiteboard (see Figure 1 1) and digitalized by taking a photo of the whiteboard (station 3).
The next phase is reassembling the bike in the workshop, for 2nd visit so it's ready for another performance measurement in the Dyno. When the final measurement is done, the result is ready for the mechanics and the customer. Although it is rare that performance is not increased, there are no guarantees when tuning a bike, so the measurements are of great importance to document the new

[5] Basically the team members should strive to participate in all the arrangements. However all team members have a personally life besides the school and project which of course most of the time must come first.

performance of the bike. In some cases, the bike is moved to another location to optimize the suspension so the bike has the best grip on the road as possible (station 4).

This is the main workflow of KJ Motorcycles, we have the following steps:

1. Dyno (1st visit) for measurement before the tuning
2. Workshop (1st visit) for separating the specific parts from the chassis
3. Engine room, tuning the engine
4. Workshop (2nd visit) reassembling the bike
5. Dyno (2nd visit) measurement after the tuning
6. Suspension Workshop, increasing the road grip for the bike

1.3.1 Problem Description -

We now have an overview of the process and we can draw a better picture of the problem.
There is no system to record the different values from the measurements therefore there are no history on the bikes that revisit the shop. Another issue is the mechanics contra the different stages. The first mechanic, Michael, is handling all tuning in the engine room and Claus is handling the measurement in the Dyno. When the corresponded measurements are recorded differently by both Michael and Claus – it's difficult for e.g. Claus to get an overview of Michael´s work in case of absence.

1.3.2 Problem Definition -

We have discussed the problem in our team.
The company now uses a whiteboard and pictures of the whiteboard for recording measurements. See figure 1 1.
It´s hard for the company to organize their photos and find one particular.
The company wants to tell the owner a story of the bike.
We want to hand in a functional system to the company, but we also want to learn the principles and concepts inside the UP, Java programming and relational databases.

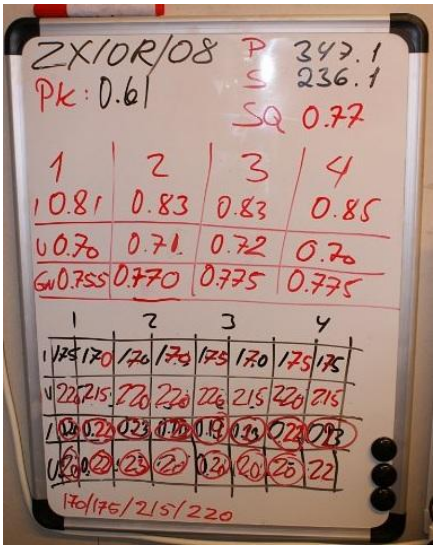


Figure 1 1 - Whiteboard, used to store information

We have decided to focus and prioritize these problems in the project.
"How can UP, Java programming and Relational Databases be used for development and implementation of a minor single user IT-System?[6]

How can we establish a jelled team, so we can overcome the great challenge we have given ourselves?

How can we focus on personal learning while planning and doing the project?
How can we use new java technologies (which we haven't learned in school) to improve our system with some cool and useful features?
How can we organize our layers and classes, so we have highest cohesion and lowest coupling?
How can we hold the connecting line throughout the project?

[6] The definition is taken directly from the project charter

1.4 Project Scope -

Basically we are aiming to extend the project throughout the transition face. Our team arguments are that we wanted to make a full project/system. We have several reasons for this, but the main arguments were that we will learn more by going through all the faces (inception, elaboration, construction, and transition). Another argument is that if we aim high, we might tend to be more focused from the beginning, because we know that the time is of the essence from day one.

When we know that we have a deadline, we thought it would be a good idea to reconsider our goals, project-planning etc. Therefore we will reconsider our project at every milestone. This is to maintain our primary goal, which is learning, for instance if we throughout the elaboration phase didn't achieved the learning goal, we can choose to extend the phase (or repeat some of the disciplines) to achieve the goals.

1.5 Shared Vision -

We paid the company KJ Motorcycles a visit in Herlev where we was shown around seeing all their workstations and got a good description of what they are doing. During the visit we took pictures and recorded videos for documentation, it can be seen here http://welovefailing.com/projects.htm

Regarding to the information we got, we have made the following description on the needs and features for the system.

1.5.1 Needs -

The company needs a smoother way to store and retrieve data about vehicles.
The needs are to quickly retrieve old data, update with new data or create new which also can be easily retrieved and updated.
Another need is to compare old data with new, which provides a picture of the work done in the shop. It should be easier to get an overview of the data. This data is relevant for the company as well as their customers.
The company wants to optimize their workflow to spend less time searching for old data and making reports.

In other words: KJ Motorcycles needs a system that can store data about the work they have done on the bikes.
The data is used to show a history about the bikes performance and rebuilds. Therefore it's important to store the data from both before and after the visit to the workshop, and to print a report to document their work for both themselves and not at least the customer. This way they will get a better overview of the different orders, and give them a better and safer way to store the important data of the vehicles.

1.5.2 Features -

From the visit and the brainstorm of the system, we thought of the following features.

- Vehicle - CRUD
- Owner - CRUD
- Create Whiteboard
- Manage Whiteboard
- Search Owner
- Upload Media
- Make Report
- Search Vehicle
- Register Order
- Register User
- Report Errors

It should be possible to do all these features on each of the following stations (Dyno, Workshop, Engine, and Suspension).

1.6 SW-Requirements -

We have discussed different software tools and workspaces for working with this project. We discussed how to handle that all software should be cross-platform because not all in the team are using Microsoft Windows. Another issue is the router-limitations at the school, which prevents us from using software which require access through other than standard ports on the network.

Another general requirement is that we couldn't buy expensive software so open-source, freeware, and community editions would be preferred.

1.6.1 Workspace

Our team needs to be able to share the report, media, and artifacts quickly and easily. The following methods are chosen based upon the following reasons:

- DropBox (For documents , pictures etc.)
- DropBox is a shared folder which synchronize automatically on each client when an internet connection becomes available
- It's easy to set up
- Light-weight

- SVN (For code)
- It's a protocol for subversion control. Basically it's a shared folder accessed through http:// (or svn:// which is a virtual protocol) with a log attached
- It's integrated into NetBeans although with limitations.
- Software can be downloaded for a stand-alone client for easier version control
- Every change in the code can be reviewed. It has the ability to write a difference-report which tells differences between two versions, line by line, file by file.
- Every change can be reverted
- Two members working on the same file at the same time by mistake can easily be sorted out

1.6.2 Tools & Software - Tvupper

- NetBeans 6.9.1 (JAVA Programming)

- This software is an IDE for JAVA-programming
- It was obvious to use this tool since it's a part of the education which our team is attending.
- Some remarked that it's a bit too intelligent making the members lazy in their work. We can't rule out the possibility of someone using more light-weight editors, but the code should always be viewable in NetBeans.

- **Derby (database)**

- Integrated database system in NetBeans
- As we are taught in Derby, this is the natural choice for our project.

- **MySQL (database – on-line)**

- Online database
- As we have issues connecting to Derby, when hosting the DB on our webserver, through the schools network because of some closed ports, we are using MySQL so we are working on the same version of the DB.

- **Visual Paradigm 8.0 (UML)**
  - This is an advanced drawing program specialized in drawing diagrams by the UML standard.
- It meets our expectations and some of us have worked with it before.
- It seems like there is no other choice. At least not any free ones.

- **MS Office 2007 / 2010 (Report / Documentation)**
  - Very common document editor and reviewer.
- It might not be cross-platform, but using Cross Office everyone should be able to run Microsoft Office 2007.
- All have a great experience in Microsoft Office

- **Open Workbench (Planning)**
- It's a program for planning and organizing the team-members' time.
- We couldn't use Microsoft Office Project because not all had it installed.
- Our challenge is the lack of experience in this program, but in an educational project there should be something new to learn about.

- **Team Viewer (Share monitors)**
- Ability to view team members' monitors on different computers.
- It's easy to set up
- It runs on standard ports
- No alternative has proven to work at our location
- We were not able to get a monitor for our office

- **Skype (instant messaging and online-communication)**
- A program for writing each other online with pop-up for each message. Also possibility for IP phone-communication
- A lot of programs were suggested, but all could agree that Skype is the easiest program to use. This might be based on experience on the program.

- **CamStudio 2.0 (Video recording software)**
- A program for recording the screen of a computer
- Used for recording video documentation of some of the use cases. Links for these will be inserted in the relevant sections, but can be found on the DVD disc as well.

- **Sony Vegas 8 (Video editing software)**
- Professional video editing software for better presentation of our videos
- Used for making some effects in the raw video files

- **Windows Live Movie Maker (Video editing software)**
- Simple video editing software
- Used for gathering the video files and converting them into small videos that applies to the YouTube rules (>15 minutes and >2 GB)

1.7 Project plan - Nyvang

Here is a simple project plan in Open Workbench, made so we have an overview of the time that is going to be spent on the different stages of the project.

	ID	Name	Start	Finish	Name
	P	Project	17-09-2010	16-12-2010	
	pe	Project Establishment	17-09-2010	26-09-2010	Nicolaj
					Lars
					Adams
					Torben
					Claus
	incep	**Inception**	**27-09-2010**	**08-10-2010**	
	incepA	Inception	27-09-2010	08-10-2010	
	incepM	Lifecycle Objectives Milestone	08-10-2010	08-10-2010	
	elab	**Elaboration**	**04-10-2010**	**15-10-2010**	
	elab1	Elaboration 1. iteration	04-10-2010	08-10-2010	
	elab2	Elaboration 2. iteration	11-10-2010	15-10-2010	
	elabM	Lifecycle Architecture Milestone	15-10-2010	15-10-2010	
	conc	**Construction**	**25-10-2010**	**03-12-2010**	
	conc1	Construction 1. iteration	25-10-2010	29-10-2010	
	conc2	Construction 2. iteration	01-11-2010	05-11-2010	
	conc3	Construction 3. iteration	08-11-2010	12-11-2010	
	conc4	Construction 4. iteration	15-11-2010	19-11-2010	
	conc5	Construction 5. iteration	22-11-2010	26-11-2010	
	conc6	Construction 6. iteration	29-11-2010	03-12-2010	
	concM	Initial operationel capability Milestone	03-12-2010	03-12-2010	
	tran	**Transition**	**06-12-2010**	**16-12-2010**	
	tran1	Transiotion 1. iteration	06-12-2010	10-12-2010	
	tran2	Transiotion 2. iteration	13-12-2010	15-12-2010	
	tranM	Product Release Milestone	16-12-2010	16-12-2010	
	1	Nicolaj			
	2	Lars			
	3	Adams			
	4	Torben			
	5	Claus			

Figure 1 2 - Project plan version 1.0

The planned time is as follow:

We have made time in the schedule for six iterations in the construction face. The reason for choosing six iterations is that we split up and we will make two use cases per iteration. Normally we would do one use case per iteration, however some of them are quite simple and we cannot all five do the same, at the

same time. We cannot tell whether we only have time for one-, all six- or even more iterations. Please note, that this is the first version of the timetable and we are almost sure that it will be changed at some point, when we have a better idea of the different iterations in the construction phase.

1.8 TeaN TALC Risks - Nyvang

In every project, there are risks. The same apply to our project.
We have made a brainstorm of the different risks that we expect throughout the project. When this is done, we can deal with most problems before they appear.

How will we deal with them?

We have made a table with listed and ranked risks. It shows some risks in different categories. When dealing with the risk list, it should be in mind that some risks can be eliminated, some minimized and some we have to learn to live with.

How to read the risk-table:

The magnitude, which is the frequency if occurrence, is measured in a scale where one being the lowest and five the highest.
The impact is measured in L, M, H, and C. More specific, the measurements are Low, Medium, High, and Critical.
The risks appear in a prioritized descending list, that is, the most critical is first.

We have chosen to include the top 3 risks in the report, and the rest in appendix.

<RSK-01> Absence (People)			
Magnitude	**Description**	**Impact**	**Strategy plan**
3	If some in the team is much absence for some reason	C	All team members have a responsibility to help others catch up on project events

<RSK-02> Lack of Skills (People)			
Magnitude	**Description**	**Impact**	**Strategy plan**
5	Lack of skills is for all of us because this is a study project.	H	Read and try to do what you have red. Ask other team members for advices

<RSK-03> Complexity (People)			
Magnitude	**Description**	**Impact**	**Strategy plan**
5	Have we aimed our project to be to complex it may result in a non-functional product	H	We stop in every iterations to see how far we are and if something should go in or out

1.9 Part conclusion – All

In this chapter, the team and the project have been established. The shared vision between the company and the team has been declared, and the time table over the planned events throughout the rest of the project has been worked out. Besides this, the company has been described and the risks of the project have been prioritized. Finally the problem description is in place and will function as the overall guideline for the project. So basically the foundation of the project is established. In the next chapter we will start on modeling the systems features with use cases, so we get an overview of the different functions of the system. We have made a foundation for our personal learning, as we described in the problem definition, besides the theory that we have learned in the class we have chosen two other techniques to learn about, that is SCRUM and agile development.

2.0 Inception

2.1 Purpose

The purpose of the inception phase is to state the system requirements, decide the boundary of the system and finding the actors and use cases. As we are unsure of the many of the specific measurements with metrics etc., we will present a mockup for the company. This will together with the shared vision give us a more specific understanding than we have now.

2.2 Overview of activities

In the inception phase we will do the following activities:

- Present a mockup for the company
- Elicit stakeholders request
- Find actors and use cases
- Structure use case diagram
- Detail use cases (aprox. 10%)
- Review requirements
- Revise project plan and risk list
- Develop iteration plan for elaboration

2.3 Mockup - Claus

To revise the requirements, we decided to make a mockup. The mockup is a good way to give the company an overview over the different features – as we see them. As an addition we needed the company to specify some of the measurements and which metric units they use. This is very important for the diagrams, where we need to describe the different variables and operations.

We made the mockup in NetBeans as a GUI, but without any real programming.
In figure 2 1, you see an example of the mockup. The rest of

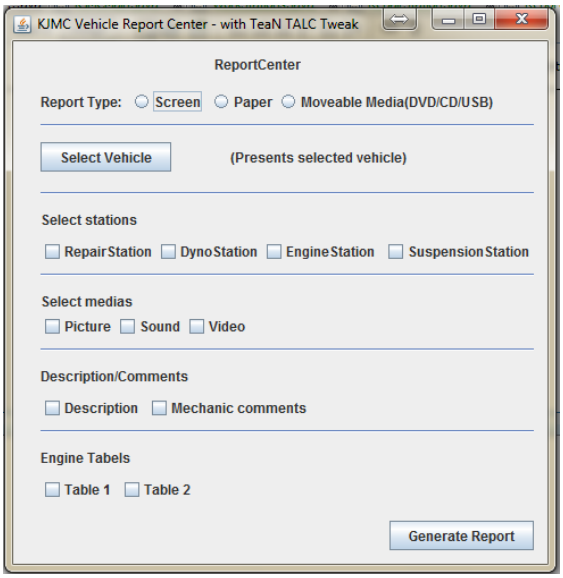


Figure 2 1 Mockup for company

the images and the e-mail correspondence with the company can be found in Appendix C

Considerations

As the company specializes in a quite special area with some very special values, we had some issues quite early in the project. When we had to do the Use cases, OC, SSD etc. we had to know which measurements are taken from which stations and what parts they are bound to. Therefore we decided to make a mockup and show it to the company. By doing this, we could get the different variables, data types etc. for the different stations. This makes it easier to do the diagrams, and therefore easier to create the different classes. Besides, it´s always an advantage to have the end user(s) on the sideline when doing a project, despite this, many team´s often forgets to involve the users.

2.4 Elicit stakeholders request

The stakeholders of the system are defined as "user". By user, we mean the mechanic who is working on the specific bike. As the some of the mechanics are partners in the company, they are not distinguished from the other mechanics.

The user uses the system to store data about vehicle.
The users request is to compare data from before and after the vehicle has been through the store.
The user also wants to use the data to generate a report about the vehicle.

2.5 Find actors and use cases

The company has little use for security for this system as it doesn't contain vital information of any kind (except some customer information). There will not be any social security numbers, prices, credit information etc. In most cases, a username and a password is the best solution when designing a system for a company, namely because of vital information and user control. In this case we choose to make a non-password system because the "only" place where the user actually is logged in, is so the changes can be traced back to a single person and to use a timestamp when working on the order. This means that every measurement has specific user initials, then anyone can see who´s working on the bike and, if needed, he/she can ask questions to the specific user. Considering this, we will not make an admin login, because the user management is done by the users themselves. The login-function should only check if the initials are correct (to minimize misspelling, mistyping etc.) then we can avoid someone using wrong initials and no one knows who to turn to if anything is wrong or non-documented.
From these considerations, we found that our system doesn't have different roles with different goals – which are why there's only one actor (see below).

Actor:

User (primary actor, only actor)

Use cases:

From the Shared Vision, Needs and Features we can list the use cases as following:

- Vehicle - CRUD[7]
- Owner - CRUD
- Create Whiteboard

[7]The basis operations Create, Read, Update and Delete

- Manage Whiteboard
- Search Owner
- Upload Media
- Make Report
- Search Vehicle
- Register Order
- Register Employee

From the above list, we will list the use cases as follows:

Use Cases:

UC1: Register Vehicle
UC2: Register Owner
UC3: Create Whiteboard
UC4: ManageWhiteboard
UC5: Search Owner
UC6: Upload Media
UC7: Make Report
UC8: Manage Owner
UC9: Manage Vehicle
UC10: Search Vehicle
UC11: Register Order
UC12: Register Employee

It would make sense to join the register* and manage*-use cases under a main case (CRUD), but at this point we already made them separately and we decided not to go back and change the diagrams.

2.6 Structure Use Case Diagram

Since we only have one actor, the use case diagram is very simple.

In the use case diagram, we demonstrate the interaction between the actor (user) and the system. The use case will help understand the basic of the technical contents of iterations.

Figure 2 2 shows the use case diagram

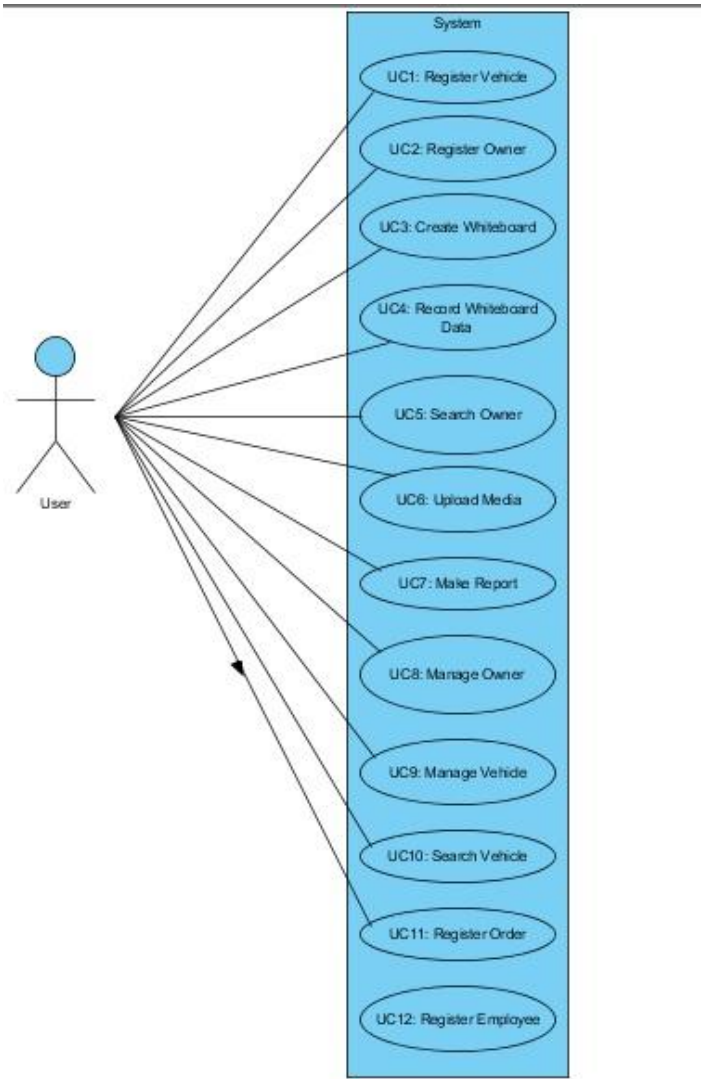


Figure 2 Use Case diagram overview

2.7 Detailed Use Case - Lars

As shown in the last section, there are at the moment 12 use cases in total. In this section you will see one of the use cases fully dressed. This will give a more detailed overview over the different stages, scenarios, goals etc. The chosen use case is UC1 Register Vehicle, as this one generally will be the example we are working with, in the next phase. We have chosen UC1 because it's the foundation for working with the system. The user cannot work with the rest of the system before there is a vehicle in place.

Since we are making 2 UC per iteration (see 1.7) there is one more fully dressed, UC2 Register Owner, see Appendix B

UC1: Register Vehicle
Scope: Tweak[MC]
Level: User goal
Primary actor: User
Stakeholders and interests:
Primary actor wants to store basic information about vehicle
Preconditions: User logged on
Post conditions: Vehicle is saved
Main Success scenario:

1. Need for Vehicle registration occurs
2. User starts a new Vehicle registration
3. User enters basic Vehicle information (type, brand, model, year, chasisNumber, licensePlate)
    a. User chose to attach an existing owner to vehicle
        i. User start Usecase: Search Owner
        ii. User select found owner
        iii. System attach owner to vehicle
4. System give VehicleID
5. System saves Vehicle
6. User ends Vehicle registration

Special requirements: None
Technology and Data Variations List:
VehicleID is given by system
Frequency of occurrence:
When a new vehicle have to be repaired/optimized approximately 500 per year.

2.8 Revise requirements - Nyvang

After sending the mockup to the company (see 2.3), we have received a positive answer about the GUI prototype, and now we have an idea of how the system should work and what kind of data we have to include in the generated report. In this activity we have added one UC more, called UC13 Login; it will be used to register which user is working on which vehicle. We have renamed UC12 Register Employee to UC12 Register User, because it should refer to the one using the system who we are calling user. Since it is a single user system used on one computer only and we don't know if they have a system for their employees, we have chosen to include the user registration in our system. In the future it should be possible to import users to this system or export them to another system.

As a result of the above, the change to the use case diagram is shown in the figure 2 3 below.

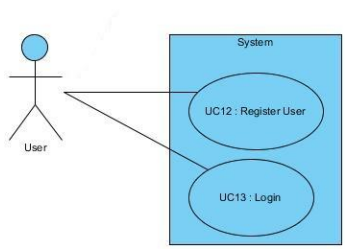


Figure 2 3 – Two new use cases added

2.9 Revise project plan and risk list - Nyvang

At this point, we have revised the project plan and we haven´t made any actual changes, however we are aware that changes will come. The revised project plan is to be found in Appendix P. The only version included in the actual report is the first.

Going through the risk list, another risk has come up. This is the contents of the report and especially what we decide to place in the appendix. This could have a massive impact on the report, as we might prioritize the wrong contents. Therefore another risk is added, shown below.

<RSK-14>Production (Report)			
Magnitude	**Description**	**Impact**	**Strategy plan**
2	Content of the report versus content of the appendix	H	All team members need to agree on the contents both places

Yet another risk has come up. This is about our aim, as we wrote earlier the overall goal is to learn and to complete the system so the company can use it in their organization. This can however be a risk as we might use too little time on the different phases throughout the project to achieve our goal. Some team members may fall behind on learning because others might know more about e.g. coding in Java. This may not be a big risk in the project itself, but can have an impact on the exam for the different team members. Therefore the risk is prioritized as shown below.

<RSK-14>Personal (Learning)			
Magnitude	**Description**	**Impact**	**Strategy plan**
2	Team members may have lack of time to learn the full project	H	Basically, the individual team member are responsible for going through the other members work and ask questions if anyone comes up

2.10 Develop iteration plan for 1st iteration - Nyvang

As mentioned in section 1.7 Project Plan, there will be two use cases per iteration. In the first iteration we have decided to start with UC 1 Register Vehicle and UC 2 Register Owner, to implement the core structure in the system. It's also implementing business value for the system.

2.11 Business analysis - Adams

KJ Motorcycles are a company with a niche business and a small clientele; this gives both pros and cons.

To optimize the system and to being able to understand and advise the company we have made a business analyses. This is done from several models that all help giving the bigger picture of the company´s situation. The

5 models used are listed below. Thus only the "Five layered model" are included in the report, the rest can be found in Appendix E

- Five layered model
- Value Chain analyses
- Five Forces analysis (Appendix E)
- Generic Strategies analysis (Appendix E)
- SWOT analysis (Included in the five layer model below)

2.11.1 The 5-layered business model

This is tool used in business analysis and it shows the different relationship with in the flow of the business and it is demonstrated below in the model.

MARKET LEVEL – Customers & Competitors

Customers: Motorbike-enthusiast, ordinary people from DK, some people from outside DK (Mostly Norway).
Competitors: Other mb- web shops, other mb-workshops in DK

OFFERING

Service: Advice on optimization and tuning. Actual optimization, tuning or complete rebuild of motorbikes. (Mb-Report, Mb-DVD)*
Product: Motorbikes, ATV's, car leasing. Spare parts to motorbikes and ATV's from web shop and workshop.
Customer Needs: Mb-parts, mb-tuning, mb-optimization.
Bundling: "Lease or buy a vehicle with service" - Leasing include service in the leasing period. Buying vehicle with service - the customer decide with service type to buy (service price per year)

ACTIVITIES & ORGANISATION

Inbound Logistic: Suppliers for different parts for both ATV's and motorbikes is: Matthies and Bridge Stone.
Operations: Dynorun, optimization, styling, complete rebuild, brake-pipelines.
Outbound Logistic: Kj uses GLS to send parts to their customers.
Marketing & Sales: Kj-Motorbikes has a website where customers are able to order parts from the store. There, customers can also sign up for their newsletter.
Service: When a customer has bought some kind of service in their shop - the customer can always come back to get advices or help to change something small for free.
Technology Development: Our project is to make kj a system that can generate a vehicle data collection with history data(changes, measurements, pictures, videos etc.) about the vehicles.
Human Resource Management: Kj-Motorbikes has mechanics to do every job in their work shop. They have a special need for knowledge in optimizing bikes, tuning bikes and total rebuilds of bikes. Furthermore they have a need of knowledge in the DynoStation. Therefore mechanics with the required knowledge and experience is hard to find.

RESOURCE LEVEL

Strengths: KJ-Motor bikes have a DynoStation that can measure air resistance as the only one in Denmark. Kj-Motorbike can rebuild a motorbike totally by them. They got high experienced mechanics that can advise customers in the right direction.
Weaknesses: The company has a small clientele, so if they make a mistake it would damage their reputation quite fast.
Opportunities: New system for Kj-motorbikes is under development. This will give kj-motorbikes opportunity to offer their customers a new attractive service.
There are several importers in Europe, getting their own brands done (they are mostly looking like the official brands) and selling the parts to cheaper prices. The official brands are usually on the market for about 6 month before they are imitated.
Threats: Competitor web shops because of lower VAT and cheaper part prices.

MARKET LEVEL - Factor Markets Suppliers

Suppliers: To rebuild vehicles of any art they must have reliable suppliers that deliver the best parts.
Capital: For the leasing part of the company they have one or two main suppliers for capital (banks).
The car they are leasing comes from many different suppliers - mainly from Germany.

2.11.2 Value Chain analysis

This is a systematic approach to examine the development of competitive advantage, and was created by M.E. Peters in his book competitive advantage (1980). It consists of series of activities that create and build value.

They culminate in the total value delivered by an organization. The organization is split into primary and support activities.

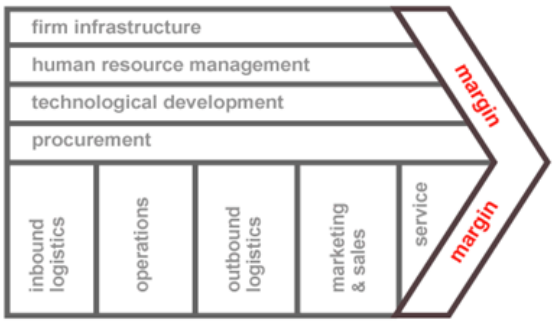


Figure 0-1 Value Chain

2.12 Part conclusion - Adams

In the inception chapter we have gained a general idea of what the system should do. The use cases outline an understanding of the system features. The mockup has shown the company what our general idea is all about and they have approved the layout idea. The project plan is revised and there's made some changes and especially the in the risk list, some risks are added. Regarding the mockup, the company´s response was that the metrics should not be pre-defined. The optimal solution is that the user adds a table in a given size (varies from bike to bike). The specific metrics are to be added by the user as they also vary from bike to bike.

The business analysis gave an overview over the marked, customers and generally about the state of the company. In the next chapter we will start first iteration in elaboration with
UC: 1 Register Vehicle and UC: 2 Register Owner in the next chapter the actual architecture are created and analyzed.

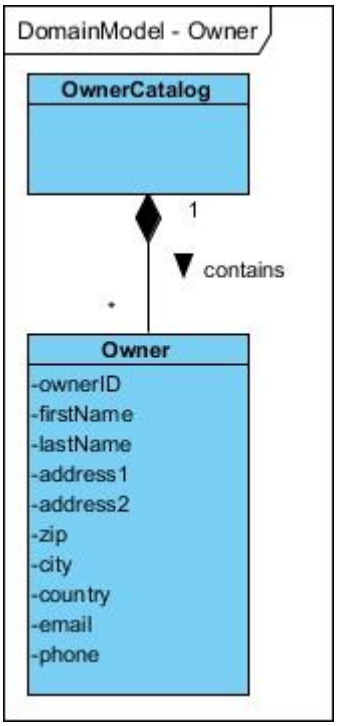


Figure 3 1 - Domainmodel

3.0 Elaboration - 1st Iteration (Register Vehicle & Register Owner)

3.1 Purpose - Lars

We have chosen two use cases; Register Vehicle & Register Owner, to implement the core structure in the system. In this chapter we will design and implement the use cases to see if the ideas we have can be completed.

3.2 Model the domain - Lars

When modeling the domain we get an overview over the complete system structure. From iteration to iteration the domain model will grow.

The conceptual classes in our domain model figure 3 1 is found in the UC2 Register Owner and new conceptual classes in the domain model figure 3 2 are found in UC1 Register Vehicle, using the class category list and noun phrases technique.
The conceptual classes in figure 3 1, related to UC2 are Owner and OwnerCatalog.
The conceptual classes in figure 3 2 related to UC1 are Vehicle and VehicleCatalog. The conceptual classes have been associated to each other (where an association is needed) and the multiplicity is shown for each class.

The conceptual class OwnerCatalog is strongly associated with Owner, also called a composition or aggregation, which mean that Owner cannot exist without OwnerCatalog. The multiplicity here is 1...* meaning that one OwnerCatalog can contain one to many Owners, and an Owner can only belong to one OwnerCatalog. The same rules apply to the VehicleCatalog and Vehicle.
The domain model is "the real world picture" and therefore we only show attributes and no operations (methods) in each class.

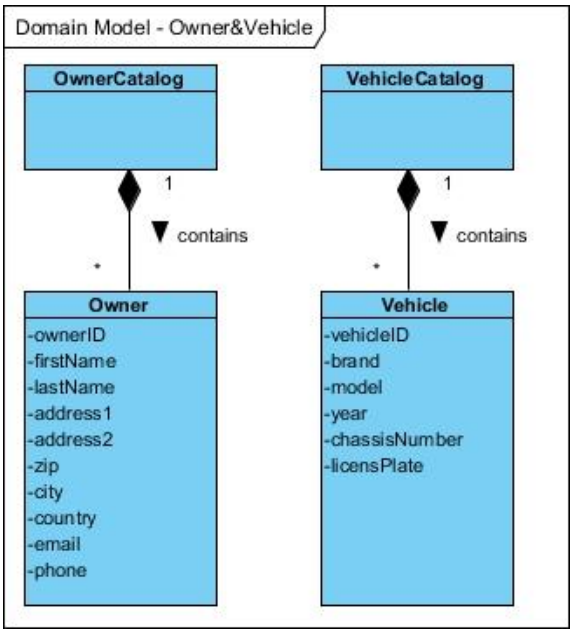


Figure 3 2 –
We allready now know Vehicle and Owner objects needs unique id's for later search in DB.

3.3 Detailed Use case - Claus

3.3.1 Write use case text

See chapter 2.7 and Appendix B Use Case

3.3.2 System Sequence Diagram, SSD

SSD is a model of all input events in a use case. This is done for the main success scenario, also known as the happy path. It shows the order (sequence) of system input events and output in a use case.

The diagram only shows the interaction between the user and the system, it will not go behind the system "border", and for this we will use the Sequence Diagram (see 3.6.1)

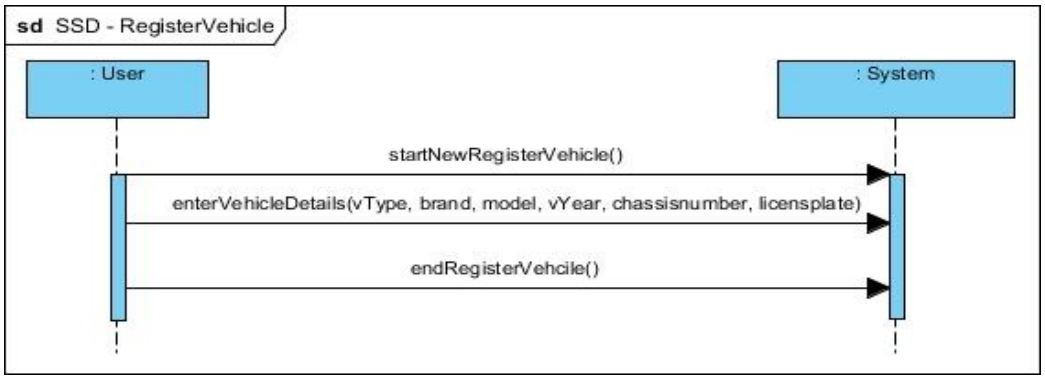


Figure 3 3 – The user starts an operation to the system "Register Vehicle" – he enters the necessary details, and ends the operation. Behind the scene the system has saved the vehicle in a relational database.

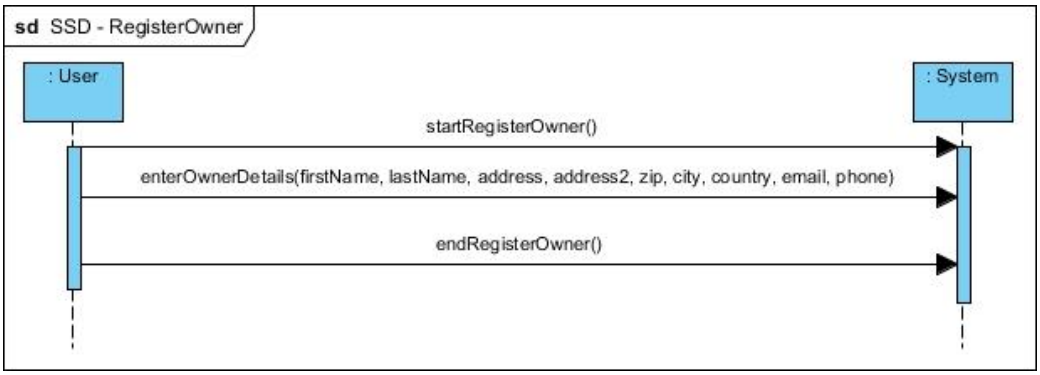


Figure 3 4 – The user starts a operation to the system "Register owner" – he enters the necessary details, and ends the operation. Behind the scene the system has saved the owner in a relational database

3.3.3 Operation Contracts

Operation Contracts [OC] is only done when there is a change of state in the domain model. For example, if an Owner object is created, a state is changed. If we e.g. search for an already existing Owner (or other object), no change is made. The same apply for OC1 Register Vehicle

The OC is a contract with the domain model about the result when registering a new owner.
It could be:

1. Object creation/deletion
2. Association formation/destroying
3. Setting of attributes

The most important thing in an OC is the post condition, describing which changes have been made.
Pre-condition states something about order of sequence so this is of course also important.

Below is the OC1 of UC1 Register Vehicle

Operation Contract – OC1 enterVehicleDetails

TeaN TALC - TweakMC

OC1: Enter vehicledetails (type, brand, model, year, chassisNumber, licensePlate)
CrossRef: Register Vehicle
Pre Conditions: Vehicle catalog exist
Post conditions:
- Vehicle instance v created
- v was associated with VehicleCatalog
- v.type was set to type
- v.brand was set to brand
- v.model was set to model
- v.year was set to year
- v.chassisNumber was set to chassisNumber
- v.licensePlate was set to licensePlate
- v.vehicleId was set by system (shown in domain model)

The OC2 for UC2 Register Owner is created with the precise same structure, see appendix F.

3.4 Architectural analysis - Torben

The system is based upon the layers listed below.

Layers:

- View
- Control
- Model
- Data Access
- Persistence

We are using MVC model, means that UI do not have direct contact with the model layer. It is not up to the model layer to decide if the UI should be green or blue, but just to present results there should be shown in the UI. We broke the rule about not connecting the model layer directly to the GUI (searchOwner/Vehicle). *Note: it's only when we require already known/created object information.*

For us, it seems to give more sense to send an object up through the layers than sending e.g. 10 lines of text pr. search. Example; One Owner object contains 10 lines of information. If we search for Larsen, we may find 10 owners = 100 lines of text. We send the objects directly to the GUI and let the GUI ask the owner class which information the object contains, and presents the needed information.

The diagram below gives an overview of the architectural design (MVC).

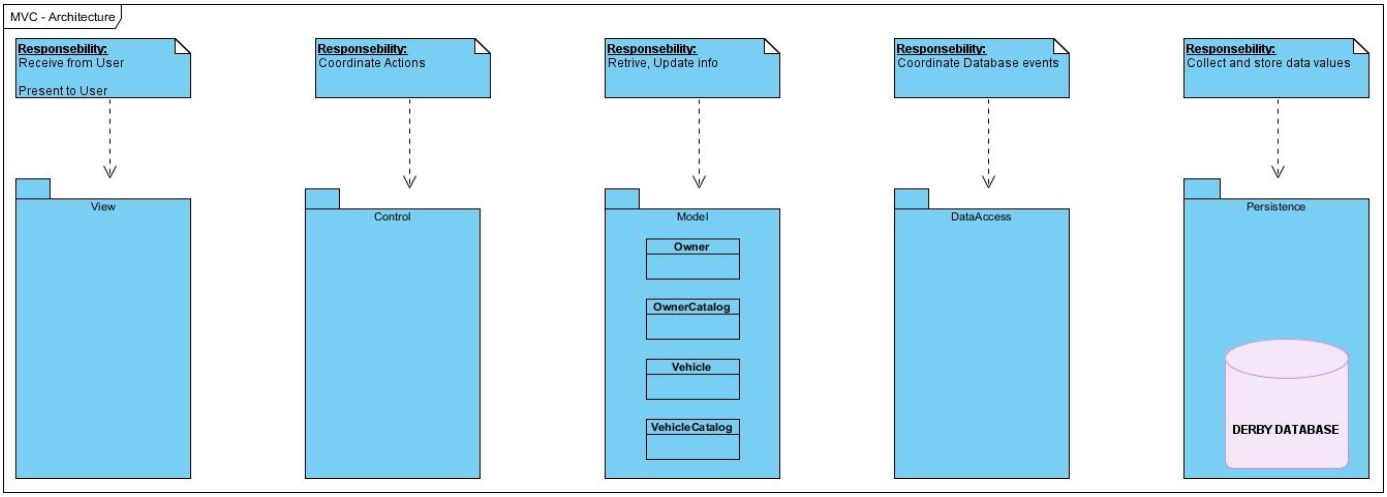


Figure 3 5: Model – View – Control (MVC) architecture.

3.5 GUI design

Here is the overview of the "rules" for our GUI. We will try to make it as understandable as possible for the user (read no surprises).

Layout:

Theme:	Theme follows Operation System (Mac, Linux, Windows etc.)
Background:	Gray
Buttons:	Solid / Metal
Headers:	Tahoma, Black, Bold - Size 18.
Text:	Tahoma, Black, Plain – Size 14

Window Size: 1027 x 768 (changes to users screen resolution)

Familiarity:

The system is build up on a natural interface, and a language that's familiar to the user.
The system is following our tapped window pattern, so the user feels familiar with the system no matter which window he's looking at.

Consistency:

The layout is build up on tapped windows through the whole system.
E.g. If the user chooses "Vehicle" tap, a new tapped pane will show all that has to do with vehicles (marked with red in Figure 3 6).

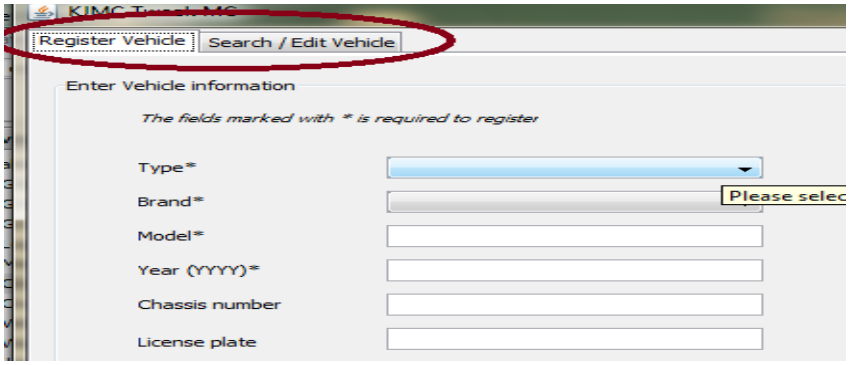


Figure 3 6 - Consistency

Minimal Surprise:

The system should not come with surprises for the user. We attempt to always keep the user in focus, so that he feels he's in control. The way we do it:
For every time we have implemented a new feature we ask ourselves, "What if..?" as many times as we feel it's necessary to make sure the user feels he's in control of the systems behavior.
Furthermore we will send beta versions to the company – so they have an opportunity to tell us if they were surprised on any features.

Recoverability:

In the whole system there's buttons to reset the current operation.
If the user chooses 'quit' in the menu 'help' it will ask "are you sure you want to quit?"
If the user chooses 'yes' and haven't saved it will ask "do you want to save the current operation? ".
Furthermore we have talked about implementing a new feature "Send helpdesk email".
This feature is when the user is stuck and don't know what to do, he will be able to send us (developers) an email with a log file attached. The log file will show us exactly where things went wrong, and we will take contact and help the user.

User Guidance:

At some point we will implement a help menu that can be called from anywhere in the system.
In the menu 'help'– you will see a help system that can guide the user through any operation.
This menu can also be found with F1 for any operation.

3.6 Use case design – All

Now we are moving away from "the real world" and putting on our software hat.
With help from Sequence- and Design Class-Diagram, we will show how we design the system. This is made after the principal of the responsibility driven design. That means that everyone work together, but everyone have the responsibility for their own tasks. To help us do this, we have used four of the nine GRASP patterns: Controller, Creator, and Low Coupling/High Cohesion. Patterns are solutions, there have been tested and approved, to commonly known problems that you run into several times during software design/construction.

The question is how we can design the system from the above?

Questions:

1. The controller pattern: Who should be responsible for controlling input from the GUI to the system?
2. The creator pattern: Who should be responsible for creating the Owner objects?
3. The low coupling pattern (related to high cohesion): How can we ensure that we have low coupling (or is it even possible in this task)?
4. The high cohesion pattern (related to low coupling): How can we ensure that we aren't "overloading" a class with too much responsibly?

Answers:

1. We have made a new object, ManageOwnerController to distribute all input from OwnerGUI to the system
2. For now it's not necessary to create new owner objects when creating a new owner.

When we create a new Owner the OwnerCatalog will pass the parameters to the OwnerDAO, which will transform them to SQL and execute them to the database.
Anyway the responsibility for creating new Owner objects are given to the OwnerDAO, because when the user search for an Owner, the result(s) are objects stored in Lists.

3. This will result in high coupling because the dataaccess layer knows about classes in the model layer, but again it seem to make more sense regarding what we concluded in 3.4.
4. It is important that we don´t give e.g. OwnerCatalog responsibility for creating a new Vehicle objects (Low Cohesion), OwnerCatalog should only concentrate about tasks regarding to Owner CRUD (High Cohesion)

The same answers apply to UC1 Register Vehicle. (Of course we have called the controller ManageVehicleController and the creator VehicleDAO instead).

3.6.1 Sequence Diagram & Start-up Sequence-Diagram (Dynamic view)

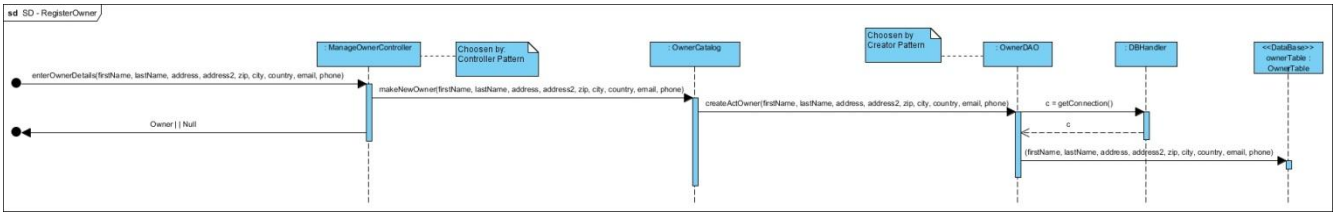


Figure 3 7 – Sequence diagram for Register Owner (Explanation and fullsize – see Appendix I SD – Register Owner)

SD diagrams show how the system works from GUI and down. In this diagram you´ll see all the objects used to create a new owner, and each objects method-call to the next object. We have added a note on which objects there are chosen by what pattern, as told above.

Looking at the SD for registering Owner, we have some objects that have to be created before we can use the system. We do this in a start-up sequence diagram. This is about the objects ManageOwnerController, OwnerCatalog, DBHandler, and OwnerDAO. See Figure 3 8

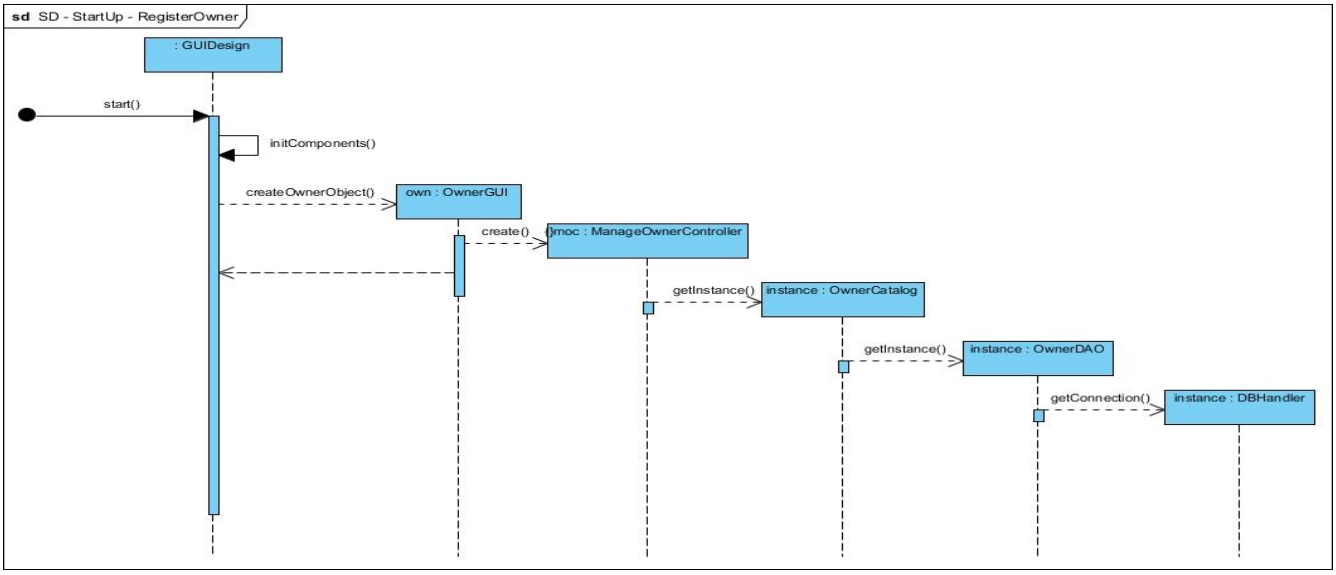


Figure 3 8 – Startup diagram for the classes involved in the Register Owner use case

In the startup-diagram there is a method called getInstance to the OwnerCatalog and the OwnerDAO, in these 2 classes we have chosen to use a Singleton Pattern. When we use a Singleton Pattern we make sure that there only will be one object of this class (for more information about Singleton see 3.7 Implementation).

3.6.2 Design Class Diagram (Static view)

Now we are in the "engine" of the system, here we will see all classes, method-signatures and attributes. It is our "total" view of the system. We have the possibility to see if there should be high/low cohesion or high/low coupling, because we will connect all the classes with dependency arrows. It looks a lot like the domain model, but here are shown much more technique, it also grow like the domain model as the project is moving on. Figure 3 9 will show a DCD for our 2 use cases UC1 & UC2.

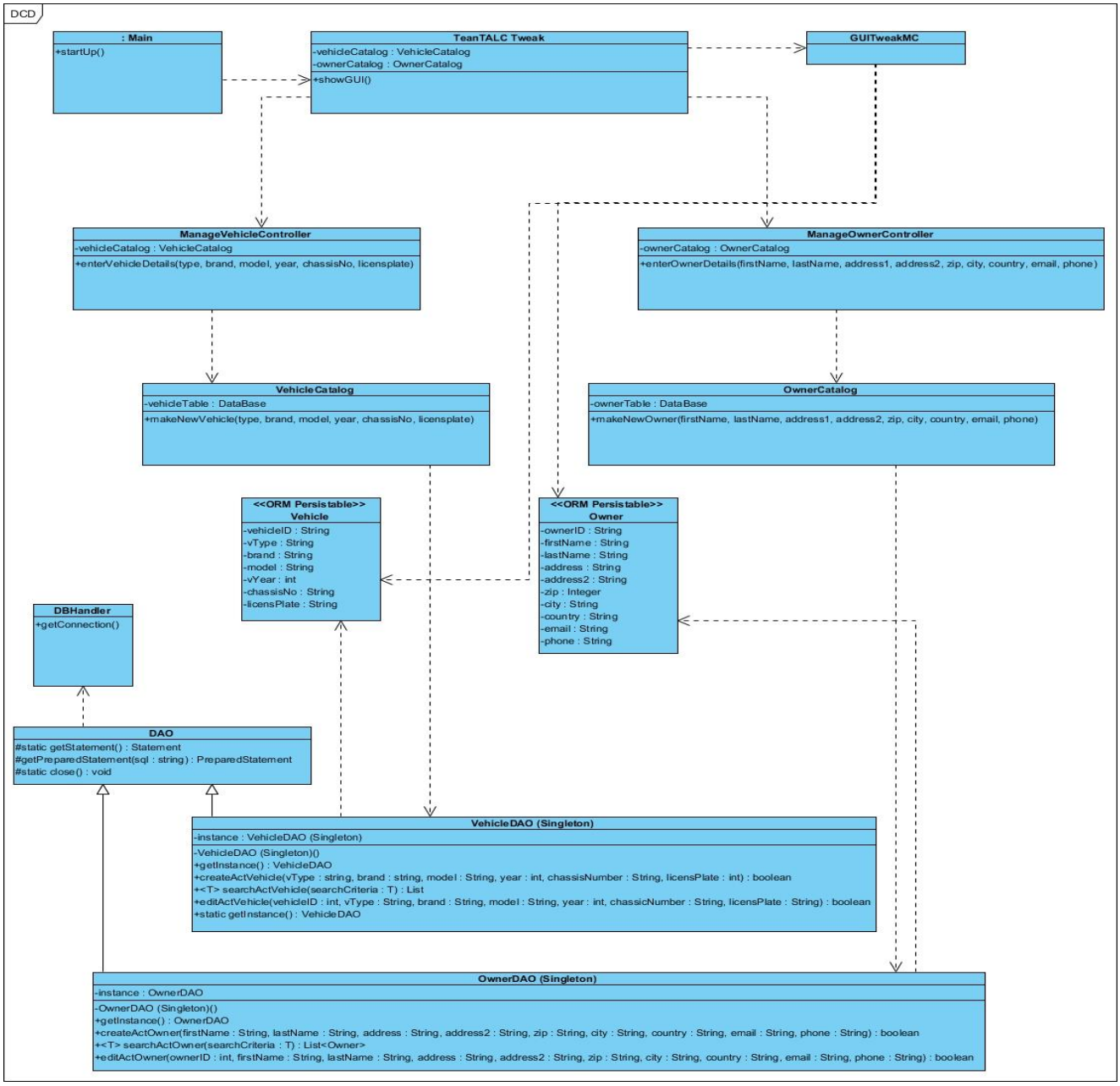


Figure 3 2 – The design class diagram for the first use cases

There are 2 different arrows on the diagram, it is the ones going from VehicleDAO and OwnerDAO to DAO, and these arrows are Generalization, meaning that the 2 subclasses inherit from 1 super class.
We also see that Vehicle and Owner are connected to both DAO and GUI, as we talked about in 3.4

3.6.3 Database Design

We have two entities in this design, Owner and Vehicle. The owner table is not taken to 3NF, because we have zip and city; this is done because we have city auto filler depending on the zip in the GUI layer. We have made the ID's in both tables to primary keys, set by the system (generated by the database) and also indexed them; this gives a quicker searching. We have chosen to let the database handle the auto numbering for Primary keys, because the system catalogs should not remember the last available unique ID number.
Some of the columns are nullable (N) meaning that it is allowed to leave them empty.

3.6.3.1 Mapping

We have taken our relevant classes from the domain model and mapped them to entities. Then we have taken the attributes in the classes and mapped them into columns where necessary.

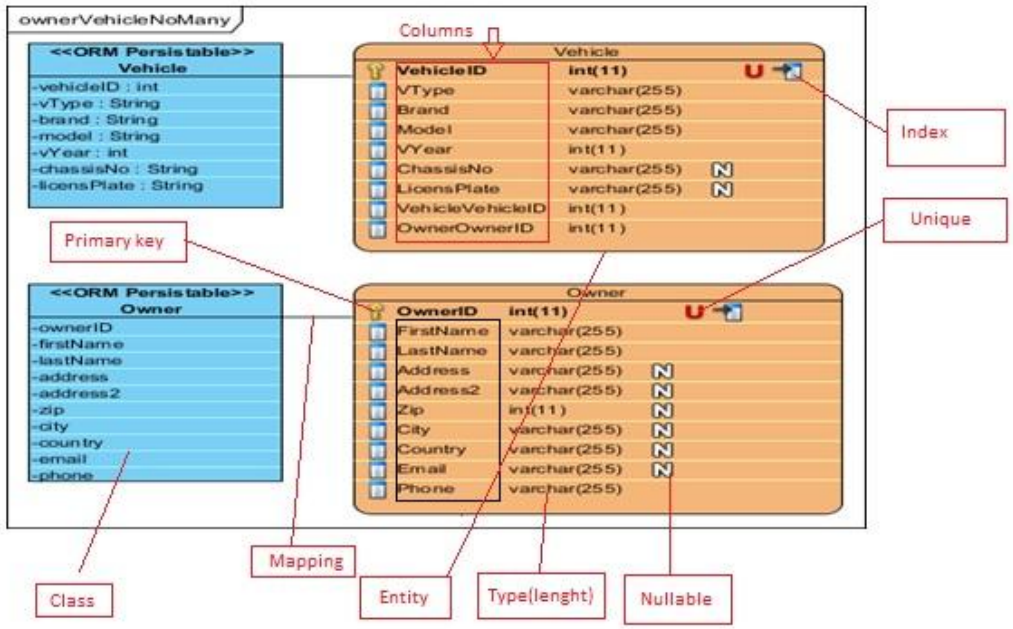


Figure 3 3 – Mapping of owner and vehicle from classes in domain model.

3.6.3.2 Indexes

See index as in a book e.g. Larman, if we would find something about Domain Model, it would be much faster to look it up in the index instead of reading through the whole book, to see if the text matches something about Domain models.

3.6.3.3 Normalization

We have mapped our entities to 2nd NF from the beginning.
It could be a consideration to map it to 3rdNF by creating a new table for zip and city, and set zip as a foreign key in the owner entity. (See figure 3 10).

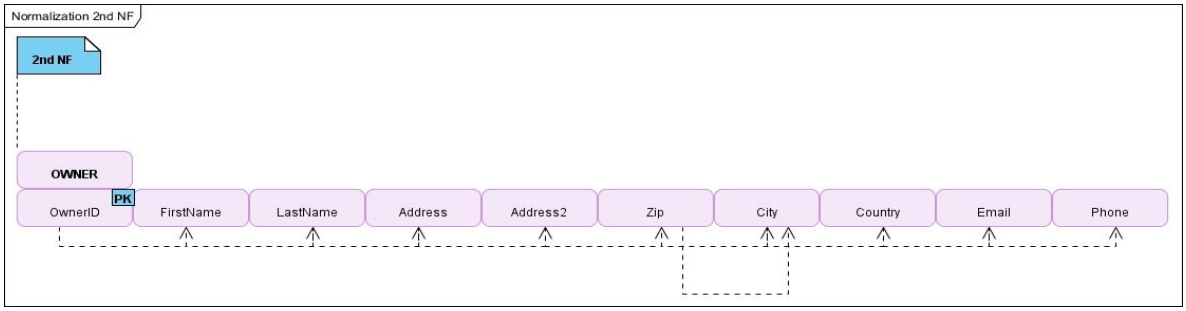


Figure 3 10 Normalization – 2nd Normal form

An owner has a unique id that will give us the rest of the entities. If we have the zip we can get the city – For moving into 3rd NF we will have to make a new table for zip and city. (See figure 3 11)

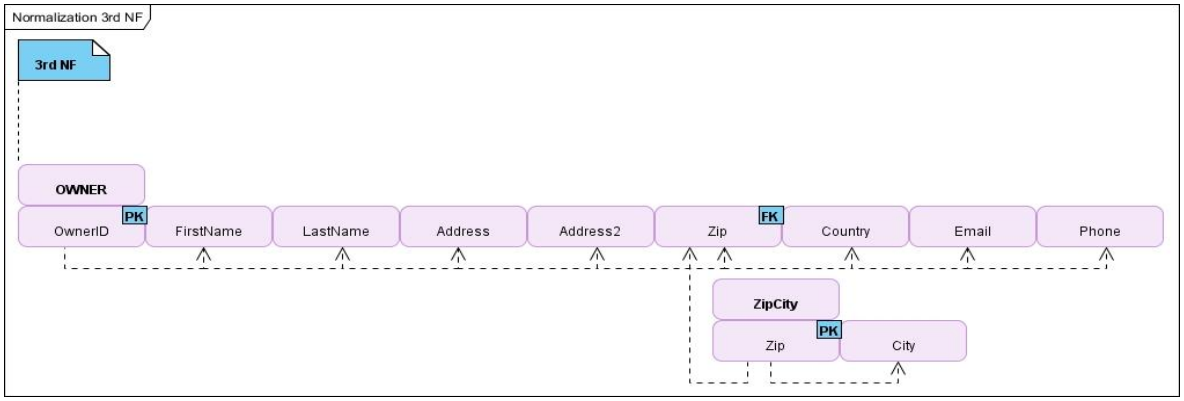


Figure 3 11 Normalization – 3rd Normal form

We split Zip and city into a new table. We are not doing this because we are loading zip and cities in from files. This is just an illustration of how it could be done in the DB.

3.6.3.4 SQL script

SQL script for Derby to create a table (owner)

```
CREATE TABLE owner (                                          1*
ownerID int NOT NULL GENERATED  ALWAYS AS IDENTITY,           2*
firstName varchar(50) NOT NULL,
lastName varchar(50) NOT NULL,                                3*
address varchar(50) DEFAULT NULL,
address2 varchar(50) DEFAULT NULL,                            4*
zip varchar(50) DEFAULT NULL,                                 5*
city varchar(50) DEFAULT NULL,
country varchar(50) DEFAULT NULL,
email varchar(50) DEFAULT NULL,
phone varchar(50) NOT NULL,                                   6*
PRIMARY KEY (ownerID)                                         7*
);

CREATE INDEX ownerIndex ON owner(ownerID);                    8*
```

Explanation

1: Here we create the owner table with the "CREATE TABLE" sql ansii/iso statement
2: Indicates that the ownerID is created as the IDENTITY and always Unique – auto generated by the database
3: Indicates that the column contains characters with the max length of 50 chars
4: Indicates that the default value is set to null
5: Column name are taken from attributes
6: Indicates that the value is not allowed to be null
7: Indicates that the ownerID is set to Primary key
8: Here the index is created on the ownerID for faster searching

3.7 Implementation – Lars, Nyvang

With the structure done we can begin the implementation of the design. The design itself is relatively simple. Following our DCD, (MVC structure) – gives us 12 classes to implement.
We simply just follow our SD and DCD and implement what we have designed.

As already told we have sent KJMC a Mockup to be sure that we are talking about the same, and from the response we have built the GUI with the correct fields to fill in Owner and Vehicle. This is the view layer. Control, model and dataaccess layer is not up to the KJMC and we have done them as shown in SD and DCD. We will not show the code here because there are not anything special to see, it is a lot of get and set.

3.7.1 Regular expressions - CheckInput

We will show some code from our utility package, here we have a CheckInput class(full documentation see Appendix Code), helping us to check if the fields in the GUI is correct, that there are letters in name, the mail address contains @, there are numbers in telephone and so on. Most of the checking we have done with Regular Expressions, there we can define if there should be letters, numbers, and special signs and how many. A big job to do but easy to use when it is done, and it can be used from project to project.

In this we check the email, there can contain letters, numbers, underscore and dot before there comes a @ after it can contain letters, numbers, hyphen and dot, at last it should contain letters min. 2 and max. 4.

```
public static boolean checkEmail(String isThisAnEmail)
  {
      boolean correctExpression = isThisAnEmail.matches
            ("^[_A-Za-z0-9-]+(\\.[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,4})$" + "{" +
isThisAnEmail.length() + "}");
      return correctExpression;
  }//end checkEmail
```

Explanation

^[_A-Za-z0-9-]+(\\.[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,4})$

means

^	# start of the line
[_A-Za-z0-9-]+	# must start with string in the bracket [ ], must contains one or more (+)
(	# start of group #1
\\.[_A-Za-z0-9-]+	# follow by a dot "." and string in the bracket [ ], must contains one or more (+)
)*	# end of group #1, this group is optional (*)
@	# must contains a "@" symbol
[A-Za-z0-9]+	# follow by string in the bracket [ ], must contains one or more (+)
(	# start of group #2 - first level TLD checking[8]
\\.[A-Za-z0-9]+	# follow by a dot "." and string in the bracket [ ], must contains one or more (+)
)*	# end of group #2, this group is optional (*)
(	# start of group #3 - second level TLD checking
\\.[A-Za-z]{2,4}	# follow by a dot "." and string in the bracket [ ], with minimum length of 2 and maximum 4
)	# end of group #3
$	# end of the line

3.7.2 Singleton pattern

We have also used the Singleton pattern to ensure that only one instance of a given class is created. Singleton classes have a private constructor and can therefore not be called from outside the class. Instead they have a static method we can call, and here we make a call to the constructor, there will check if an instance of the class have been created. If not, it will create a new or else it will return the existing.

[8] TLD Check stands for "Top-Level-Domain Check". This is a simple script which queries the whois.internic.net server and checks to see if the selected domain name is available or registered.

The script can return results for .com, .net, and .org domain names. Additionally, it will also check to see if the domain field is left blank, not a TLD, or if there are problems communicating with the Internic server.

The Singelton pattern

```
public class VehicleDAO extends DAO
{
//Instance variables
   private static VehicleDAO instance;

   /**
    * Private constructor for VehicleDAO
    */
   private VehicleDAO()
   {
   }//end constructor

   /**
    * Singleton method for VehicleDAO
    * @return instance of VehicleDAO
    */
   public static VehicleDAO getInstance()
   {
      if(instance == null)
      {
         instance = new VehicleDAO();
      }//end if
      return instance;
   }//end getInstance method
```

3.7.3 Prepared Statement – OwnerDAO

The last code example is from our dataaccess layer; here we use Prepared Statements to create a new owner (Full Java documentation see Appendix Code).

```
…code omitted
PreparedStatement pstm = getPreparedStatement("INSERT INTO OWNER (OWNERID, FIRSTNAME, LASTNAME,
ADDRESS, ADDRESS2, ZIP, CITY, COUNTRY, PHONE, EMAIL) VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");
        pstm.setInt(1, ownerID);
        pstm.setString(2, firstName);
        pstm.setString(3, lastName);
        pstm.setString(4, address);
        pstm.setString(5, address2);
        pstm.setString(6, zip);
        pstm.setString(7, city);
        pstm.setString(8, country);
        pstm.setString(9, phone);
        pstm.setString(10, email);
        pstm.execute();
…code omitted
```

The prepared statement create a SQL script where the question marks are replaced with the "*pstm.setInt(1, ownerID);" values. The order of the values is exactly the same as the order of the columns in the table.*

3.7.4 Unit test

We have chosen to make Junit test on our classes in Netbeans. We are making it for all relevant classes and layers. The good thing about Junit is that it is a kind of automatic test, meaning that it pre generate a lot of code for the chosen class. When we first have made the Junit test class we can run it again and again to check if any of our changes have any effect through the system. We are testing in single classes and we also test through the system, from the controller layer to the dataaccess layer. This have found a fail in one of our classes, we wouldn't have found ourselves, see Figure 3 12.

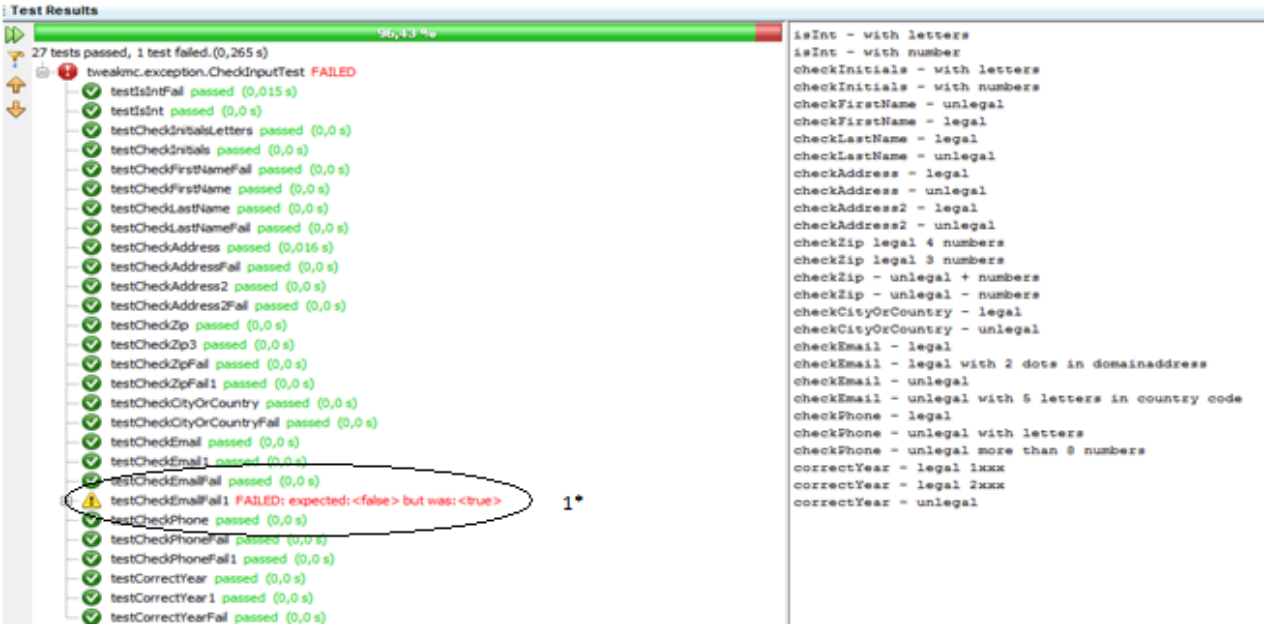


Figure 3 12 JUnit Test – Owner object (Full size see Appendix M - JUnitTest)

1: As we can see from the figure, the test have found an error in the method checking email address in the Owner class, and now we can correct this error, before moving on. Instead of moving on, and discovering the error at a later time when more code has been created (and more has to be corrected to fix the error).

```
 /**
 * Test of checkEmail method, of class CheckInput.
 */
@Test
public void testCheckEmail1() {
   System.out.println("checkEmail - legal with 2 dots in domainaddress");
   String isThisAnEmail = "test@dk.test.dk";
   boolean expResult = true;
   boolean result = CheckInput.checkEmail(isThisAnEmail);
   assertEquals(expResult, result); }
/**
 * Test of checkEmail method, of class CheckInput.
 */
@Test
public void testCheckEmailFail() {
   System.out.println("checkEmail - unlegal");
   String isThisAnEmail = "qqq.test@..test.dk";
   boolean expResult = false;
   boolean result = CheckInput.checkEmail(isThisAnEmail);
```

```
    assertEquals(expResult, result); }
  /**
   * Test of checkEmail method, of class CheckInput.
   */
  @Test
  public void testCheckEmailFail1() {
    System.out.println("checkEmail - unlegal with 5 letters in country code");
    String isThisAnEmail = "test@test.dkasd";
    boolean expResult = false;
    boolean result = CheckInput.checkEmail(isThisAnEmail);
    assertEquals(expResult, result); }
```

Code example of the JUnit test class

3.7.5 Review code

We have now done our first coding in iteration 1, and at the end we have encountered some problems with the GUI. The sizes are not so easy to manipulate and we have a big problem with the "main JFrame". Therefore we will have to redesign/code the "main JFrame" to fit the size, and the solution will probably be that we will write it ourselves instead of using Netbeans GUI designer.
We have also found* a new Layout style called "Nimbus" which we will implement in the 2nd iteration.

We have faced big problems with tracking errors through the system (connection problems, driver problems etc.). So we have decided to make a new class in the Controller layer, in the utility package that will collect and store all exceptions in a log file, with a timestamp; this will make it easier to track errors.

3.7.5.1 Considerations - Log file

How should we handle system errors?
Errors can show up on the screen so the user is informed on what may have gone wrong. This solution is not the optimal solution for the company because the users will, generally, have no idea on what to do about the errors. Therefore the discussion ended with the following solution.

All errors (exception) are logged to a *.txt file with a timestamp. By doing this, the programmer have the option to see the error and therefore have a better chance to find and work on the specific error.

How should we handle "Fail40[9]" errors?
System should present an error message telling the user what to do or correct and highlight the error.

3.8 Testing – Lars

3.8.1 Plan Test

As we have created a utility class called "CheckInput" – that verifies that the users input to fields – we will now test that everything is working according to the glossary (see Appendix L).
The first three test cases are shown below (see appendix M) for more test cases.

[9] Fail40 is a common used "error code" for user errors
* Found at http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/nimbus.html

3.8.2 Design Test

We choose Black box test structured in an equivalence set, for testing users input to the system.
Afterwards we made a white box test in a JUnit test on the systems inside behavior.
We have organized our black box test cases like this:

System name: **Tweak[MC]**	Phase: **XX**		
Test case: **XX**			
Risk: **XX** Extend: …			
Test cases:	TC1 - **XX**	Expected	Actual Result
Tx	*equivalence*		
Tx2	*equivalence*		
Done by:	**Date**	JUnit reference:	
Accepted by:	**Date**		

3.8.3 Execute equivalent test

System name: **Tweak[MC]**	Phase: **Elaboration**		
Test case: **Register Owner**			
Risk: **Medium** Extend: 1. Register a new Owner with valid and invalid values in fields. 2. If the user tries to push the "Save" button every field is checked and the status label will tell what's wrong.			
Test cases**First Name**	Equivalence	Expected	Actual Result
T1: "! Lars 32 " – User push "save" button	*First name may not contain signs*	First name field turns red	First name field turns red (see appendix M)
T2 : " " – User push "save" button	*First name may not be empty*	First name field turns red	First name field turns red (see appendix M)
T3 : " Lars 23 " – User push "save" button	*First name may contain letters and numbers*	OK	First name field stays white (see appendix M)
Done by: Lars	**29/11-2010**	JUnit reference:**see Figure 3 12**	
Accepted by: Susanne	**29/11-2010**		

System name: **Tweak[MC]**	Phase: **Elaboration**		
Test case: **Register Owner**			
Risk: **Medium** Extend: 1. Register a new Owner with valid and invalid values in fields. 2. We have a 'focus lost listener' that will highlight the field if something's wrong 3. If the user tries to push the "Save" button every field is checked and the status label will tell what's wrong.			
Test cases**Zip**	Equivalence	Expected	Actual Result
T4 : "98" – User push "save" button	*Zip should be between 100 and 9998*	Zip field turns red	Zip field turns red (see appendix M)
T5 : "9999 " – User push "save" button	*Zip should be between 100 and 9998*	Zip field turns red	Zip field turns red (see appendix M)
T6 : " 100 " – User push "save" button	*Zip should be between 100 and 9998*	OK	Zip field stays white and city field shows city

			(see appendix M)
Done by: Lars	**29/11-2010**	JUnit reference: **see Figure 3 12**	
Accepted by: Claus	**29/11-2010**		

3.9 Revise requirements - Nyvang

As the first iteration is complete, we now have a working system. It is now possible to create an owner and a vehicle, which are saved in the database. Despite the fact that this is some very simple functions, we have a system that is now working and we have the possibility to perform the initial user tests and of course, we can also test the system ourselves. We have found that there will be some flaws in the system, and it will be nice for user to report it directly to the designers/coders, therefore we will make a new Use case called Help-Desk

3.10 Part conclusion - all

Looking at our problem definition we have come around the personal learning, getting stronger and found our weakness in Netbeans GUI designer. We have learned that GUI is a very time consuming issue, and therefore we will also revise our project plan.
We have looked at the database and are starting to get a feeling of the "SQL way". After the revising of the code we will add some new packages in our Control layer, instead of making a new layer.

4.0 Elaboration – 2nd Iteration (Register Vehicle & Register Owner)

4.1 Purpose - Tvupper

In this chapter we will implement some of the special features, we discovered doing 1. Iteration, of the use-cases and last, but not least, the database for storing owners and vehicle, will be redesigned. Besides the functionality, we will investigate the possibility for implementing a new design for the UI, as we have heard that the new version of Java contains several possibilities for using another design, that are more up to date, and therefore less ninetyish ☺.
From the above, the structure for this chapter will be the following:

- Re-design of the GUI
- Auto complete
- Database finish

4.2 Model the domain - Claus

We have added a new relation between Vehicle and Owner called ownership, regarding to our discussion below

!! At this point we are beginning to lose the bigger picture... Therefore we will do a "full" domain model for the Construction phase, with the knowledge we have at this point. This means, that it isn´t the final version but an early version that should help us maintain the overview of the system structure, relationships etc.

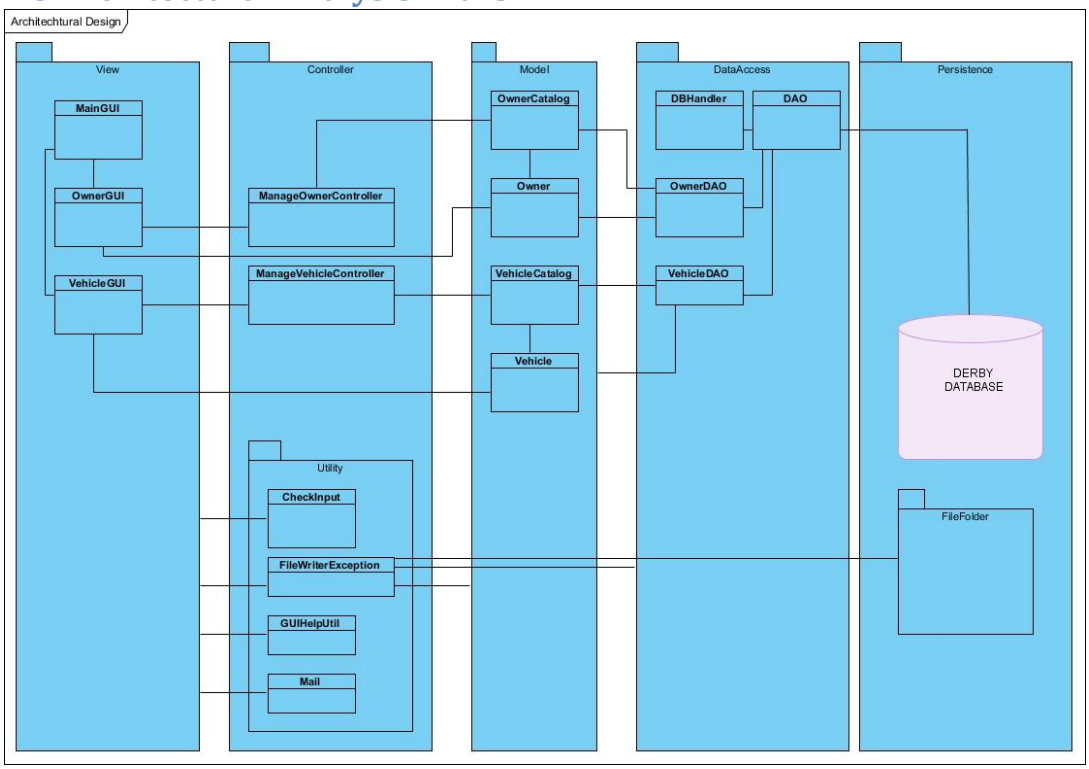
4.2.1 Discussion – Bike vs. Owner

When we made the use cases, we didn't agree on the main focus of the system and company. We thought that the system should be a database with customers that had a bike attached. But some did not agree so we had a discussion about what to build the system around. Actually, there is a big difference of the two different views.

Which is the main focus?
The question is: Should the bike have one owner or vice versa?

Basically the main concern in the company is evolving around the bike, and the bike can have more than one owner in a lifetime. We have decided that the bike can have one owner (at a time). This is because it is the bikes that have the different measurements and generally it is better if the owner is just a changeable attribute of the bike and not the other way around. Implementing this also make it possible the other way around, therefore we have to change our database design, because we now will have a many to many relationship, meaning that we will have to add a new relation called the ownership between bike and owner, telling who is the active owner of the bike, and who has been owners of the bike.

4.3 Architectural Analysis - Lars



We have added the FileWriterException class in the utility package in the ControlLayer. It has a connection to all other layers, to catch and write exceptions thrown. It will write the exception to logFile.txt in the FileFolder layer inside the persistence layer.

Figure 4 1 – The architectural design

4.4 GUI design – Claus

In this iteration we have implemented a new style called Nimbus, instead of the Solid/Metal. All functions are the same as mention in last iteration, but the Look is more “round” and “soft”. See Figure 4 2.

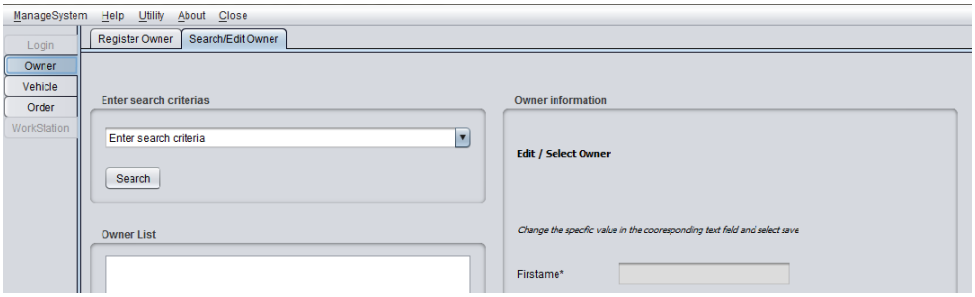


Figure 4 2 – Screenshot of the GUI with the new Look and Feel named “Nimbus”

4.4.1 LoFi Prototyping

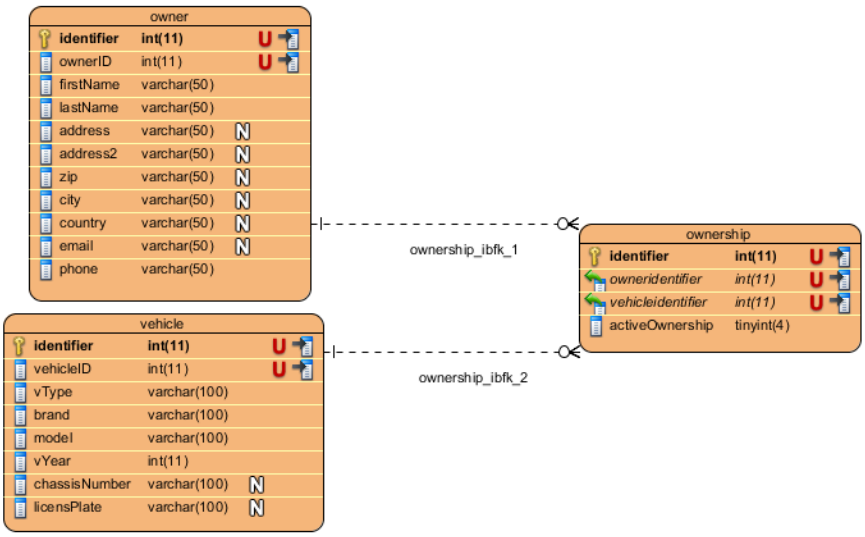
We have done a LoFi paper mockup in the class. All the artifacts are presented with the report. We also did a video of the mock up there can be seen at Youtube - *http://www.youtube.com/user/TeanTalc*. We present the test user for some test cases we want to run through. All in all it was a good exercise for us to see how our thinking about the design of the GUI was understood by others. We had made some happy path test cases, and therefore we didn't get any error feedback, so this is a thing we have to consider for the next test cases. It would be nice to see how user reacts on fail messages.

4.5 Use case design – Lars

4.5.1 Database Design

We have added a new table regarding to the many to many relation between Owner and Vehicle, taking the design to 3NF between these two.

After reviewing our SQL code and database design we have decided that every table should have a Primary key that is "stupid", meaning that it should not be used for anything else that being a primary key. In our old design we have chosen the ID's as primary key, but it could happened that the user of the system would like all the ID's to start with e.g. KJ and then a number(KJ01), with our new design we can change the name on the ID's without have to changed all the primary keys. Regarding to our Autocomplete (see 4.6.2) function in the system, we have chosen to create tables for vehicle type, brand and model, and then connect the vehicle and type, brand, model with foreign keys. For new design see Figure 4 3




```
CREATE TABLE owner (
identifier int NOT NULL GENERATED ALWAYS AS IDENTITY,
ownerID int NOT NULL,
firstName varchar(50) NOT NULL,
 lastName varchar(50) NOT NULL,
 address varchar(50) DEFAULT NULL,
 address2 varchar(50) DEFAULT NULL,
 zip varchar(50) DEFAULT NULL,
 city varchar(50) DEFAULT NULL,
 country varchar(50) DEFAULT NULL,
 email varchar(50) DEFAULT NULL,
 phone varchar(50) NOT NULL,
 PRIMARY KEY (ownerID)
);

CREATE INDEX ownerIndex ON owner(identifier);
```

```
CREATE TABLE vehicle (
identifier int NOT NULL GENERATED ALWAYS AS IDENTITY,
vehicleID int NOT NULL,
vType varchar(100) NOT NULL,
 brand varchar(100) NOT NULL,
 model varchar(100) NOT NULL,
 vYear int NOT NULL,
 chassisNumber varchar(100) DEFAULT NULL,
 licensPlate varchar(100) DEFAULT NULL,
 PRIMARY KEY (vehicleID)
);

CREATE INDEX vehicleIndex ON vehicle(identifier);
```

Figure 4 3 – Mapping of ownership

4.5.1.1 SQL – DDL

Script for creating 3 different tables in the Derby database

```
CREATE TABLE ownership (
identifier int NOT NULL GENERATED ALWAYS AS IDENTITY,
owneridentifier int NOT NULL,
vehicleidentifier int NOT NULL, activeOwnership smallint
NOT NULL, PRIMARY KEY (identifier),
CONSTRAINT ownership_ibfk_1 FOREIGN KEY
(owneridentifier) REFERENCES owner (identifier) ON DELETE
SET DEFAULT, CONSTRAINT ownership_ibfk_2 FOREIGN KEY
(vehicleidentifier) REFERENCES vehicle (identifier)
);
CREATE INDEX ownershipIndex ON ownership(identifier)
```

4.6 Implementation - Claus

4.6.1 Nimbus - New style

We have implemented a new style there is available from JAVA 6.10, and will be fully integrated in JAVA 7. We cannot expect that all can run this new style, therefore we have made a solution, there we try to use the Nimbus style, and if it's not possible it will run the default style.

Below you see the a part of the code for this feature

```
try
  for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels())
    {
       if ("Nimbus".equals(info.getName()))
       {
          UIManager.setLookAndFeel(info.getClassName());
          break;
       } //end if

   catch (Exception e)
      FileWriterException.writeLogFile("Look and feel settings: " + e.getMessage());
   finally
      actScreenSize = tKit.getScreenSize();
      actScreenResolution = tKit.getScreenResolution();
      initComponents();
```

4.6.2 Autocomplete

We have found a new library called SwingX; that have a lot of design functions for the GUI. Until now we have implemented autocomplete, meaning when you start typing a word it auto complete it if it knows it. There is not much code for this, since it is a well-done library, so nothing to show. For further information you should look at https://swingx.dev.java.net/

4.6.3 FileWriterException

Her we have the code for our exception writer, there will write to a txt file and give each exception a timestamp.

```
/**
   * Write exceptions to file for error searching
   * @param logMessage the exception
   */
  public static void writeLogFile(String logMessage)
  {
     PrintWriter pw;
     try
```

```
    {
      pw = new PrintWriter(new FileWriter("logfile.txt", true));
      pw.println(sdf.format(myC.getTime()) + " / " + logMessage + "\n");
      pw.close();
    } // end try
    catch (FileNotFoundException ex)
    {
    } //end catch
    catch (IOException ioex)
    {
    } // end catch
  } // end method writeLogFile
} // end FileWrtierController class
```

4.6.5 Fix defects

We have run all tests and no defect to fix. We have tried to throw some exceptions and the Logfile system is working as expected. See Figure 4 4.

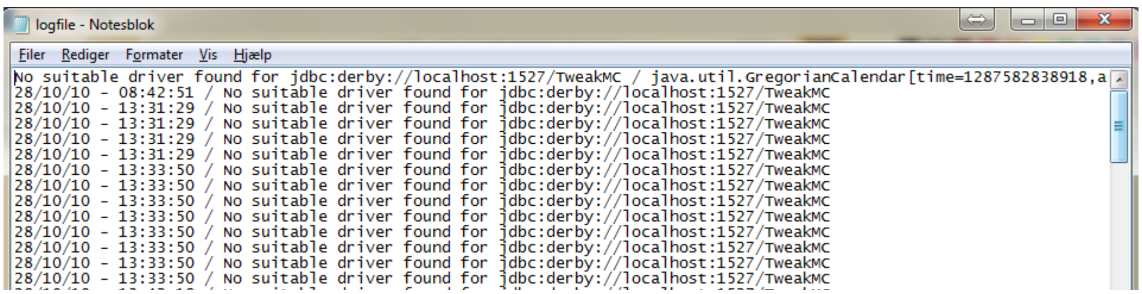


Figure 4 4 – Screenshot from the logfile.txt

4.6.6 Review code

For now our code seems to be good, the frames still are annoying with their sizes. But we have found the problem. We have to add scroll panes and will work on that side running the other iterations. The main functions are now working.

4.7 Revise requirements - Nyvang

Small changes how made us think more (again), it should be possible to add company details; therefore we have a UC called ManageCompany. It should also be possible to connect the vehicle and our workstations in an order so UC Register Order is now a fact. And it should be possible to see who work on that and when. Login and ManageUser UC's are added.

4.8 Part conclusion - all

Regarding to our problem definition we have been around UP, ending our elaboration phase. We reach our second milestone☺. We have done a lot of JAVA programming and found out about GUI design. The database structure is very important; it is a must that we map it right first time. Therefore some have attended to a database course in design to get an overview.

As concluded in the Domain model, we have lost the bigger picture of the system and therefore we will draw a "full" but not final domain model, this should help us to see how to map database and continue our programming. We can ask if we are missing any UC and so on. This has been a big lesson in Project development that we will use from now on: **draw a "full" but not final domain model.**

In our shared vision we agreed in just sharing the roles as they were coming. This still counts, but we must recognize that the Project Manger role must be place at one person, and that one person must fulfill the assignment in being a Project Manager, otherwise the project will run “off-track”.

The risk part has grown; the time is now against us, because of all the time we have used on the GUI. So GUI has become a risk. Despite this, we have decided not to add more risks to the list in chapter 1.8

At the time we think that we have made a very good architectural design, we have our utility class in the controller class, which will handle all conflict, it also help checking the rules for information we will store through the CheckInput class. And we have the possibility to make more helper classes in the utility package.

5.0 Elaboration 3rd iteration – User defined spreadsheet

5.1 Purpose - Claus

This iteration is divided into several use cases 2 “head” and 3 “sub”, see figure 5 1 - UC Diagram.
The user want to create a “spreadsheet” where he can define the size himself and alternative give the headers name. It should be possible to add data in the spreadsheet and save the data for later use. It should also be possible to alter in the data and save it again.
There is 2 alternatives either we could let the user create a spreadsheet in MS or OpenOffice and upload it to the system. Or we could create our own tables and present them integrated in the system. The problem is to save the table, since we don’t know what size and what kind of data we should store. It would be quit a job to create tables in a database and structure them, so we will try to store them as serializable object on the disc.
There are 2 flows in this. Flow 1 define and create the spreadsheet, fill in data and save it. Flow 2 open existing spreadsheet alters data and save it again or delete spreadsheet. This will produce the following use cases, Create and Manage user defined spreadsheet (head use case) this include Create spreadsheet and Manage spreadsheet (sub use case) and Find and Manage user defined spreadsheet (head use case) this include Find spreadsheet and Manage spreadsheet (sub use case). You should also note that the 2 head use cases have an extension that is Manage Result, there is a part of the manage spreadsheet flow, see SD shown in 5.6.1.

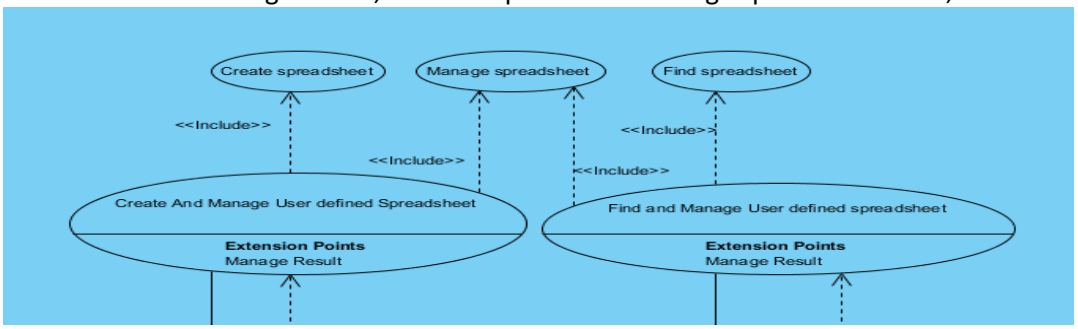


Figure 5 1 – Use case diagram for Spreadsheet

5.2 Model the domain - Claus

We have not added any conceptual classes on the domain model, since the spreadsheet is pure technical issues. They will be presented at the domain model as Result of type media. It has been quite a challenge to figure how this will work.
At first we had thought the spreadsheet as a Result and therefore it was on the domain model, but one thing was drawing another was coding, it would not “fit” together. The end was that, the data inside the user defined spreadsheet was the result, but not the shell spreadsheet.

5.3 Detail use case - Claus

5.3.1 Write use case text

Create And Manage User defined Spreadsheet - Use Case

Use Case Details

Name: Create And Manage User defined Spreadsheet
Rank: Medium
Stereotype: UseCase
Abstract: false
Documentation:

Flow of Events	
	1. Need for creating user defined spreadsheet
	2. User start Create And Manage User defined Spreadsheet
	3. User start Create spreadsheet
	4. System present spreadsheet
	5. User start Manage spreadsheet
	6. User end Create And Manage User defined Spreadsheet

Use Case Details

Level: User
Complexity: High
Use Case Status: Name Only
Preconditions: User have data and know how many rows and columns there should be used
Post-conditions: Spreadsheet with data or empty spreadsheet is saved

Find and Manage User defined spreadsheet - Use Case

Use Case Details

Name: Find and Manage User defined spreadsheet
Rank: Unspecified
Stereotype: UseCase
Abstract: False
Documentation:

Flow of Events	
	1. User start Find and Manage User defined spreadsheet
	2. User start Find spreadsheet
	3. System present spreadsheet
	4. User start Manage spreadsheet
	5. User end Find and Manage User defined spreadsheet

Manage spreadsheet - Use Case

Use Case Details

Name: Manage spreadsheet
Rank: Medium
Stereotype: UseCase
Abstract: false

Flow of Events

Flow of Events	
	1. User starts manage spreadsheet
	2.if Spreadsheet is empty
	2.1. User enters data
	3.else if Spreadsheet is filled
	3.1. User modifies data
	4.else if User want to save empty spreadsheet
	4.1. Go to step 5
	end if
	5. System save spreadsheetdata as Result
	6. User ends manage spreadsheet

Use Case Details

Level: User
Complexity: Medium
Use Case Status: Base
Preconditions: SpreadsheetCatalog exist. Spreadsheet is selected or present
Post-conditions: Spreadsheet is modified or filled or empty and saved
Author: clausbeck
Assumptions: Frequency 300 times pr. Year

5.3.2 System sequence diagram

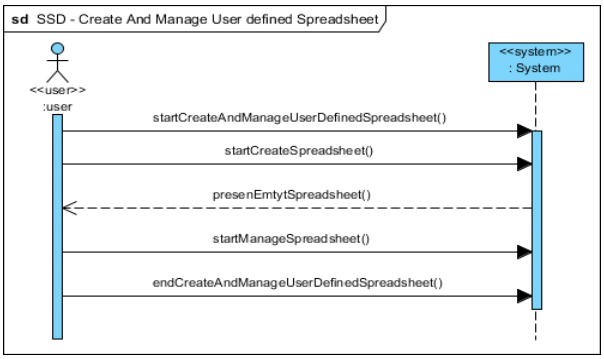


Figure 5 3 - Start for creating a spreadsheet, is taken to Figure 5 4 there you enter rows, columns and alternative names for column, afterwards you will return to here and continue to the Figure 5 5 Manage Spreadsheet where you will enter or edit data, and save (one flow) the spreadsheet data as a result. And return to here.

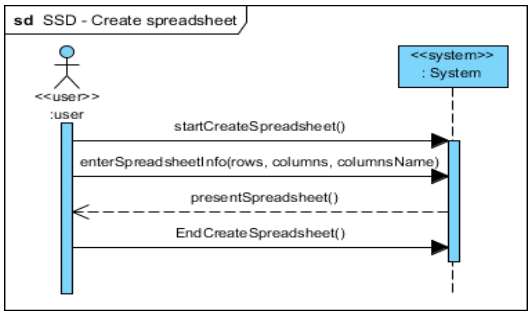


Figure 5 4 - Enter the basic information for creating a user defined spreadsheet.

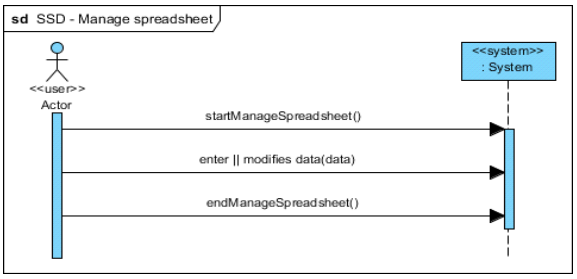


Figure 5 5 – Enter or modify data in the spreadsheet. Or delete data or complete spreadsheet, and return either to Figure 5 3 Create and Manage or Figure 5 7 Find and Manage use case.

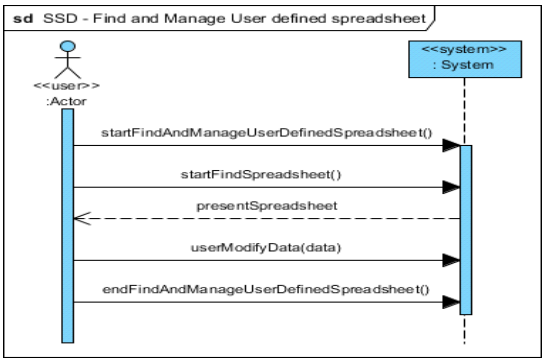


Figure 5 6 – Start for finding an existing spreadsheet, is taken to Figure 5 7 there searchCriterias are set, afterwards you will return to here and continue to the Figure 5 5 Manage Spreadsheet where you will enter or edit data, and save (one flow) the spreadsheet data as a result. And return to here.

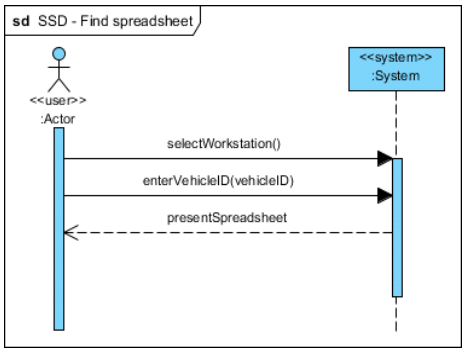


Figure 5 7 – Searchcriteria is set as vehicleID and are found if exist.

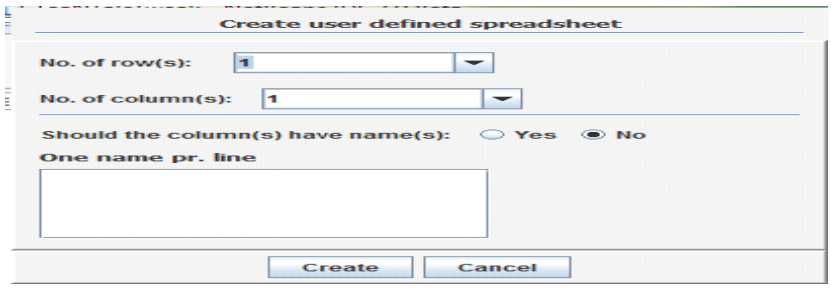
5.3.3 Operation Contracts

Since we have decided that spreadsheet is not a part of the system as itself but pure technical, it will not create any objects. The data from the spreadsheet saved as a Result will create an object, but then the operation contract will be made in the Result use case.

5.4 Architectural analysis - Claus

Spreadsheet have place in the model layer as Spreadsheet and SpreadsheetCatalog, se more in Design class Diagram, 5.7.2

5.5 GUI Design - Torben



As the rest of the system it will be presented in Nimbus style if possible. The spreadsheet itself is just a JTable there will be integrated in the Result GUI. There is one GUI for creating the spreadsheet.

Figure 5 8 – View of Create spreadsheet GUI. It is possible to define how many rows and columns there should be. The combobox is filled with 1-1000, and is auto decorated (meaning that you starts enter the number you want and it will be presented). There is a choice of giving the column names and enter the name(s) in the text area, one pr. line.

If you have chosen to give the column names, and you enter to many or not enough, one of these two error messages will be presented.

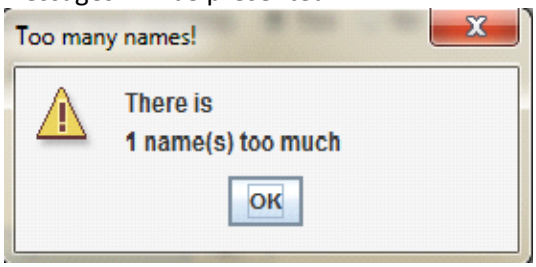


Figure 5 9 – One name to much is entered in the Create GUI

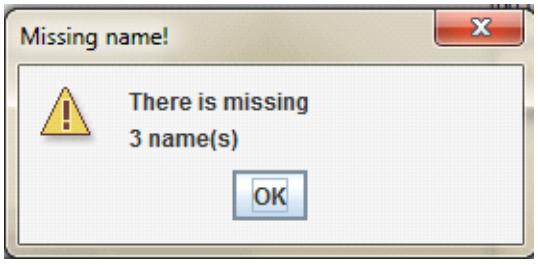


Figure 5 10 – There is missing 3 names in the Create GUI

5.6 Use case Design - Claus

5.6.1 Sequence Diagram (Dynamic View)

See last page (A3 fold out).

What a story the largest SD we have made until now. Here it was important to tread carefully. You start in GUI and go to ManageSpreadsheetController there will pass on basic information to create an empty spreadsheet shell. Now there is the option to save and empty spreadsheet or fill it the data, no matter what task you decide to do the new model in the spreadsheet is a Result. To store it as an object we have to build a new table with the data, and save it on the disc. Since it is a Result we are storing now, we have to call the ResultCatalog, but before the ResultCatalog can create a Result it had to have a path where the spreadsheet is saved, therefore the ResultCatalog calls the SpreadsheetCatalog and ask it to create a new table with the data and save and return a path. The SpreadsheetCatalog creates a new spreadsheet, send it to the ObjectHandler that will save it on a disc in a given folder and with a given name. After what the Result now have all information to create a row in a database table.

5.6.2 Design Class Diagram (Static view)

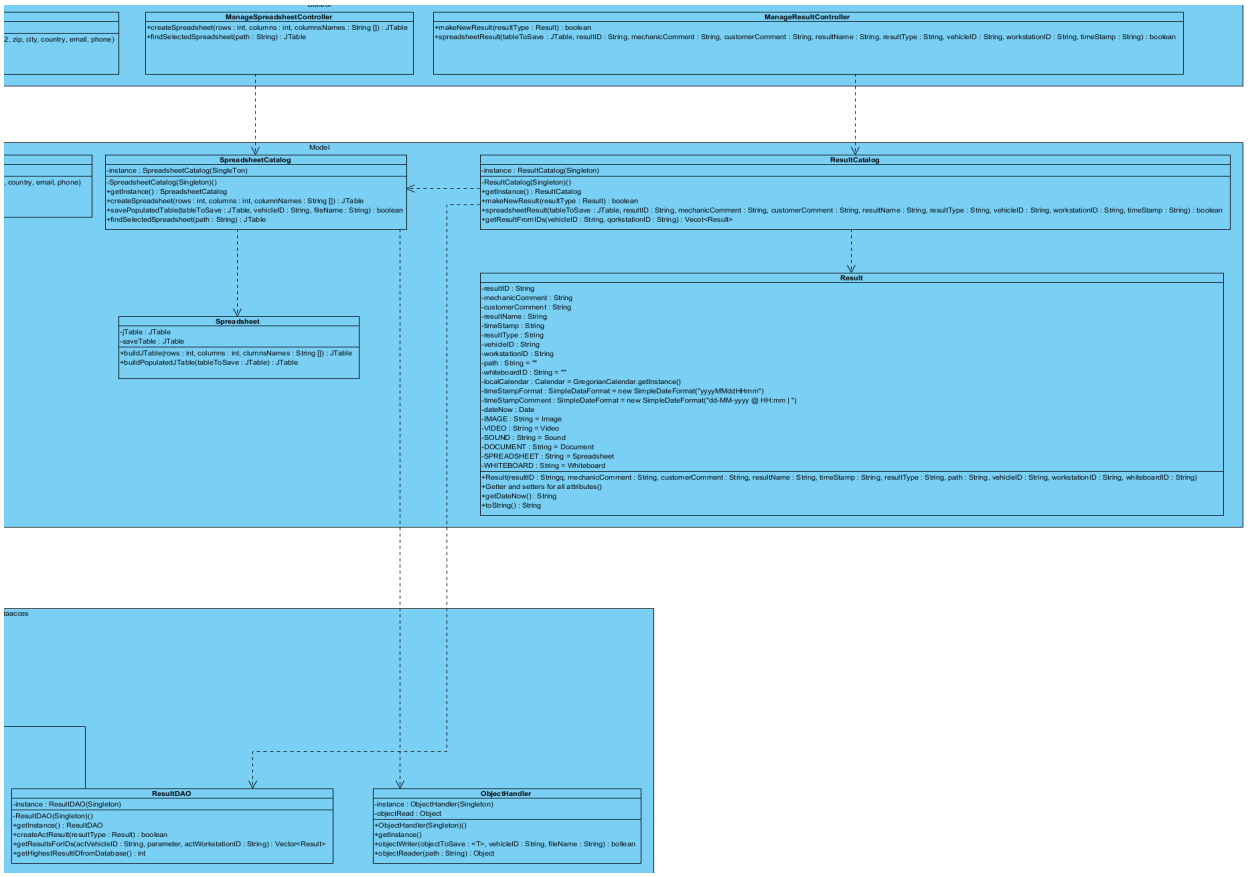


Figure 5 11 – View of the connections between Spreadsheet and Result, and how it is saved in a file and in database.

Here is the picture for how the different classes talk together. It gives a view over how we start creating a spreadsheet and how we can get a Result out of it. As shown the SpreadsheetCatalog is connected with the ObjectHandler in the dataaccess layer, the ObjectHandler will write the Spreadsheet as a Serializable table to a given folder on the disc, this give a path which we will add in the Result. Therefore ResultCatalog and SpreadsheetCatalog is connected. ResultCatalog will talk with ResultDAO that will create a row in the ResultTable in the database. And this is the explanation on why Create and Manage spreadsheet have an extension to ManageResult in the use case diagram.

5.6.3 Database Design

There is no specific table for spreadsheets as they are stored as files on the disc. There are a reference as Foreign Key in the Result Table, that will be described more specific in the Result use case, however here is a short description and overview to see the connection from spreadsheet to Result. This table is taken to 3rd NF

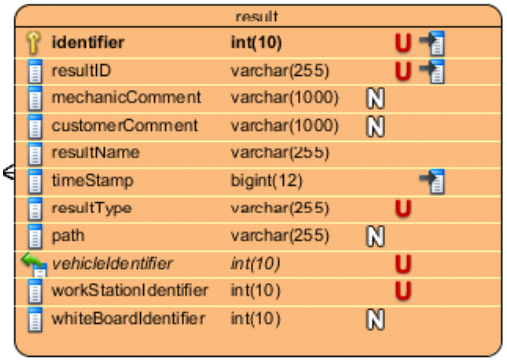


Figure 5 12 Result Table, holds all information about a result. If the result is uploaded (saved file on disc) there will be given a string in path attribute, to locate the file again. If the result is a whiteboard is it identified by a foreign key from a WhiteboardTable (not designed yet. An overview can be seen on the Domain Model) A result is always connected to a vehicle (and order comes automatically through the vehicle) and a workstation, it is only possible to see a result on the workstation it's done on.

5.7 Implementation - Lars

5.7.1 Plan component integration

The first component we will implement is the Spreadsheet and thereafter SpredsheetCatalog. Then comes ManageSpreadsheetController and at last ObjectHandler. This is the first time ObjectHandler will be used and we have decided that it should handle both read and write objects.

5.7.2 Implement component

Here is a part of the most important code for creating a spreadsheet. The hard part is the one that we will create a populated spreadsheet from the data the user have entered.

```
1  public JTable buildPopulatedJTable(JTable tableToSave)
2      {
3          Object[][] data = new Object[tableToSave.getRowCount()][tableToSave.getColumnCount()];
4          String[] columnNames = new String[tableToSave.getColumnCount()];
5          //Save data from table to 2 dimensionel array
6          for (int i = 0; i < tableToSave.getRowCount(); i++)
7          {
8              for (int j = 0; j < tableToSave.getColumnCount(); j++)
9              {
10                 data[i][j] = tableToSave.getModel().getValueAt(i, j);
11             }//end for
12         }//end for
13         //Save columnames to array
14         for(int i = 0; i < tableToSave.getColumnCount(); i++)
15         {
16             columnNames[i] = tableToSave.getColumnName(i);
17         }//end for
18         //Create a new table with the data and names for saving as bytes
19         saveTable = new JTable(data, columnNames);
20         System.gc();
21         return saveTable;
22     }//end buildPopulatedJTable method
```

Figure 5 13 Creating a populated spreadsheet for saving on disc

Line 3: Creating a 2 dimensional array that can hold objects, with the sizes of the rows and columns

Line 6: Initialize how many rows to run through

Line 8: Initialize how many columns in each row to run through

Line 10: Get out the data from the specific cell in the table and store the value (data – that will be a Result) in the 2 dimensional array

Line 16: Saves the headers for the table

Line 19: Creating a new JTable for saving. Note that we call a different constructor than we did when we was creating the empty spreadsheet. Here we call with the parameters data – which is a 2 dimensional array and an array with column names

Line 20: Just because we need a cleanup in our heads after doing this method… (Garbage collector)

```
1  public JTable buildJTable(int rows, int columns, String[] columnNames)
2      {
3          jTable = new JTable(rows, columns);
4          if(columnNames.length > 0)
5          {
6              for (int i = 0; i < columnNames.length; i++)
7              {
8                  jTable.getColumnModel().getColumn(i).setHeaderValue(columnNames[i]);
9              }//end for
10         }//end if
11         return jTable;
12     }//end buildJTable method
```

Figure 5 14 - Creating a new spreadsheet
Line 3: Create the table with the given rows and columns
Line 4: If the user has given names to the columns you go into a loop
Line 8: Running through the loop and set the names on columns (headers)

```
1  public static <T> boolean objectWriter(T objectToSave, String vehicleID, String filename)
2      {
3          ObjectOutputStream oos;
4          try
5          {
6              oos = new ObjectOutputStream(new FileOutputStream("data/" + vehicleID + "/" + filename));
7              oos.writeObject(objectToSave);
8              oos.close();
9              return true;
10         }//end try
11         catch (IOException ex)
12         {
13             Logger.getLogger(ObjectHandler.class.getName()).log(Level.SEVERE, null, ex);
14             FileWriterException.writeLogFile(ObjectHandler.class.getName() + " / objectWriter / " + ex.getMessage());
15             return false;
16         }//end catch
17     }//end objectWriter method
```

Figure 5 15 – Writing ex. a Spreadsheet to a file
Line 1: We have made the type generic (<T>) so we can send any kind of object to the writer and save it as a file
Line 6: Here we give the file a name and a directory to save it in. It is saved in the data folder, where every vehicle has its own folder that is created when you register a new vehicle.
Line 7: The object is written to a file
Line 14: Our fail log, here we call the method in the utility package and write fails in the Logfile. It is logged with a date and time and the class + method + fail message

5.7.3 Perform JUnit test

After performing JUnittest it shows that we haven't got the user defined headers with us in the populated spreadsheet.

5.7.4 Fix defects

The reason for the problem mentioned in the last section (5.7.3) was that all data is in the model of the table (spreadsheet) and therefore we assume that the headers were there too. But it is only the headers the system had set that are in the model. The user defined headers are in the shell (JTable). So we correct the code to get headers from there instead (if any was set).

5.8 Revise project plan and risk list - Adams

This part took as long time as planned as there were, no problems with storing the spreadsheet as an object there haven't been any changes to the Risk list.

5.9 Revise requirements - Torben

We have to make some new thoughts about the Whiteboards. Since the spreadsheet are going to be the cornerstone for these. Whiteboards and spreadsheet in them self are not Results, the content of these are Results, therefore we have to change that they are sub classes to Result.

5.10 Part conclusion - All

This use case was one of the “hard nuts”, if we didn’t success to store the spreadsheet as an object; we have to come up with whole new ideas for the program.

There are a lot of challenges in making spreadsheets. Was it possible to create a spreadsheet from time to time with a user defined size? And was it possible to store the data the user enters afterwards? Since we can say yes to these questions we must conclude that we have learned something about programming objects and store them as serializables, and therefore we have used Java and UP /UML, learned some new things in JAVA, minimized the risks (since there were no problem in saving) and used our classes across the layers from GUI to dataaccess.

6.0 Construction 1st Iteration – Send Helpdesk Email - Nyvang

6.1 Purpose

In this iteration we will model the UC14 Send Helpdesk Email.

Because of the relatively large number of iterations we have to complete, we have decided to give a very brief introduction and a brief presentation of the diagram changes, for the remaining iterations.

6.1.1 Considerations

As we are working our way through the project, we have come up with some new ideas for the system. The specific idea relevant for this chapter is an error reporting system that allows the user to send a technical question to the programmers of the system. Earlier in the process we talked about making a hotline for the company, but we couldn't see how it should work, as we all generally are in school when the system is typically used. Now, on the other hand, we have come up with an idea of making an e-mail based support/error reporting system. As we know that it's impossible to develop a flawless system, and furthermore to identify and correct all possible bugs and/or errors. And it makes little sense to deliver a piece of somewhat complex software without any support. It would not make sense to deliver a server to a company without any support and/or service either.

In addition to the above, we will now discus if there´s any other ways we can benefit from implementing the use case. When we are about to begin the user test with our test team we can use this function for generating and maintaining a standard on the feedback we will get from the testing activity. We can change the method slightly, so it contains some standard questions and test-use cases for our testers to answer/try. For example, we can create a standard form of the email with a use case like, "try to do something that are relevant to one of the other use case, and then write your thoughts into the error "feedback" reporting system". If we conclude this in the function, we can get some standard feedback answers and therefore the feedback can be compared to other answers, which will give us a much better overview of the tester’s experience. Another benefit we can take advantage of is that at the same time we will have a through testing the e-mail system.

Therefore the decision is that another use case is added at this point. This use-case will be called: UC14: Send Helpdesk Email and will be a part of the 1st iteration in the construction phase, so we can start using the modified version as a method for getting standardized feedback from our testing team.

6.2 Model the domain

Nothing new since the email function is purely technical and therefore it doesn't influence the domain model.

6.3 Detailed Use Case

6.3.1 Write use case text

Show the fully dressed use case of UC14: Send Helpdesk Email

UC14: Send Helpdesk Email
Scope: TweakMC
Level: User goal / System engineer goal
Primary actor: User
Stakeholders and interests:
User is experiencing technical issues that requires external help
Preconditions: User has issues that cannot be resolved by himself and therefore he needs to be escalated. Log file and internet connection must be available
Post conditions: Error report has been sent as an Email
Main Success scenario:

1. Error symptoms are identified by user
2. User starts the help menu and selects "Report technical issue"
3. The symptoms are described into the textbox
4. System adds the log file as an attachment
5. E-mail is sent from a predefined email address, to another predefined address
6. User ends send helpdesk email

Special requirements: Log file and User to memorize the error symptoms and can describe them so others can understand them. Alternatively, log file has to be present when sending the auto generated error report
Frequency of occurrence:
Every time user is stuck and the system not reacting as expected.

6.3.2 System Sequence Diagram

See Appendix H

Figure 6 1 – Screenshot from the Helpdesk UI

6.4 Architectural Analysis

The mail function is put in utility package in the controller layer. It is a help function for the system and the developers. See Appendix G

6.5 GUI design

The error reporting mail function will be a separate JFrame pop up dialog, with possibility to add the issue and

the sender's mail address, if the user will have a copy. All GUI-standards are the same as in the main system. See Figure 6 1

6.5.1 HiFi prototyping

We have visited the company KJMC to show them the first example of the system, and to have a talk with them about some issues we have about understanding their expectations to how the tables for e.g. measurements should be created, presented and stored.
The response we got from them was that we are moving in the right direction.
We have recorded the test of the system so we always can go back and see the users' behavior to the system.

We have gained a lot experience about the company expectations to the system compared to both our programming experience and our expectations to the final system.
We have now knowledge about how to design and implement the tables to meet the company's expectations.
It gives us challenges to meet those expectations – but hey, it is a study project and we love a great challenge☺

There is a video clip from the HiFi prototyping on Youtube - *http://www.youtube.com/user/TeanTalc*

6.6 Use Case Design

We have imported a special library containing the options we need to send an email from our system.

6.6.1 Sequence Diagram

See Appendix I – Send helpdesk Email

6.6.2 Design Class Diagram

The mail class is added in the utility package, attributes and operations are shown. See Appendix J

6.7 Implementation

This class only contains one method which is mail() returning nothing. The special part in this code snippet is that some on the instance fields are public and static, it is MESSAGE and SEND_TO. This means that we can set the message and send to address without setter methods in the code. For full code see Appendix L.

6.7.1 Review code

Building an error report sent by email another time, we will maybe consider to make the mail method static and with 2 parameters; MESSAGE and SEND_TO. This will lead to good JAVA standard and keep all instance fields private. This will also give us the opportunity to call the method directly without making a new object of the class, and therefore much more flexible to use in rest of the system.

6.8 Part conclusion

This iteration has been made by one member of the team; Nyvang, and therefore he have the most knowledge of this part of the system. The rest of the team members now know there's a simple way of sending mails from the system and therefore we have all learned some new things in programming.

The project management role is now in place and is "played" by one person for a week at a time. This has given a lot more structure and keeps the project on track. We have now printed big copies of the domain model and database design, and put it on the wall next to our office. When there are any doubts or just discussions we all gather around these diagrams, where we can paint and change name on attributes etc. This has given a bigger picture for all in the team.

7.0 Construction 2nd Iteration – Search Owner & Manage Owner

7.1 Purpose - Lars

In this chapter we will design and implements the feature to search an owner.
The use-cases we are modeling and implementing is

- UC 8: Manage Owner
- UC 5: Search Owner

In addition, we will look at the JavaHelp system and implement it in our system.

7.2 Detailed Use case - Torben

See Appendix B

7.2.1 System Sequence Diagram

See Appendix H

7.3 Use case analysis - Adams

It is definitely a relevant use-case for the system, because the user wants to search for an owner to see his or hers information.

7.4 Use case design

7.4.1 Sequence Diagram

See appendix I

7.4.2 Design Class Diagram

The design class diagram has been updated with the new operation Search Owner (see appendix J – DCD: Construction 2nd + 3rd Iteration)

7.5 Implementation - Lars

A new thing here is the DDL (Database definition language) for search owner.

It looks like this (for Full code – See Appendix CODE):
(? = Preparedstatement)

```
("SELECT * FROM OWNER
WHERE LOWER(firstName) LIKE LOWER(?)
OR LOWER(lastName) LIKE LOWER(?)
OR LOWER(address) LIKE LOWER(?)
OR LOWER(address2) LIKE LOWER(?)
OR LOWER(zip) LIKE LOWER(?)
OR LOWER(city) = LOWER(?)
OR LOWER(country) LIKE LOWER(?)
OR LOWER(phone) = LOWER(?)
OR LOWER(email) LIKE LOWER(?)");
```

Another very cool new thing here is that we have chosen to use Generic search criteria. This gives the user opportunity to search for an owner with any kind of character.
Here is the method signature in the OwnerDAO for searchActOwner (for full code see Appendix - CODE)

```
public <T> List<Owner> searchActOwner(T searchCriteria)
```

At first we thought about making the search function look like the registration (a search field for each variable on e.g. an owner). This can, however, make the user confused because of all those fields to fill. Therefore we chose the generic type of search which is much more user friendly. Besides it makes the system handle less search criteria at a time. When using the generic version only a single variable (search criteria) is passed through the architectural layers.

7.5.1 Review code

We have found out that it is a very good idea to make methods for search with generic types.
The only hard thing is that we have to cast every search criteria to the right data type.
We have made the search method so that it returns a List with all the results as objects.

7.6 Part Conclusion - All

As the Owner use cases are live, we have reached another important part of the system. But what is more interesting (in the learning goal) is that we have learned how the JavaHelp works and we have implemented it in the system. This is actually a very nice feature because it involves several different programming languages like XML*, HTML*, C++ and Java.

At this point we can see that our project plan will not hold (as expected). Therefore we have made a new plan which can be found in Appendix P.

** These are actually markup languages, but they do need to be coded although.*

8.0 Construction 3rd Iteration – Search Vehicle

8.1 Purpose - Adams

This chapter will only briefly go through the subjects which are much like the previous chapter. New things will be described in details.

For documentation, see appendix B; use case 10 – Search Vehicle + Appendix B – SearchVehicle Diagrams. (For Full code – See Appendix CODE):

In this chapter we will design and implement the feature to search for a vehicle.

The use cases which we are modeling and implementing is

- UC 9: Manage Vehicle
- UC 10: Search Vehicle

8.2 Detailed Use case - Claus

See Appendix B

8.2.1 System Sequence Diagram

See Appendix H

8.3 Use case analysis - Claus

It is definitely a relevant use-case for the system, because the user wants to search for a vehicle to see the information about it.

8.4 Use case design - Adams

8.4.1 Sequence Diagram

See appendix I

8.4.2 Design Class Diagram

The design class diagram has been updated with the new operation Search Vehicle (see appendix J – DCD: Construction 2. + 3. Iteration)

8.5 Implementation - Torben

In this DDL we have something new. We need to take into consideration that database tables are constructed with foreign keys in some of the coloumns. These columns refer to e.g. vehicle brand in another. This was used earlier for autocomplete implementation and now we need a joint statement in order to return a resultset which consists of the data, we need and not an identifier for another table (if we didn't consider this, a searchresult would simply be presented to the user with numbers instead of e.g. brands – this wouldn't be user-friendly although it would work)

It looks like this *(for Full code – See Appendix CODE):*

CODE: SEARCH VEHICLE (? = Preparedstatement)

```
("SELECT v.VEHICLEID, vt.SUGGESTIONS AS vType, vb.SUGGESTIONS as brand,
vm.SUGGESTIONS as model, v.VYEAR, v.CHASSISNUMBER, v.LICENSPLATE
FROM VEHICLE v, vehicleType vt, vehicleBrand vb, vehicleModel vm
WHERE (LOWER(vt.suggestions) LIKE LOWER(?)
OR LOWER(vb.suggestions) LIKE LOWER(?)
OR LOWER(vm.suggestions) LIKE LOWER(?)
OR LOWER(chassisNumber) = LOWER(?)
OR LOWER(licensPlate) = LOWER(?))
AND
(v.vtype = vt.identifier AND v.brand = vb.identifier AND v.model = vm.identifier)")
```

For the explanation we will examine this beginning with each of the highlighted words:

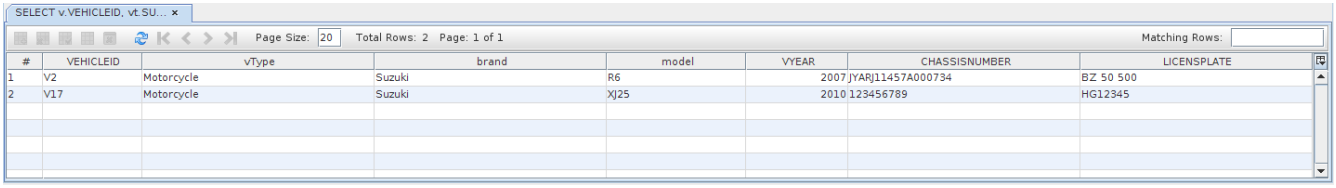
Select: In this part we collect the columns which we'll need. These are:

- vehicleID (v.VEHICLEID) from the vehicle table
- vehicleType (vt.suggestions) from the vehicleType table
- vehicleBrand (vb.suggestions) from the vehicleBrand table
- vehicleModel (vm.suggestions) from the vehicleModel table
- year (v.VYEAR) from the vehicle table
- chassisnumber (v.CHASSISNUMBER) from the vehicle table
- licensplate (v.LICENSPLATE) from the vehicle table

FROM: In this part we select which tables we're going to use:

- vehicle
- vehicleType
- vehicleBrand
- vehicleModel

WHERE:
This is where we put in the searchCriteria. By using the LOWER() on both the searchCriteria and the result there's not distinguished between lower and upper cases.
AND:
The last part avoids multiple identical results:
If we search for Suzuki (and its present in the database), an output would look like this:


SELECT v.VEHICLEID, vt.SU... ×

Page Size: 20 Total Rows: 2 Page: 1 of 1 Matching Rows:

#	VEHICLEID	vType	brand	model	VYEAR	CHASSISNUMBER	LICENSPLATE
1	V2	Motorcycle	Suzuki	R6	2007	JYARJ11457A000734	BZ 50 500
2	V17	Motorcycle	Suzuki	XJ25	2010	123456789	HG12345

8.5.1 Review code

The earlier decision to make type, brand and model as foreign keys in other tables makes a little more complicated to extract the data which we want for a search.
We have made the search method returning a List with all the results.

8.6 Part conclusion - All

We now have the ability to search for a vehicle from the database like we can search for a user. In these two use cases we found that generic programming can be very useful when you just learn how it works. It is very efficient to use the generic search method instead of (roughly) using a search method for each field/attribute. We improved the design a great deal in contradiction to the other methods where we should have had more methods that are very much alike. This also reflects our goal about high cohesion and low coupling as we have fewer methods. But even though this method is quite handy, it can be improved. Therefore we have decided to develop a "SuperSearch"-method which should be able to search on everything, wherever you call the method in the system. We have chosen not to make a use case, as almost all methods are present in the already implemented search functions and therefore the only new class we are adding is a new DAO called SuperSearchDAO that handles all the enquiries from the user.

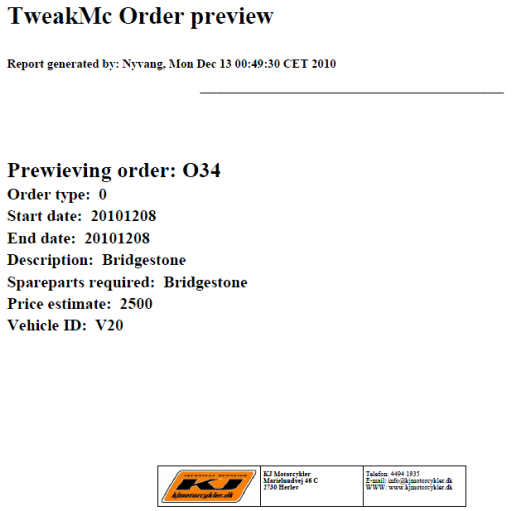
9.0 Highlights

In this final chapter we have chosen to show some of the highlights of the system. As we have made an "almost" complete system we are under a lot of pressure regarding the number of pages allowed. Therefore we have decided to include some of the highlights you can find in all the diagrams and code in the appendix.

9.1 Print order as PDF – Nyvang

The company needs a way of presenting the reports that are created of measurements from the different stages. These stages, however, haven't been completed yet. Instead we can present the function of printing an order from the UC19: Display ToDo List. The OrderPDF class in the Utility layer receives the order information from the OrderGUI and inserts these in a new PDF together with KJMC-logo and contact information. Some of the code is explained below:

```
... code omitted...
private static final String FILE = "KJMC_OrderPreview" + ".pdf"; //specify
```


TweakMc Order preview

Report generated by: Nyvang, Mon Dec 13 00:49:30 CET 2010

Prewieving order: O34
Order type: 0
Start date: 20101208
End date: 20101208
Description: Bridgestone
Spareparts required: Bridgestone
Price estimate: 2500
Vehicle ID: V20

KJ Motorcykler
Marienhøjvej 46 C
2730 Herlev

Telefon: 4494 1935
E-mail: info@kjmotorcykler.dk
WWW: www.kjmotorcykler.dk

Figure 9 1 – The generated Order PDF

```
filename
    public static String orderID = ""; // the Order attributes are set from the OrderGUI
    public static String orderType = ""; // the Order attributes are set from the OrderGUI
    public static String startDate = ""; // the Order attributes are set from the OrderGUI
        … code omitted…
    Document document = new Document(); // create a new object (object reference) of iText.Document
    // call the PdfWriter method from iText      with the document and FILE as attributes
    PdfWriter.getInstance(document, new FileOutputStream(FILE));
   document.open(); // opens the document before writing
    addMetaData(document); // add MetaData, which are the document properties for a PDF file
    addTitlePage(document); // add content, which are the Order attributes specified by the OrderGUI
    document.close();  // opens the document before writing
    PdfPTable table = new PdfPTable(3); // create a table to contain the logo and contact info
        table.setWidthPercentage(43); // specify width of table in percentage of the full  page width
            table.addCell(new Jpeg(imageURL)); // add the logo from a specific location
            table.addCell(new Phrase("KJ Motorcykler\n"
                + "Marielundvej 46 C\n" // add contact info
                + "2730 Herlev\n" // add contact info
                , logoFont)); // chose used font (specified in instance variables)
         … code omitted…
```

This was a selected section of the OrderPDF class that generates the PDF file. Example of the created PDF can be found in the Appendix R or a small version in image 9 1. The full code can be found in the code appendix.

9.2 The DatePicker - All

When an order is created, we need 2 different dates. The easiest way is to let the user type a date manually into the text field, but with this method follows a very big risk for miss-typed dates which are hard to check with regular expressions. Therefore we are in need of another solution, when we made the autocomplete in the 2nd iteration of elaboration. We came across a demo from SwingLabs who have a solution for precise this problem. The solution is the DatePicker which allows the user to open a small JFrame with a calendar to choose a date from. This date is then automatically displayed in the relevant text field. The same applies to the other date (expEndDate). When we have these dates, we can easily compare them and make sure that e.g. the startDate is before (or lower) than the end date, etc. This is done by casting the date to a "long"-data type which is much easier to handle in calculations. These dates are also used for generating the data in UC19 Generate ToDolist, where we calculate the date in seven days (in milliseconds) and returns all orders that are starting or ending within this time period. Code example follows:

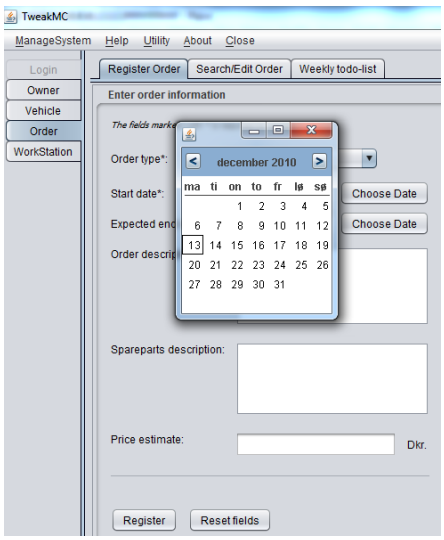


Figure 9 2 – The JFrame that displays the DatePicker function

```
…code omitted…
    final JFrame frame = new JFrame(); // create a new final JFrame
    final JXDatePicker picker = new JXDatePicker(new Date()); //initializes the datePicker
function
    Calendar calendar = picker.getMonthView().getCalendar(); // initialize a new Calendar reference and use the month view
    picker.getMonthView().setLowerBound(calendar.getTime()); // sets todays date as default date when the frame is set visible
    JXMonthView monthView = picker.getMonthView(); // initializes the month-view in the frame when set visible
    monthView.setPreferredCols(1); // specifies the rows and columns of the date frame (e.g. number of months to show in the frame)
    picker.addActionListener(new ActionListener()
      public void actionPerformed(ActionEvent e) // create a new Action listener in an inner class
        SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd"); // formats the date to a SQL-freindly format
```

```
        textField.setText(sdf.format(picker.getDate())); // formats the date to a SQL-freindly format
        frame.dispose(); // close the frame, and only the frame on close (instead of both this frame and the system frame, which is default)
    frame.getContentPane().add(monthView); //as we are using a Swing-JFrame (and not a AWT-JFrame), we need to call the method .add from the getContentPane
    frame.pack(); // pack the frame and components
    frame.setVisible(true); // set the frame to visible (display the frame to the user)
…code omitted…
```

Besides the Create Order use case, the DatePicker is also used in the Manage Order use case. The full code can be viewed in the code appendix.

9.3 SuperSearch - All

The generic search methods are used for searching with unknown search criteria's. We decided to gather these methods in a single GUI that can be called from anywhere in the system. This will support our goal about cohesion and coupling as we are gathering methods that are related in the same class. As you see in figure 9 3, the SuperSearchGUI are asking what to search for, and what kind of information you have available.

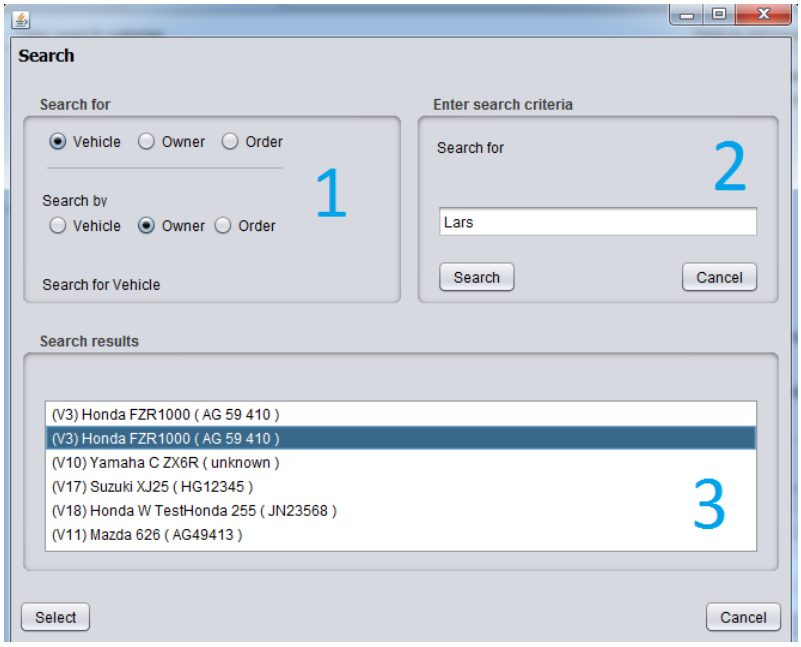


Figure 9 3 – The SuperSearchGUI while searching for a vehicle using owner info

In this case, we want to find a vehicle by using owner information. Therefore we start by selecting Vehicle by Owner in nr. 1. Second we type the search criteria, which in this case is "Lars" this is nr. 2. After pressing search, we will find all vehicles that are attached to an owner who have the name Lars (as a part of his name) which are nr. 3. A simple search in the database will show weather the search results are correct. Because we cheat a little, we know that Lars have the owner Identifier "19". Therefore we can do a "select * from tweakmc.ownership"-statement to get all the entries in the table, sorted by the identifier that we are searching for. The results are as follows:

#	identifier	owneridentifier	vehicleidentifier	activeOwnership
5	17	14	20	1
6	5	19	3	1
7	9	19	3	1
8	14	19	10	1
9	16	19	17	1
10	18	19	18	1
11	20	19	11	1
12	15	23	14	1
13	21	25	27	1

Figure 9 4 – A query in the database asking to show all entries in the Ownership table

A simple count shows, that the owner with the identifier "19" (Lars) have 6 vehicles attached, which is precise what the SuperSearchGUI are showing in figure 9 3. The code for this search is quite complicated and a little too large to insert into the report, however it can be found in the Appendix – CODE.

9.4 The JavaHelp system - Nyvang

The JavaHelp is a well-known add-on from Oracle, and it´s used for making a help function for different applications. The system in itself looks a lot like the standard help function that is in a wide variety of applications, like some of the older Windows versions.

9.4.1 Architectural placement

As we don't really need a lot of Java code for the system to work, diagrams aren't relevant. And as the user only can press one single button (F1) to enter the help system, there aren't really use for a stand-alone use case. Actually, only one method is required for creating and executing the system. Therefore we had some thoughts about the right placement in our architecture. It could be placed in the model layer because that is where the fundamental functions are executed. But then again we could place it in the utility package as it easily can be categorized as a utility. However half of the function in the Class is used for creating a GUI, dressed as a JFrame. And there are no calls through the different layers of the rest of the system, we have decided to place the package inside the GUI (view)-layer, but in a package for itself, like: tweakmc.view.javahelp.

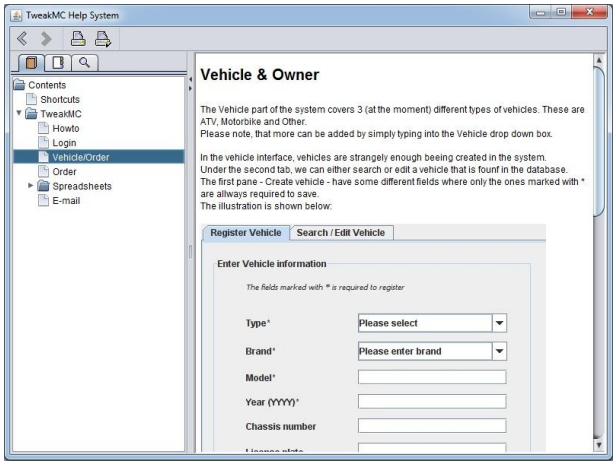


Figure 9 5 – The javaHelp UI

9.4.2 How does it work?

The JavaHelp system consists of 6 different core components. These components will be briefly described below. A graphical overview of the part system is found in figure 9 6

1) **The Help File(s)** – Index.html & *.html. The first component is the help file(s) in itself. These are created in ordinary plain html where you can insert pictures, tables and so on. My curiosity led me to insert a simple dynamic html script, written in JavaScript, a slideshow actually. However it´s not allowed to insert JavaScript's in the help files, but at least I tried.
2) ***The helpset file*** –jhelpset.hs (XML) The helpset file is the "manager" of the remaining components, we can compare it to a roundabout with signs that connects different roads, and the signs show where to find the different components. The jhelpset file is also the only file that is read by the HelpViewer, which is loaded in the single java class. In our project this class is called: JavaHelp.java
3) ***The map file*** – jhelpmap.jhm (XML) The map file maps (or specifies the location of) the different help files, that is the html files (see "url" in the example below). It also links the html files to specific id´s (see "target" in the example), so the system knows when and where to show the different html files.
Example: <mapID target="Simple.Introduction" url="index.html"/>

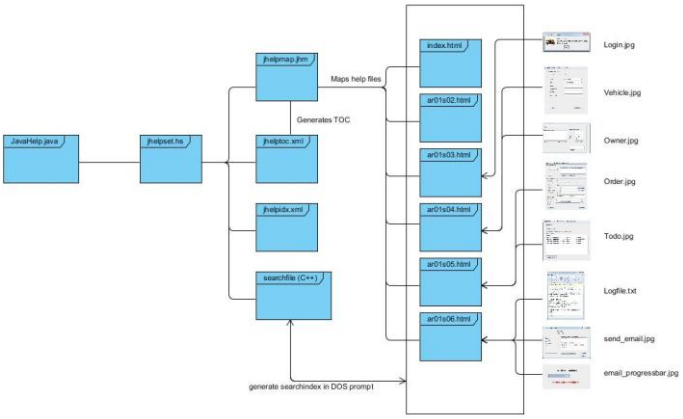


Figure 9 6 – The javaHelp system

4) ***The Table Of Contents*** – jhelptoc.xml(XML) The ToC uses the mapIDs target to a header, that is displayed to the user. There´s really nothing special about this one, as it is an ordinary Table of contents file, like we know it from ex. Word.
5) ***The index file*** – jhelpidx.xml(XML) The index file maps the id of a file to a specific html file. The example below maps the id "id47474747" to the specific section of this html file "index.html#teantalc". This means that we can link directly to ex. the section "tean talc" of a long html file, instead of just linking to the file itself. This is not really used in our project, as the help files are relatively short
   Example: *<indexitem text="teantalc" target="id47474747"/>*
6) ***The search data.*** One of the big advantages of using the JavaHelp system instead if building one ourselves, is the search feature. Like the similar help systems we mentioned before, the search feature is very effective as it creates 4 different files in binary format that contains info about all words used in the *.html files, that is our actually help files. When this is done we can – in principal matters – search for any word used in our help files and we will have a list over the different words and the page number(s) on which they appear. The data is created by a little program that comes with the package from Oracle named "jhindexer.jar". The commandline for generating the files is shown below in an example: (jhindexer "sourceFileDir*" *.html)*
   Example: *%JHELP_CLASSPATH%\JavaHelp\bin\jhindexer.jar c:\...\NetbeansProjects\JavaHelp\src "filename".html*
   Figure 9 6 shows the connections between the different components in the JavaHelp system

9.4.3 The Java Class

In the following section we will explain the code of the single Java Class, used for creating and calling the JavaHelp system. But before we can get anything working, we need a library. This is luckily included in the JavaHelp Technical Development Kit 2.0-package, which we downloaded from Java.net
https://javahelp.dev.java.net

9.4.3.1 code

```
public class JavaHelp
…. Code omitted
    JHelp helpViewer = null;
        ClassLoader cl = JavaHelpTest.class.getClassLoader();// Get the classloader of this class.
        // Use the findHelpSet method of HelpSet to create a URL referencing the helpset file.
        URL url = HelpSet.findHelpSet(cl, "jhelpset.hs");
        helpViewer = new JHelp(new HelpSet(cl, url)); // Create a new JHelp object with a new HelpSet.
        helpViewer.setCurrentID("Simple.Introduction");// Set the initial entry point in the table of contents.
    // create a new frame, set size, add the helpViewer to the contentPane, set the right close operation and
finally set the frame to visible
…code omitted
```

10.0 Conclusion – All

We chose to develop a full functional system that should be used in "real life" in KJMC. Even though we have come a long way, this goal, however, has not been achieved within the time frame, but we are determined to keep on developing until release.

As we were following the UP quite strictly, we came across a discipline that, after our opinion, was a little misplaced. In this case we are referring to the placement of the domain model. As we described within the report we, at some point, lost the bigger picture of the domain. The UP tells us that the domain model should be build bit by bit as we finish the different iterations, but we have experienced that when a system grows above a

certain size, this is nearly impossible. Therefore we chose to model the domain in a single diagram so we could have a central overview of the system domain. This helped us a lot, and we are definitely going to use this method in future projects.
One of the great challenges was to understand the company's requirements; this wasn't because their description wasn't good enough, but more because of our inexperience asking the right questions. We actually first learned their full idea after the mockup and the second visit where they tested use case 1 and 2. Again we can refer to the domain model that we should have made right away to get the bigger picture, instead of just making the different use cases. When we did the domain model we finally understood the full and very complex requirements.
Regarding the UP and UML in general, we have found that a complex system is nearly impossible to create without using the UML. When we were doing especially the spreadsheets we came to the conclusion that the code was way too complex for a simple inefficient human brain like ours, and if we hadn't created the UML diagrams in advance, we could not have done the code.
As with the code and diagrams, the database became equally complex. When we were creating the database, we discovered how important it was to draw very detailed diagrams to map from. But we found that the database is a very powerful tool for storing data in an efficient and logic way. We had some issues with the Derby database as it was a local database, and regarding to the development phase it wasn't a good solution when we are 5 team members working in it simultaneously. Therefore we chose to use MySQL while developing the system so all members had the same version and data - always.

In the process we have used different ways to grow from a group to a team, and our hopes were to become a jelled team. We are convinced, that our efforts with socializing outside the school has been a success, and that we have gained a very good team spirit and maybe even a jelled team.
We started by setting some very high expectations to ourselves and agreed on a relatively strict group agreement. This was, as we see it, a good way to keep everybody committed and focused on the task at hand.
The difficulties regarding the personal learning and planning was primarily the planning of the project as we experienced that almost every discipline contains unforeseen issues that we just haven't thought about. First of all because of our experience, but also because software developments simply just are a profession where loads of issues will occur, because the subject is so big that no one can know all the techniques. Therefore every new project will contain non-explored techniques and areas.

One of our main goals was to achieve new knowledge about the Java language. We have tried to explore the language and read up on the newest technologies and features. We had many ideas about features and how to make the system more useful, these were primarily ideas that came up while reading up on some of the cool functions in Java when we were doing the project. Kind of like a parallel brainstorm. The functions were mostly developed by the person who got the idea so we could use the thoughts we had from the beginning (of the idea). By doing it this way, each of us got "our own projects" and we could expand our knowledge on this specific area. This method is useful when you work in a team that have different interests and different skills within the Java language, a very important strategy for the personal learning of the individual team member instead of sitting in a group where one or two are coding the whole thing - often you learn much more by reading, trying and solving a problem yourself.

We chose to use the MVC-structure to organize our layers and classes. This method was very useful when we used the "standard" layers when working with a database (Model, View, and Control). We encountered some issues when making the Utility-layer. We had our doubts about the placement of the new layer as this wasn't in

the MVC theory. We had a discussion about the placement and we of course agreed that it wasn't in the view-layer, but the discussion were weather it was closer to the model or the control layer. At the end we thought about the functions of the classes inside this layer, and agreed to place it within the control layer.
A very important part of developing software is the design patterns about: Low Coupling and High Cohesion. These are important for maintaining and developing the software in the future. This had a high priority for us as the pattern is an essential part of developing new maintainable software. As they generally follow each other we had the theory in our heads when designing the layers, classes, and methods. We have deviated from the patterns a few places "just to make things work" despite the fact that it was bad design. From this we can conclude that following a design pattern to the dot isn't always the most appropriate way of coding and designing.

Throughout the report we had a vision about making a connecting line so we could tell a story to the reader about the process we have gone through in the process. In the end of each chapter, we stopped and looked back on the previous chapter and the problem definition, to make a part conclusion regarding the goals we had achieved so far. This was a very good thing for ourselves as well as the reader, because it helped ourselves to use the artifacts we had created. In the beginning of the project we didn´t think the part conclusions could help ourselves as much as it did.

For our personally evaluations see Appendix A

11.0 Bibliography

11.1 English Literature

Ford, William H – Topp, William R

- Data structures with Java, Prentice Hall , 2005

Kirfer, Michael – Bernstein, Arthur – Lewis, Philip M

- Database Systems, Pearson Education, Inc. 2005

Deacon, John

- Object-Oriented Analysis and Design, Addison-Wesley, 2005

Horstmann, Cay S

- Big Java 4th edition, International Student Version, John Whiley & Sons, Inc. 2010

Larman, Craig

- Applying UML and Patterns 3rd edition, Pearson Education, Inc. 2004

Roskilde Business College

- 2nd semester Software Design Note Collection, Autumn 2010
- 2nd semester Software Design Relational Databases, Autumn 2010

11.2 Danish Literature

Hansen, Christian

- Advanceret Java-programmering, Teknisk Forlag, 1. udgave, 1999

Molich, Rolf

- Bruger-venlige edb-systemer, Teknisk Forlag, 2. udgave, 1998

Poul Staal Vinje

- Software Test 2nd edition, 2005
- Struktureret test, Ingeniøren bøger, 1. udgave, 2002

Frederiksen, Kim

- SQL og Database design, Kim Frederiksen – www.dataguru.dk , 2010

11.3 Webography

The holy Bible

- http://UPEDU.org - *École Polytechnique de Montréal, 2010*

SwingX

- https://swingx.dev.java.net/, SwingLabs, 2009

Nimbus

- http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/nimbus.html, Oracle, 2010

JavaHelpTDK

- https://javahelp.dev.java.net/, Oracle, 2007

Java video tutorials

- http://thenewboston.com, Bucky, 2009

Java tutorials

- http://www.roseindia.net/java/, RoseIndia, 2010
- http://java2s.com/, Java2, 2009
- http://download.oracle.com/javase/tutorial/uiswing/components/table.html, Oracle, 2010

Database guides

- http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db29.doc.apsg/db2z_identitycols.htm, IBM DB2, 2010
- http://www.w3schools.com/sql, MySQL, 2010 ☺

11.4 Java Library references

As the system is using several different libraries, we have made a list of these. They aren't made by us, but are under the public license (GNU) and are therefore free for non commercial use.

The libraries are:

iText (iText-5.0.5.jar) – Used for generating PDF files

JavaHelp (jh.jar) – The JavaHelp system

JavaMail (mail.jar) – The JavaMail system

SwingX (swingx-0.9.2.jar) – Used for autocomplete and datepicker

MySQL JDBC driver (mysql-connector-java-5. 1. 12-bin.jar) – Used for connecting to the MySQL database

DerbyClient (derbyclient.jar) – Used for connecting to the Derby database