P3 - PROJECT

Ruben Mensink
Palle Thillemann
Jonathan Karlsson
Elyas Bassam Bahodi
Frederik Østerby Hansen

An Inventory Management System for a Danmission Charity Shop

Supervisor:
Mikkel Hansen

Department of Computer Science
Aalborg University
Denmark

12/21-2016

This page was intentionally left blank

AALBORG UNIVERSITET
STUDENTERRAPPORT

Department of Computer Science
Aalborg University
Selma Lagerlöfs Vej 300
9220 Aalborg East, DK
Telephone: +45 9940 9940
Telefax: +45 9940 9798
http://www.cs.aau.dk

Title:

Inventory Management System for a Danmission Charity Shop

Theme:

Developing Applications - from users to Data, Algorithms and Tests - and back again

Project Period:

9/5-2016 - 12/21-2016

Project Group:

ds306e16

Authors:

Elyas Bassam Bahodi
Frederik Østerby Hansen
Jonathan Karlsson
Palle Thillemann
Ruben Mensink

Supervisor:

Mikkel Hansen

Total Pages: 117 + frontpage
Appendix: 4
Completion Date: 12/21-2016

Synopsis:

This project aims to develop a software solution for managing inventory in a thrift store. Specifically a local Danmission store. This store has issues with incorporating IT into their environment, and therefore does not reap the benefits an IT-system can provide. The specific store in question asks for statistical capabilities, concerning their sales, so they can monitor and possibly optimize their store.
We set out to solve this issue, by developing a decentralized software solution, where clients can access data. The system is simplistic, and yet highly customized to the stores specific needs. The system allows for registration, monitoring and removal of products. This forms the foundation of the system, upon where it can generate these statistics.

Preface

This report concerns the development of a solution to problems faced by a Danmission charity shop. In this report we will document our work and thought process throughout the analysis and modeling of the problem, as well as during the design, implementation and test of the solution.

We explored issues in thrift stores concerning their use of IT, as we anticipated that many of these types of stores/organizations were not reaping the benefits that IT-systems can provide. We wanted to see how such stores work; which issues they face, and help them realize IT systems to improve their work.

We worked with a Danmission thrift store, conducted interviews with the store manager, and analysed the store environment and the workflow. Through this we created our problem domain analysis, which later formed the foundation for our application domain models. Afterwards, we set out to design a system, which would solve the issues. We developed an inventory management system, which contains functionality to specially cater to the store's needs. Specifically, the target audience expressed a need to access statistics concerning which types of items are sold, in order to improve the store and its revenue. By tracking inventory through this system, the system can generate statistics. Additionally, the tried to design a comfortable and simplistic user interface to fit the target audience. We achieved a modern-looking, fairly user friendly experience, which Lone Alstrup (the store's manager), deemed to be usable in the store.

We would like to thank Lone Alstrup, the manager of the Danmission store, Tom Sand Jensen, development consultant of Danmission, and the employees of the Danmission store for participating in this project. We would also like to express our appreciation to our supervisor Mikkel Hansen.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Buying and selling old clothing or other second hand items is certainly not a new trend, but lately, technology has made it easier for every smartphone user to put items for sale on various buying/selling platforms, such as DBA (Den Blå Avis) and Trendsales [1]. However, these platforms are not the only ways of participating. Other options include flea markets, auction houses, garage sales, antique stores [2] and traditional charity organizations, that organize thrift stores, based on donations from private citizens [3].

There are various important aspects of recycling and donating old clothes, such as the fact that it has a significant impact on the environment. For example it takes between 1,400 and 2,700 liters of clean water to produce a single t-shirt [4]. It is also estimated that between 60% and 80% of the clothing, that a person in Denmark discards, could have used by someone else, and could have therefore been resold through a second hand store. In total, 90,000 metric tons of clothing are discarded each year in Denmark [5]. Furthermore, even if the recycled clothes are too worn to be used as clothing again, there are still many different purposes, that these clothes could be used for. An example would be, that old t-shirts could be made into washcloths. As such, the former t-shirt serves a new purpose instead of being burnt.

It has also been stated that further use of recycling could create more jobs. According to the EU Commission, it is estimated, that there are about half a million potential jobs within the EU, that have to do with recycling [4].

1.2 Initiating Problem

We have chosen to focus this project on the problems and challenges within thrift stores and related organizations, so that they can facilitate recycling more efficiently.

Below we formulate the questions intended as a starting point for the project. We will attempt to examine the problem by exploring a number of questions derived from it. Based on our initial understanding of thrift stores, we expect that they do not make use of many IT-systems. For this reason, we expect that there is a lack of optimal management and oversight of some of the different elements involved in the daily work.

How do thrift stores function, and are there any issues, that could be remedied by IT-systems?

The problem will be examined using the questions below.

- *Which different thrift stores exist, and how do they function?*
- *How does the law affect the sale of second-hand goods?*
- *Which problems do thrift stores face, that may be solved using IT-systems?*

To answer these questions, we will start by choosing one or more thrift stores to contact.

1.3 Differences Between Stores and Organizations

In this section we will go through some thrift stores, that focus on charity, and the organizations that these belong to. From the perspective of a private citizen who is simply interested in acquiring second hand goods, there are plenty of places to choose from. We will solely focus on Danish stores and organizations.

1.3.1 Typical Thrift Stores

The major charitable organizations in Denmark are the following [6]:

- Blå Kors
- Danmission
- Dansk Folkehjælp
- Frelsens Hær
- Kirkens Korshær
- Kræftens Bekæmpelse
- Mission Afrika

- Røde Kors

To differentiate between thrift stores, we looked at some basic attributes of thrift stores belonging to major charitable organizations. This was intended to provide a broad overview of these thrift stores, and can function as a point of reference for further research.

This overview consists of the thrift stores' workforce, the services they provide, the technology they use, as well as some statistical information about how many thrift stores the organizations support. The overview can be seen in table 1.1, where the major charitable organizations are compared to each other.

Table 1.1: Thrift Stores in Denmark

Category	Volunteers	Receives Used Products	Pick-Up Service	Has physical stores	Number of physical stores	Has a webshop
Blå Kors	✓	✓	✓	✓	*58*	✓*(only clothing)*
Danmission	✓	✓	✓	✓	*86*	✓*(but extremely limited selection)*
Dansk Folkehjælp	✓	✗	✗	✗	*None*	✗
Frelsens Hær	✗	✓*(only clothing)*	✗	✓	*25*	✓*(only clothing)*
Kirkens Korshær	✓	✓	✓	✓	*240*	✓*(but only for stores in Copenhagen)*
Mission Afrika	✓	✓	✓	✓	*74*	✗
Røde Kors	✓	✓	✗	✓	*200*	✗

1.3.2 Municipality Driven Stores

Danish municipalities will often have a few places where citizens can deliver their old goods, instead of throwing these out, alleviating the load on recycling sites and landfills. Through this initiative, the costs associated with waste management are reduced, and environmental friendliness is promoted.

Aalborg municipality runs a recycling center in Nørresundby (Sundsholmen), which also acts as a thrift store. Here citizens can hand in their old stuff such as: bikes, lamps, clothes etc. By reselling this, the store can then generate profit, which will go towards various local charities. Between year 2012 and 2015 Sundsholmen generated 250,000 DKK for charities. The municipality-driven stores do not generate a profit for the store, but rather for charities [7]. It is also possible for people who receive social assistance, or no assistance at all, to get furniture and other items for free, if their social case worker permits it. [8]

1.4 Choice of Organization

We are choosing to focus on thrift stores owned by Danmission. In this section we will explain our reasoning for this choice, as well as present general information about the organization.

1.4.1 Why Danmission

Of the organizations mentioned in the previous section, only a few make use of a webshop or something similar, and those that do, only do it in a limited capacity. This may indicate that none of the organizations make use of many IT systems, which would mean that none of the organizations are bad choices for exploring the initiating problem.

The next step was to contact specific stores to figure out if we could visit these for an interview. Additionally, it would be best if we could contact many of their stores as well, as it would allow us the possibility of examining the problem through a quantitative study, instead of just individual interviews. The e-mail addresses of Danmission's individual stores are readily available on Danmission's main website, so we started by contacting these.

As our questions and requests of collaboration were answered favourably by Danmission, there was no need to continue looking for another organization.

1.4.2 About Danmission

Danmission is a charitable organization that was established on January 1^{st} in the year 2000. This organisation is a merge between *Det Danske Missionselskab (The Danish Missionary Society, DMS)* and *Dansk Santalmission (Danish Santalmission)* which both originate from the nineteenth century. Over 8,000 people are supporters of the Danmission organization in Denmark, and approximately 2,500 of these people are voluntary workers in the Danmission charity shops. [9]

The daily work that Danmission does is supervised by its main office in Hellerup, where about 30 people are employed. These people are responsible for the four different departments within Danmission: *Church & Dialogue*, *Poverty Reduction*, *Communication* and *Administration*. Administration will be the focus of this project, as it is where the charity shops belong within the structure. [9]

Danmission currently owns and operates through more than 80 charity shops across Denmark, and the profit of all these shops goes directly to Danmission's work. This consist of managing and supporting programs in 12 different countries located in Asia, Africa and the Middle East, as well as in Denmark. Danmission's focus in the countries in Asia, Africa and the Middle East is on poverty reduction, religious dialogue and church development. This is achieved with the help of about 20 people employed abroad, whose jobs consist of managing and supporting the previously mentioned programs. In Denmark, Danmission's primary goal is to promote dialogue between individuals and groups from different faiths and cultures, and thereby trying to ensure

peaceful coexistence. [9]

To expand upon our understanding of the work of the charity shops, we have examined how the danish Sales Law works in relation to the sale of second hand goods. As this is not a necessary step to continue the project, but may still become important later, a chapter describing the Sale's Law can be found in appendix A. To directly continue the project and to pin down an exact problem plaguing the Danmission thrift stores, we created and sent out a questionnaire.

1.5 Questionnaire Data Analyzed

To start this project we need to figure out whether charity shops were even in the need of IT-systems or not. Additionally, we were interested in figuring out whether our pre-conceptions of charity shops, IT-systems and websites actually hold true. To do this, we had scheduled an interview with a specific charity shop. However, we figured that there was little harm in reaching out to a broader target audience, than just a single charity shop, so we also created a questionnaire to send out to multiple charity shops.

The purpose of this quantitative examination of charity shops was mainly to sense whether issues concerning the use of IT-systems were localized to the individual charity shops, or not. In the questionnaire we asked questions like whether the store utilizes any IT-system to keep track of their inventory and to show products on their website, why or why not, if they are in the need of one, and whether they would want to collaborate with us to make such a system[10].

We sent the questionnaires to a number of charity shops owned by the danish organization Danmission, and received thirteen answers. Although this is a fairly small sample size, the answers to one question in particular are so extremely one-sided, that they may well be indicating a trend. Additionally, a few of the questions we asked were not able to be answered through multiple-choice. These answers to these are worth a look regardless of the sample size.

In this section we will go through the interesting answers to our questionnaire. The full questionnaire and its results are linked at appendix B. It is an online *Google Forms* questionnaire, that was designed in a way that not all answers lead to the same next question. For this reason, not all questions were answered by every informant.

1.5.1 Use of IT-Systems for Inventory Management

Of the thirteen informants, twelve have answered that they do not currently use any IT-system to manage inventory. All of these twelve have answered that they do not keep written track of their products either. All have been asked why they do not do this and here we will examine their written responses. The responses can be sorted into three main concerns, which are mentioned below in no particular order.

- One sentiment repeated by many of the answers is that the store is too small and has too many unique products for an IT-system to be applicable.
- Another response is the concern that it would be too time consuming for very little, if any, gain.
- The last concern is that it would be very difficult to implement, as the competencies of the employees in regards to IT-systems varies a lot.

Additionally, the following answer (translated) mentions a concern, that cannot be sorted with the three main ones: "The board of directors of the store would want to use the cash register for documenting which products are the most profitable, but as our store personnel consists solely of retirees without knowledge of IT, we must acknowledge that further use of IT-Systems that the personnel has to utilize is a plan for the future."

1.5.2 Summary of Analysis

Because of the way that the questionnaire did not put everybody through the same questions, there were too few answers to the other questions to be able to conclude anything of use. The questions mentioned above, as well as their respective answers, may be very interesting for the development of a product.

The answers alone may not be very conclusive, but if the Danmission interviewees, that we have planned on speaking with, agree, then we should consider basing our product requirements upon those answers. The following concerns may be necessary to solve for a software system to be useful to a Danmission charity shops:

- The system should be unobtrusive to use.
- The system should be simple to use and should not require any learning to understand.
- The system should keep statistics for product transactions, so that it can be determined which are the most profitable.

Development consultant and regional manager of Danmission Tom Sand Jensen answered the questionnaire and contacted us directly as well. He specifically advised that

many of the answers were probably very subjective. Therefore, they would not be reflective of the fact that many of the choices in regards to IT are not made by the stores themselves, but rather by Danmission's leadership.

This raises the question of how much say the stores get in regards to their use of IT-systems. At this point it is not very clear, since they seemingly do get to decide whether to display products on a website, but not other elements. This is cleared up in the section 2.1, as it begs the question if our target audience should not be the Danmission's leadership instead.

1.6 Interviewing the Day-to-Day Manager at a Danmission Charity Shop

On Friday the 30^{th} September 2016 we interviewed Lone Alstrup, who is the store manager at a Danmission charity shop. We decided it was best to do a semi-structured interview, because of our limited understanding of charity shops, and this structure allows us to ask follow-up questions in case we learned something that was unexpected. The questions were based upon our initial understanding of charity shops and organizations, as well as the system concerns that we noticed during the questionnaire analysis 1.5.2.

In this section we analyse and summarise the interview that we have conducted. The structure of the analysis will follow the same categories as the interview questions and in the same order. We have also compiled notes, which were taken during the interview. These can be found in appendix E. The interview, and questionnaire, have been used to gain an understanding of the situation of how a charity shop runs, and therefore acts as a basis for further modeling of the problem domain as described in [11].

1.6.1 Legal and Businesslike Circumstances

Warranty and Returning Products

With regards to warranty and taking products back, the charity shops at Danmission have a few basic rules. When a product is purchased, and is an electronic device of some kind, or has a mechanical function, then the state of the product is approved with the buyer of the product present. An example would be a lamp, where the buyer is shown that the product works by turning the lamp on and off. The same is done with all electronic equipment. In case the customer returns with a defect product, they are certain that when the customer left the shop, the product was in working condition. The thrift store will therefore not provide a refund or similar. In case a customer wishes to

return an item, that is still in the same condition as when it was sold, then the customer may receive store credit.

Price Estimation of Items

The store has a dedicated contact, whom they can ask for an estimation of the worth of items. This person is a former auctioneer, and the employees usually call her to come by when they reach a certain amount of products that need to be estimated. They also have another employee, who has worked in the store for a long time, who has specific people, that he can call to also get items estimated.

Donations

Donations come from different places, but mostly from private citizens. They have had contact with the municipality, who had a lot of office equipment, that they did not need any longer. They have also gotten items from a flea market, which was held at a shopping centre, called *Friis*, in Aalborg.

Storage Space and Inventory Management

Danmission has multiple stores in the Aalborg area, and all of these, except the one that we visited, sell furniture. The one that we visited had ceased trading furniture, as it was taking up a lot of space compared to the profit made from each individual piece. Another thing about storage space, is that the store can be in a situation, where they have to refuse a donation, because they lack the space to store the items. In these cases, they refer to other Danmission charity shops, or must ultimately refuse goods.

Another concern about storage was how long the individual items were kept in the store while waiting to be sold. We expected that it did not matter how long an item would sit on the shelf, but it is actually a problem according to the staff and the manager Lone. They have not done any formal research, but they estimate that they have around 100-200 regular customers, who come by the store at least once a week. Out of all possible customers, they say that most are regulars and therefore are only interested in the newly added products. This leads to the problem that if an item is not sold within a certain time frame, then it is not going to be sold at all. The time frame according to staff and manager is around 1-2 weeks.

This leads to an inventory problem, because they have to keep track of items that have been there for too long. Currently they simply write a date or the number of the week, on the price tag of the item. At one point they tried color coding items, and through those keep track of items' storage time, but this did not work out. If the product has been sitting in the store for about 3 months, without much attention from customers, the

staff will make a decision on whether or not they think the product can be sold. Likely, the store will get rid of the product. Lone mentions that this is often done by donating the products to charitable causes in Africa.

Rent

Every Danmission charity shop needs to sustain itself economically and pay its own bills. The specific store that we visited had to pay 30,000 DKK a month for rent through their own sales. The organization will not help individual stores with their rent, nor will they pay for the store to have Internet access or utilities of any kind. The organization does, however, provide one laptop computer for each store.

Advertising

Advertising is something that the store is interested in, but is a struggle to achieve financially, since a traditional advert in a newspaper can cost the store up to 10,000 DKK. The manager is therefore interested in, and has taken some steps towards, getting inventory posted online as to reach a greater amount of people with fewer resources invested.

Customer Services

The shop, together with the four other Danmission charity shops in Aalborg, share a transportation vehicle, which is used for delivering furniture and other products, as well as being used for picking up donations from private citizens and organizations. It is mainly used for picking up furniture and other items, but is also used for transporting items that could not be sold to the waste disposal site.

1.6.2 Workday and Staff

The stores' staff is solely made up of volunteers, and in total, Danmission has 8,000 volunteers distributed throughout the organization. Due to differing ethnicity of the volunteers, some of them do not speak or read danish well, and therefore communication may have to be carried out in English. The volunteers do not have issues with carrying out different work tasks, though most do have a preference. For example the people who work at the cash register are often the same. The volunteers are usually between 50 and 80 years of age.

Store Manager

The manager's responsibilities include the following:

- Work plans

- Hiring
- Accounting
- Communicating with the development consultant and the rest of the organization.
- Buying coffee, snacks and other maintenance products
- Creating and updating lists of phone numbers
- Printing various signs for use in and around the store.

The store has a hierarchy, similar to what you would find in a traditional corporate environment. Lone Alstrup describes this environment as a "mini workplace". The store is visited by a labor inspectorate, just like any other workplace, as the store is classified as commercial activity. The stores' manager is responsible for accounting and reporting to the organization each month. This can be done more often, but is not a requirement. This information is sent via an Excel spreadsheet to the central accounting department of the organization.
The other employees have the following responsibilities:

- Sorting inventory and marking products with a price tag
- Attending the cash register and helping customers.

These are the core tasks that take up the most time, but there more things done on an *ad hoc* basis, for instance:

- Cleaning
- Receiving donations and checking these items
- Stacking the shelves with sorted items
- Removing items that have been in the store for too long
- Seasonal projects, such as Christmas decorating the store
- Delivering products

The staff works in teams consisting of 2-3 people, where one team always works a whole day at the store. The teams usually consist of the same people, because they enjoy working with each other, and are therefore not likely to change.

1.6.3 IT-systems and Administration

Cash Register System

The store uses a cash register system and a payment terminal as the only two payment options. The two systems are not integrated, which is why they have to manually combine the transaction information, when the monthly sales report is to be sent to the organization. This is done in an Excel spreadsheet. It is of the stores' interest to have these two systems integrated, as it would save them time.

Statistics of Sales

There is a limiting factor in using the standard cash register system, which is that products are not categorized in any way. This means that the store does not have statistics with regards to what kinds of products are sold. The interviewee mentioned that the store was interested in a system that can keep track of what kinds of products are popular. The way the store handles this currently is through their own perception of what kinds of products they sell the most of.

1.6.4 Technical Experience

The technical capabilities of the staff are very limited. Few staff members are confidently able to work with IT. This is a major issue in the store, as each attempt to incorporate anything IT-related in the store, leaves members of the staff in severe discontent, which usually results in multiple staff members wanting to quit. This was the case two years ago when the store switched to offering the use of payment terminals, and is the case once more, as the store is currently in the progress of integrating their payment options into the cash register system.

1.7 Limiting the Scope

In this section we will limit the scope of our project and thereby cut away the excess complications and issues, that we do not wish to involve in further development of the system.

In sections 1.5 and 1.6 we found out that there is a wide range of problems, that could define the scope of the project. These problems can be categorized as follows:

- Problems concerning the organization, represented by informant Tom Sand Jensen
- Localized problems, concerning individual stores and their manager
- Individual problems, concerning only individual staff members

An overview of these problems can be seen on figure 1.1. The the overlapping boxes signify who the problems affect. For example, the box containing the problem *no statistics about sales*. This problem is a concern of the store manager *Lone Alstrup* and the regional manager *Tom Jensen*.

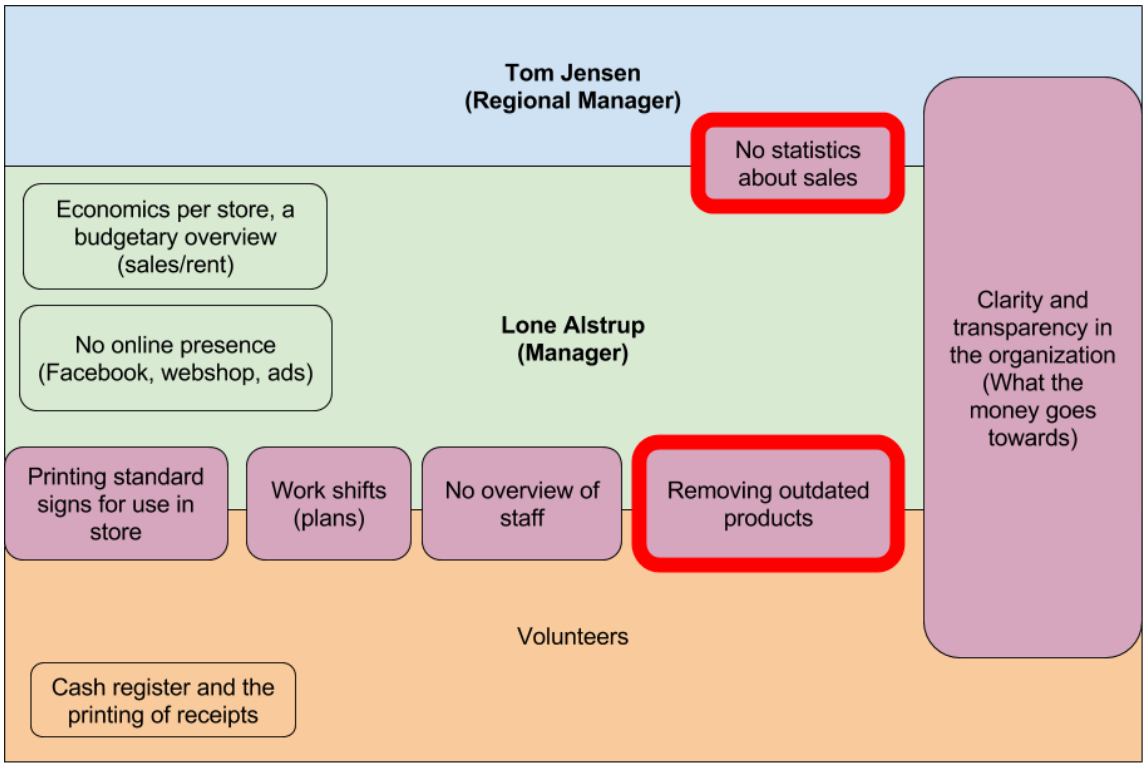


Figure 1.1: Overview of the different problems at a Danmission charity shop

1.7.1 Project Focus

We have chosen to focus on issues concerning statistics, as well as outdated inventory, which can be seen on figure 1.1 marked with a red outline, and according to the figure, the first problem relates to the regional manager and the store manager, while the latter problem is a concern to the volunteers and the store manager.
Our primary reason for choosing the problems *no statistics about sales* and *removing outdated products* is, that matters of statistics are handled very loosely, as described in section 1.6.3, which we consider a non-solution. The secondary reason for these choices is, that the solution to the problem of lacking sales statistics can act as the foundation for the solution of other problems, such as the low online presence, and the removal of old products from the shelves.
The remaining identified problems in figure 1.1 already have some kind of existing implementation in the store, although these are often simplistic non-IT solutions they are still solutions and therefore not as important as problems where there is no solution at all, which is a reason against focusing on these problems.

1.8 Problem Statement

After analyzing the problem and limiting its scope, we can now define the problem statement for this project. Our main focus in this project will now be on examining and

solving the problem statement. The problem statement is the following:

The employees in the Danmission shop do not currently have any way to see which items that sell well and which ones do not. Furthermore, there is currently no dedicated system for keeping track of the amount of time an item has been part of the shop's selection. How can an IT-system help the employees keep statistics of what they sell the most and least of, and determine for how long they have had the specific items?

To solve the problem statement, we will be defining the target audience for the end product. Afterwards we will create a system definition of the solution, in order to alleviate the problems of the target audience. Lastly, we will be defining the concrete requirements that any created system will have to abide by. First, however, we will specify which development method we will be using throughout the project to develop the solution.

1.9 Development Method

In this section we will describe how we are planning to structure our development process throughout this project, and how we expect this method to aid in our work flow. We will describe and choose between two different development methods: Waterfall and Iterative.

1.9.1 Waterfall method

The Waterfall method is also referred to as a linear-sequential life cycle model, and is in its most simple form, a very straightforward way of organizing a project. Different phases of project development are conducted and completed in a linear manner, as can be seen on figure 1.2.

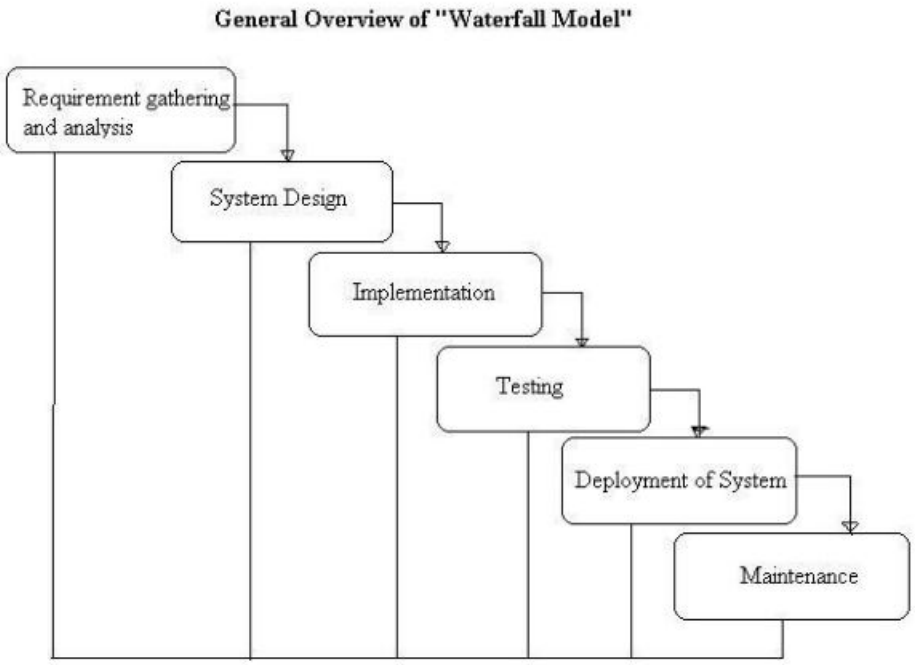


Figure 1.2: An overview over the sequence of the phases.

The early phases are, as such, completed early, and the late phases late. Only once a phase has been completed and reviewed does the next phase begin. In its most fundamental form, the Waterfall method does not allow for returning to previous phases in order to revise these. Therefore the Waterfall method is not very flexible, though it is very effective if the project it is applied to has a very well defined problem and if there exist no ambiguous requirements. The Waterfall method is good for complex problems with little to no uncertainty concerning the requirements. [12]

1.9.2 Iterative method

The Iterative method, on the other hand, works by refining iterations of the project until the requirements are met. Each iteration is a mini-project itself, where the phases are completed and feedback is collected, see figure 1.3.

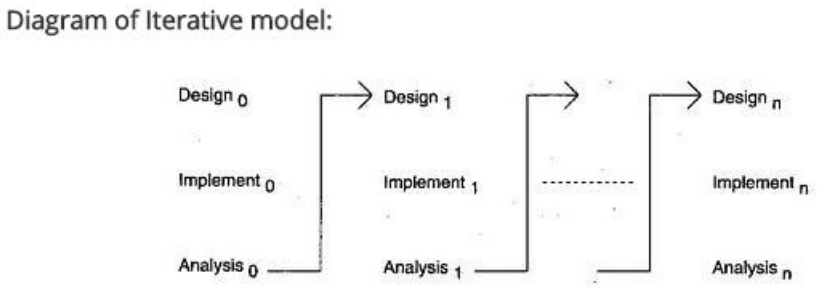


Figure 1.3: An overview over Iterations

Most projects using the Iterative method use iterations that last from one to six weeks each. This method works especially well on large projects, as it starts to become impossible to cover all the the problem's intricacies in only one, large iteration.

The major benefit of this method is that it forces late stages of the program to be drafted early, which allows issues concerning these to be caught early as well. The method does, as such, handle uncertainty concerning the requirements well, though it does also encourage it. [13]

1.9.3 Structuring project development

For the project as a whole, we are planning to use the Waterfall method with few Iterative qualities added. The main benefit that we plan to gain using the Waterfall method is a a clearer and more rigorous work flow, which allows a better measure of the projects progression and thereby better time scheduling.

As we will be writing this project for a target audience with requirements that are not set in stone, we do not wish to lose all project flexibility. To achieve this, we will be drafting latter phases of the project prior to reaching them in the Waterfall sequence. Through this we aim to gain feedback concerning the direction of the project early.

Specifically, we will be drafting a prototype of the program early, and then interview the Danmission store to figure out if the project is heading in the correct direction. We will also be testing the usability of the program early, to allow time for design and functionality changes. In short, we are aiming to use the Waterfall method as our main structure of development, but will be communicating with Danmission early the whole way through to avoid pitfalls caused by requirement uncertainty.

Chapter 2

Problem Analysis

2.1 Danmission and IT-systems

First of all, it is necessary to examine who is authorized to make any sort of changes to IT-systems in Danmission charity shops, otherwise any system might be created using the wrong preconceptions.

The information that we have gathered about this topic through the questionnaire, interview as well as our own research has not been sufficient to reach a conclusion. For this reason we have contacted Tom Sand Jensen, the development consultant and regional manager of Danmission, and asked him about their IT-systems and how much the individual stores may decide in regards to these. We hope to gain insight into the situation concerning IT-systems in Danmission charity shops.

2.1.1 Implementation of IT-systems

Danmission currently encourages the use of IT-systems in their individual stores, so the organisation may take advantage of the tools that these offer for the management of large scale purchases and training of employees. For this purpose they have already planned to implement a specific electronic system for accounting in all of their stores during year 2017. Outside of electronic accounting, this system will allow for the data to be automatically imported into the organization's central accounting, and allow the individual shops to get subsequent reports and financial statements. This tool will not only ease the shop's accounting, but also allow the management fast and continuous insight into the economy of the individual stores.

Additionally, the central management is also currently exploring a unification of cash register solutions. It is however a very long process to implement these kinds of systems, as there are so many volunteers involved, many of which are not very familiar with IT in general. This is also reflected in the information that we received during our

interview with Lone Alstrup, the store manager of the Danmission charity store. It was mentioned that the change to using payment terminal was problematic, as employees were threatening to quit.

2.1.2 Use of Websites

We also specifically asked Tom about who is allowed control in regards to the individual stores' websites, and how much say the stores themselves have in this regard. This question originates from our initial research of the different charity shop organizations, where we found that some of their websites were very different from the rest; see for instance [14] compared to the standard appearance of [15].

The reason for these disparities is that, some years ago, several of the individual stores created websites themselves. Currently the stores with unusual websites are starting to migrate to the organization's website domain. After some instruction, they are allowed to customize their own websites, though it obviously has to be in accordance with the available template.

Although Danmission is simply using *WordPress* to create and modify their websites, instructing volunteers in its use has proven to be difficult. The volunteers are generally not very interested in working with the websites, and much prefer the use of Facebook.

2.1.3 IT-system Authority

To summarize, the stores themselves have a high amount of control over how they utilize IT-systems and their own website. The management of Danmission is currently working on getting their stores to use IT-systems at all, and is therefore currently not exercising much control over their use.

Additionally, the management of Danmission currently has plans to increase the use of IT-systems on a large scale within the year 2017, which means that the idea of creating IT-systems for the charity shops is not impossible. Furthermore, the management is clearly able to introduce the use of IT-systems into the charity shops, as they did with payment terminals in 2014.

We now know that the main target audience for our product can be the volunteers working in the charity shops, without it causing issues for the product's viability in real world usage. As such we also choose these as our main target audience. We choose to focus on the management of Danmission only as a secondary target audience. This means that we will tweak the program and its design towards their wishes only if these tweaks

will not cause issues for the main target audience. The obvious exception is when we would risk the product's viability by not changing the program in some way.

2.2 PACT Analysis

In this section we will be creating a PACT analysis on charity shops to further our understanding of the problem as a cohesive whole. Here we will be reviewing the information from the previous chapter 1. PACT is an abbreviation of "**P**eople, **A**ctivities, **C**ontext and **T**echnologies". Each of these topics will be described in the following subsections.

2.2.1 People

In this part of the PACT analysis we will be focusing on the people who work in charity shops. People have unique personalities and as a result they tend to react differently under certain conditions. Therefore, it is important that the system created during this project is user friendly to as much of the target audience as possible. People will have varying degrees of technical expertise, and this is something that should be considered during the development of a solution, so that both inexperienced users and experts will find the system easy to use.

Users of the system will mostly consist of voluntary workers. This will likely entail that employee working hours vary significantly. For this reason we expect that the ease of use of the system will be more important to its quality rather than the time-efficiency.

It is also important that the system is usable by users who are physically impaired. Therefore, things such as color blindness, poor vision and other visual deficiencies need to be considered when designing the user interface. As a result we might need to reduce usage of small and difficult to see objects in the final product. There may even be the need of an option that allows the user to enlarge buttons and text displayed on the screen. Furthermore, if pictures are used there should also be a written description available, that explains what the picture is depicting. Additionally, there should be a heavy use of easy-to-understand icons, instead of only text. This would also make it easier for users who suffer from dyslexia or similar conditions.

2.2.2 Activities

Products/wares are frequently added to or removed from the store. Therefore, it is essential that these operations in the final system are easy to find and use. This will also be of help to users, who are not experienced with IT-systems.

The system should be easy and fast to operate in day-to-day use. Data that needs to be added to the system would probably consist of a product name, price and some sort of unique identifier (ID, barcode etc). On top of that, to improve the interaction, a picture of the product would also need to be added. This will make the products easier to browse and find, thus making the system easier to use and understand.

2.2.3 Context

The system in a physical context will be accessed in the charity shop, by the employees. Since the system's data primarily will be stored on a server and accessed through the Internet, the environment will have to be centralized. Meaning the content will be stored on a remote server, within a database structure. Furthermore, different charity shops are most likely using different computers which means that the system should be able to operate in different environments and operating systems.

2.2.4 Technologies

The technological platforms available will vary greatly, but as a minimum the system should be able to run on old laptops, as our target audience has a purpose other than using money on equipment. As second hand stores are usually run by organizations that mainly focus on charity, spending money and time on new hardware is not part of their usual method of operation. For this reason, the focus should be on supporting older versions of technical platforms (older browsers or operating systems). As a result of this, the system should function comfortably on a wide variety of screen sizes and resolutions. Buttons, text and arrangement/layout should fit as many different systems as possible, while still living up to the other design requirements previously mentioned.

Additionally, we cannot expect that the store's connection to the Internet to be completely stable, neither can we expect the connection to be fast. Therefore, it would make sense to make the system operate asynchronously, where a copy of the database is stored locally, and then synchronized with the remote database periodically.

2.3 Exploring the Target Audience

In section 2.1 we decided that our main target audience would be the volunteers working at Danmission's charity shop. In this section we attempt to gain a greater understanding of the target audience, through the use of personas and scenarios.

2.3.1 Personas

The personas that we are creating will be using the understanding that we have gained through the PACT-analysis, questionnaire as well as the the interview at the Danmission charity shop. One of the personas will represent the staff, and the other will represent a store manager.

The two personas are shown on figures 2.1 and 2.2. We wanted to explore how the typical older volunteer navigates current tasks, and why that person is motivated to work in the charity shop. Therefore most of the persona, as seen on figure 2.1, is based on observed data, although our persona has some visual impairments, which are typical for older people [16]. Visual impairments is something that we think we should explore early on so that we can account for this in the design process. Implementing a new system can have an effect on how current tasks are done, possibly changing those. This is something we also want to be aware of, in that if a system changes the way current tasks are done, then the new way might not be something that the employees like, which will affect their motivation and willingness to volunteer.

Michael Hansen, Volunteer at Danmission — Age: 71

I like working in the store. It gives me something to do, while I also get to spend time with a good friend.

General Profile

- Born 1945, Michael Hansen in Hirtshals, Nordjylland
- Married in his early 30's.
- Retired at the age of 63.
- Has worked as a volunteer for 8 years.
- Acts as the primary babysitter for his four grandchildren.
- Has degraded vision due to age.

Likes and Dislikes

- Dislikes change and prefers to do things the same way that he has always done them.
- Enjoys talking with the other employees.
- Likes the thought of making a positive difference for someone else.
- Prefers to start work at noon.
- Prefers working with inventory, sorting and labeling items.

Environment

- Usually drives to the charity shop himself, which takes about 10-15 minutes.
- Dedicates 5-10 hours a week, working as a volunteer.

Attitude to Computers and Technology

- Generally avoids using anything that has to do with technology and computers.
- Does not own a mobile phone and does not use email, but owns and uses a landline.

Purpose of using the system

- To register and deregister products, as well as getting information about outdated products.

Figure 2.1: Persona 1 - The Employee Persona

Linda Frederiksen, Manager at a Danmission Shop — Age: 42

Doing all this extra work is tough, but really worth it, since it is for a good cause.

General Profile

- Born 1974, Linda Frederiksen in Nørresundby, Nordjylland.
- Married in her late 20's
- Has a Bachelor's degree in history.
- Is a part time teacher, while also works as a manager at Danmission two days a week.
- Has worked as a volunteer for 4 years.
- Has three children, aged 14, 12, and 9.

Likes

- Likes organizing and implementing new ideas.
- Enjoys pursuing her vision for a more uniform Danmission charity shop chain, together with the regional manager Mark Nygaard.

Environment

- Is able to drive or walk to work, since she lives close by.
- Dedicates 8-16 hours a week, working as a volunteer.

Attitude to Computers and Technology

- Generally tries to include computers and technology in overcoming daily problems and challenges.
- Owns and uses a smartphone, as well as using social media and email, on a daily basis.

Purpose of using the system

- To get an overview of the shop's inventory and statistics of sales.

Figure 2.2: Persona 2 - The Manager persona

We will henceforth use these personas when considering what the target audience might prefer while designing the system. These simplified and less complex representations of the target audience will allow us to disregard needless nuances during the design of the broad strokes of the system. We will hereafter focus on the process of formulating the requirements of the end product.

2.4 Scenarios

Based on information gathered in the interview, seen in section 1.6, we will be creating a number of scenarios that are intended to illuminate how the target audience may benefit from an IT system.

Story 1

Scenario type: User story.
PACT-Overview:

- People = Michael persona
- Activity = registering products
- Context = Danmission store
- Technology = no technology.

Rationale
This scenario is about portraying how the employees handle the procedure of receiving donations, to putting the products up for sale.

Michael has just arrived for his afternoon shift. He has just had a cup of coffee with his co-workers, where they also exchanged some information about current tasks that need to be done. The early shift was supposed to have gone through a new donation and placed the items in the inventory (figure 2.3), but they had taken a break instead. Michael spends some time going through the items, checking for condition and if it is something they think they can sell. He places the items in the inventory; where-ever it makes sense. Then the manager comes to ask, if he can find and add some more footwear products to the store. He finds a pair of boots in the inventory and examines them carefully looking for a certain brand, which could have an effect on the products value. He reads a name, slightly faded, inside of the shoe, enough to make out what it says, although it is not a brand he recognizes. To his right, there is a list of brand names (2.4), which he uses to figure out, if the price of the pair of boots should be more than usual. He finds that the brand is indeed on the list and therefore proceeds to add another 100 kroner to the standard price.

Figure 2.3: A picture of Danmissions unsorted inventory

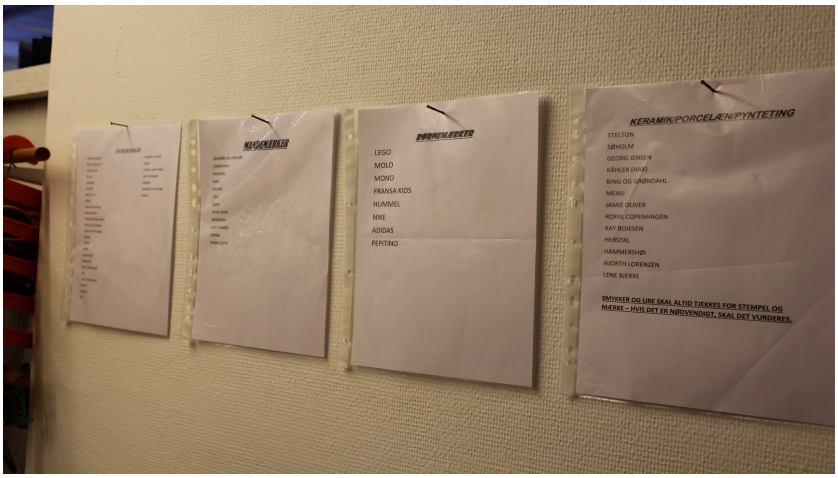


Figure 2.4: A picture of the lists of brands in the charity shop.

Story 2

Scenario type: User story.
PACT-Overview:

- People = Michael persona
- Activity = finding outdated products
- Context = Danmission store
- Technology = no technology.

Rationale
In this scenario we describe another core activity of the employees, which occurs frequently.

Later in the day, Michael and the manager decide to find the products that have been in the store for too long. It is not easy, because a lot of labels have to be looked at manually, but Michael does not mind. His first approach is usually to go off of intuition, and tries to remember which products have been there for a long time. After having found a number of items, mostly clothing, the labels are removed, then the items are put into a plastic bag and are now ready to be sent.

Story 3

Scenario type: User story.
PACT-Overview:

- People = Linda persona
- Activity = Making online advertisements for products in the store
- Context = Danmission store
- Technology = A laptop with Internet access, and a smartphone with the ability to transfer files to the laptop.

Rationale
To highlight how the current situation is limited by the fact, that the manager has no clear and readily available overview, of their current products, which also limits their ability to post online advertisements

Linda, the manager, wants to put out a new advertisement, which she likes to do through the store's Facebook group. Firstly she needs to figure out what products would be the best to put online. She decides to consult with the rest of the staff to get an overview.

They pick out some items, after which Linda takes some pictures, with her phone, for the online advertisement. When posting the advertisement to the Facebook group, she realizes that the price is not clearly visible on the labels. She then writes the price into the item description, for the online advertisement.

Story 4

Scenario type: User story.
PACT-Overview:

- People = Linda persona
- Activity = Printing another set of lists of brands
- Context = Danmission store
- Technology = Laptop and a printer.

Rationale
Describes the updating process of the brands that the employees have to account for, but also some of the other aspects of the managers work.

One of the employees encountered a new brand that they have not had in the store before. The employee asks Linda if the brand is of higher value, after which Linda decides to search on the Internet for some answers. After having spent some time searching, she decides to call on a contact, an auctioneer. After the phone conversation, Linda has been informed about this type of brand and wants to make sure the employees are reminded of the new brand. She opens the document containing all the brand names on the shops' computer, and adds the name to the list, after which she prints the updated document, and replaces the existing paper document, with the newly printed one.

2.4.1 Conceptional Scenario

Scenario type: Conceptual scenario.
PACT-Overview:

- People = Manager/employee
- Activity = Searching for information
- Context = Danmission store
- Technology = Laptop or other device, such as a tablet, with Internet access.

Rationale
This scenario is about portraying how the employees handle the procedure of receiving

donations, to putting the products up for sale.

-

The employees that can use/have access to the laptop, distributed by Danmission, will be able to register products. When a product is sold the same device can unregister the product from the shop's system. If a product is outdated the device will receive a notification and then the employee will know exactly what product to unregister. If the manager also has a basic set of computer skills and has access to the Internet, through a laptop or other device, she will be able to have an overview of their current stock, time of day when they sell the most and what they sell the most of. The laptop will also serve as an aide in setting up online advertisements.

Keeping all of these lessons in mind, we can now create a system definition that might satisfy the target audience.

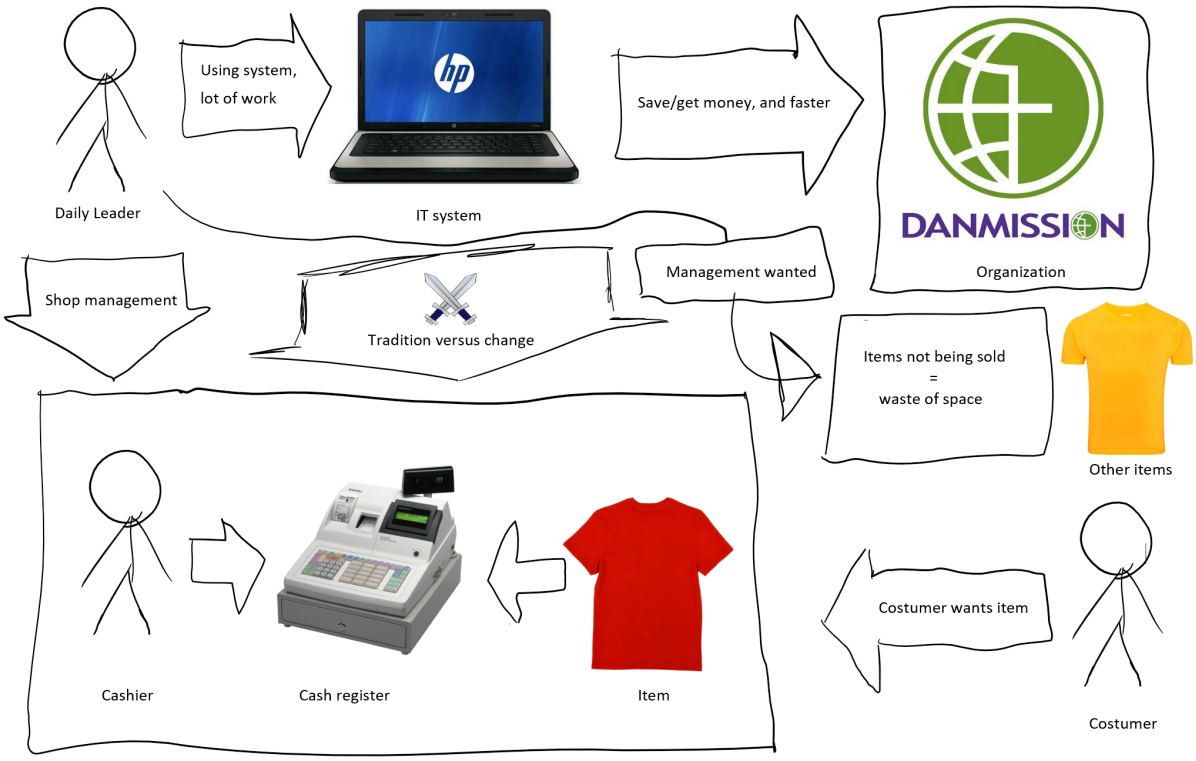
2.5 Choice of System

In this section we will be creating a system definition to explain our current understanding of how we may start improving charity shop management through an IT-system. As soon as we have created a system definition that the target audience agrees with, then we can start finding specific ways of solving their problems and designing the system.

2.5.1 Situation

To create a system definition we first need to understand and get an overview of the current situation in the charity store that we plan to create a system for. To better understand it and get an overview, we have here created a rich picture [17] of the situation as we understand it. The rich picture is made from the interview that we conducted with *Lone Alstrup* from one of Danmission's charity shops where she described how products are handled in the store during the interview. The rich picture can be seen in figure 2.5.

Figure 2.5: A rich picture of the Danmission charity shop where we conducted an interview



We use the rich picture to obtain a certain degree of understanding of the charity shop's current management, and their wishes for the future, as we found out in the section 1.6 through the interview. While we have two groups with differing goals, the store manager and employees, we found out that, as illustrated in figure 2.5, that Lone is prone to using and is experienced with IT-systems, while on the other hand, we have the employees, that are not experienced with IT-systems and wish to keep their current workflow. This desire is shown in the box at the bottom of the rich picture in figure 2.5.

This leads us to the current understanding that there is a lot of varying experience with IT-systems in the shop, and as well a wish to change the current management to a management, that uses IT. But we also found that there might be problems with the varying experiences with IT-systems in the shop, since some employees are prone to quit when something complex is presented. We will therefore take these issues into consideration while formulating the FACTOR-criteria of our system definition.

2.5.2 FACTOR

In this subsection we will look at the FACTOR-criteria, which will be the used as a basis for the system definition. The FACTOR-criteria will be based on the current understanding of the charity shop, gained through the interview in section 1.6, the PACT-analysis in section 2.2 and the rich picture in section 2.5.1.

The criteria of a FACTOR-analysis are the following: **F**unctionality, **A**pplication domain, **C**onditions, **T**echnology, **O**bjects and **R**esponsibility [18]. We are, as such, trying to figure out how each of the FACTOR-criteria will be fulfilled, prior to making the actual system definition. The FACTOR-analysis can be seen in table 2.1 below.

Table 2.1: FACTOR-analysis

F	Help with the registration and bookkeeping of second-hand goods. Notify when a product/item is outdated.
A	The IT-system is to be used primarily by the manager, but also the employees in the charity shop.
C	Users will have varying experience with IT-systems because of the voluntary workforce.
T	A cheap laptop owned by Danmission, Internet access, no touchscreen.
O	Transactions, inventory and statistics.
R	The IT-system should help determine which products are being sold/not sold and increase revenue.

2.5.3 System Definition

Through the FACTOR-analysis in table 2.1 we can now create the system definition. The system definition is as follows:

A tool in the form of an IT-system, that is able to help with the registration and bookkeeping of second-hand goods, and notify employees/managers of outdated products. Registration of products and the general management of the system is handled through a Windows-based application running on a cheap laptop. The goal of the IT-system is to increase the revenue of the charity shop by generating statistics of sales and expenditures, through which the store will know which products to keep on display, and which products to remove.

Any system, that we might create, will have to abide by the system definition. Additionally, throughout our collaboration with Danmission and research on the topic, we found a number of specific wishes, requirements and problems, that our system could be focused on. We will in the following section be presenting all of these, so that we might choose which ones to focus on and how, specifically, to design our system.

2.6 System Requirements

In this section the system requirements will be stated. These requirements are based on the knowledge gained from creating persona and scenarios in section 2.3 as well as the interview conducted with Lone, seen in section 1.6. They will be divided into requirements that are *soft* and requirements that are *strict*. The soft requirements are options that the target audience would like to have, but are not mandatory for the system's viability. For each requirement we will give a short summary of why the requirement exists and what it means.

2.6.1 Strict requirements

The strict requirements for the system are the following:

1 Gather statistics

This requirement arises from our interview, where the store manager heavily emphasized the wish that the system should keep track of which type of items in the charity shop sell well. This would allow the managers to gauge which products are popular, and thereby focus on the sale of these.

2 Notify user about outdated products

This requirement arises from the store manager's mentions the issue that is outdated products taking up space in the store and having little chance to be sold. This requirement ensures that the store can keep track of the time that items have been for sale, and make changes or discard the product, if necessary.

3 Bookkeeping of transactions

Since the sales law provides a two years warranty, described in appendix A, it is important for the system to be able to keep track of any transaction for up to two years, in case a customer returns a product. Furthermore, the bookkeeping of transactions will also be useful for gathering and generating statistics, as described in the first strict requirement.

2.6.2 Soft requirements

The soft requirements for the system are the following:

1 Allow showing items on a website

Through the interview and DBA's report [19], we found that a webshop or a similar system might be very beneficial to charity shops. Through this requirement it is meant that the system should be programmed in a way, that allows for easy coupling to a webshop or something similar.

2 Printable identification of products

In any system that helps manage the inventory of the store it will be necessary to identify the physical products in the store, which correspond to certain products in the system. It would therefore be advantageous if the physical products were distinguishable from each other through some label or specific identification. This soft requirement arises, as it should be easy and fast to create unique labels.

3 Allow the switching of languages in the program

During the interview it was mentioned that they do have trouble with communication at times, because the store's volunteers may not be able to speak danish. This soft requirement ensures that language barriers do not cause issues while utilizing the program.

With the system requirements now set, we can begin to design and later implement the system for the target audience. We will later use these requirements as a gauge for how well our system ideas fix the issues of the target audience.

Chapter 3

Prototyping

In this chapter we will be presenting our target audience with a horizontal prototype, to test design ideas. Through this we wish to gain a better understanding of their perspective and specific wishes for the system. Afterwards, we will be using this information to approach the final design of the system.

3.1 Description of Functionality

In this section we will conceptualize and describe a solution to the problem statement, and decide upon its functionality. This solution should conform to both the system definition and system requirements, and is intended to be far less ambiguous than any previous mention of a solution.

As such, the point of this chapter is to get an overview of the different systems that we could choose to create, and an understanding of why we are choosing the functionality that we end up creating our prototype for. After this we are able to create a prototype of the solution, and ask our target audience which one they would prefer, and then iterate on that basis. We will start by listing the requirements and envision solutions to these individually. Afterwards, we will visualize a combined solution to then develop further and create a prototype of.

3.1.1 Partial solutions

The problem statement focuses heavily on the lack of statistics over which products are more desired than others, and the system definition widens this focus to also include the registration and bookkeeping of the products in question. Here we will create partial solutions to both the strict and soft system requirements.

Strict: Gather statistics

To solve this requirement, there obviously needs to be some functionality in the system that uses the data in it to create statistics. Less obvious, however, for the statistics generated by the system to be meaningful in any way, the sample size needs to be large enough. This necessitates that there are a lot of products and transaction in the system. To then adhere to this requirement in a proper way, it is therefore important that adding, removing and editing products in the system is as easy and time efficient as possible. Inefficiency in this department may heavily damage the product's viability.

Strict: Notify user about outdated products

To conform to this requirement, the system has to keep of the date at which the products were added to the store. This means either that the users manually have to add this information to the products, or that the system simply uses the date at which the individual products have been created in the system.

Strict: Bookkeeping of transactions

To meet this requirement, the system will simply need to keep track of the transactions and removed products for the required time, before allowing deletion of these.

Soft: Allow showing items on a website

For this requirement to be solved, the design architecture of the system should be created in a way that allows for easy coupling with external programs or services. This is not a consideration, which will be important during the design of this prototype, however.

Soft: Printable identification of products

This requirement can be solved in a number of ways, though all require the store to own printer that works with this purpose. The system should then be able to interact with the printer, for the requirement to be considered properly solved. The product identification to print could be numbers, barcodes, QR-codes or other similarly working, and readable/scan-able identification. The alternative to solving this requirement is leaving identification of the products to the users. They can then chose for instance to write numbers on a piece of paper, or just try to recognize the products once a customer wishes to buy one.

Soft: Allow the switching of languages in the program

Similar to the soft requirement about showing items on a website, solving this requirement necessitates that the program architecture is designed in a way that allows for this

functionality to be added. It will, however, not be important for the development of the prototype.

3.1.2 Joint solutions

Herein we will describe the joint solution, that we are envisioning, and explain how the target audience is expected to use the system. Additionally, if pertinent, we will explain how the solution will solve specific requirements.

Solution One

See figure 3.1 for the solution, and how adding new products to the system would work.

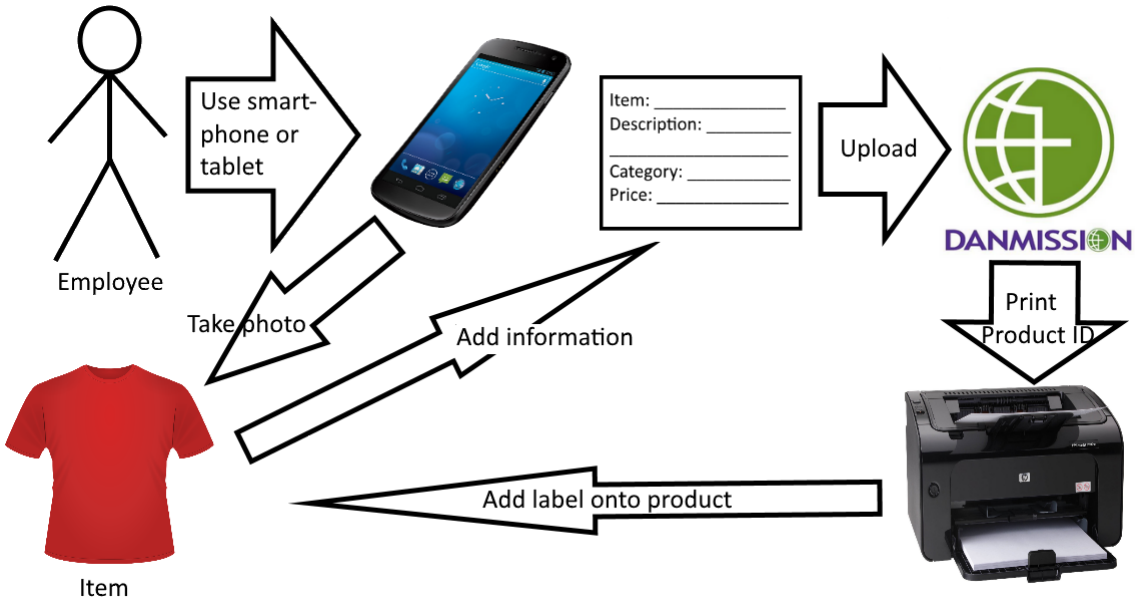


Figure 3.1: Illustration of the solution

The idea behind this solution is that, anytime an employee receives an item donation, the item is added to the system by first taking a picture of it. Important information about the item is added, and it is uploaded to the system's main hub. The system then automatically prints a label of identification, which the employee marks the product with. When a customer then buys the item, the cashier scans the label and the item is removed from the inventory. Taking a picture of the product makes the system very well suited to be extended to show the inventory on a web front or online store.

The store manager has the additional option of viewing statistics about the items that are, and have been, on inventory. If the system has detected any item, that has been on inventory for longer than intended, then it will prompt the manager to remove said item from the store and the system, as the product will likely not be sold, and is just taking

up space in the store.

The solution was initially envisioned to be used only on a laptop, but as taking a picture of the product is a major part of the solution, we will expand the solution to work with smartphones or tablets. As such, the management-part of the system and its statistics will still be exclusively done on a laptop. Taking a picture of the product and adding these to the system, is, however, a task that is better suited for hand held devices with cameras built in. The option of adding products to the system through the computer, should, however, remain, as it cannot be expected that all employees own a smartphone or tablet.

We will now create a prototype of the solution and ask the target audience for feedback on it. This prototype will be of the part of the system, which is planned to run on a tablet. This part of the application is expected to be the main interface that the employees will be interacting with on a daily basis, and is therefore the most relevant for feedback.

3.2 Development of the prototype

In this chapter we will be creating a prototype to the solution described in the previous chapter, so that we may ask the target audience which elements they like or do not like. We wish, for instance, to figure out whether the employees can use the systems in addition to their routine assignments.

3.2.1 Planning

Through the creation and testing of these prototypes, we intend to answer the following questions:

- How will the users react to the prototype and will they be able to imagine how the final system might look?
- Do the users have any further requirements or wishes for the user interface or functionality?
- Is the system practical to implement in the store? Is it practical to incorporate it into the daily operations?
- Is the GUI intuitive and effective enough to be used in daily work?
- Do the users prefer the use of QR codes for inventory management over other solutions (like writing labels by hand)?

The prototype will be created through Google Slides, as such that we are able to mirror the user interaction with the GUI as closely as possible with the final product. It works in the way that clicking any of the different buttons illustrated on the slides, will redirect

the user to a different slide, to emulate a working user interface. The prototype will illustrate the GUI for the part of the solution that is an application for a tablets, as it is this part of the system that the employees are intended to be interacting the most with.

3.2.2 Interface design

We will delve further into how we fit the user interface of the product to the users in a later section. For prototyping, we found it sufficient to keep to consistent core ideas and values about how prototype user interface should look. These values are:

Minimalism

We intend for there to only be very few, if any, aesthetic additions to the final products, which do not directly correspond with the system's functionality. As such, we wish for GUI elements, like buttons to press, to be very simple to understand. Additionally, we will attempt to make these as big and easy to see as reasonably possible.

Low choice complexity

We intend for there to be as few choices as possible for the user to consider on each sheet of the prototype and product. This way we are ensuring that, at any time when a user is confronted with a question or an option, that user reached the question or option deliberately.

Partitioning of functionality

In the same vein as the previous, we will attempt to keep the complexity low for the average user, by only showing and allowing functionality to the type of users, who are intended to use it. This may seem like a restrictive measure, but we find it important, as it will be difficult enough to get the average user to even use the basic functionality of the system. Disturbing the users with additional functionality would be counterproductive towards this goal. Worth noting is that this will create a need for management of the different user types (for instance manager and employee).

Based on our users (persona), we have put together a mock-up of what a tablet interface of the system might look like. With big colorful buttons and easily readable big fonts, the interface is very simplistic and minimalistic. As the users are mostly elderly volunteers, with very little experience with IT systems, the interface of the prototype is meant to run on a touch-based device such as a tablet. As a result of this, we have big, visually appealing buttons, with high contrast. We have chosen to theme the interface based on Danmission's own color scheme (green, white and purple).

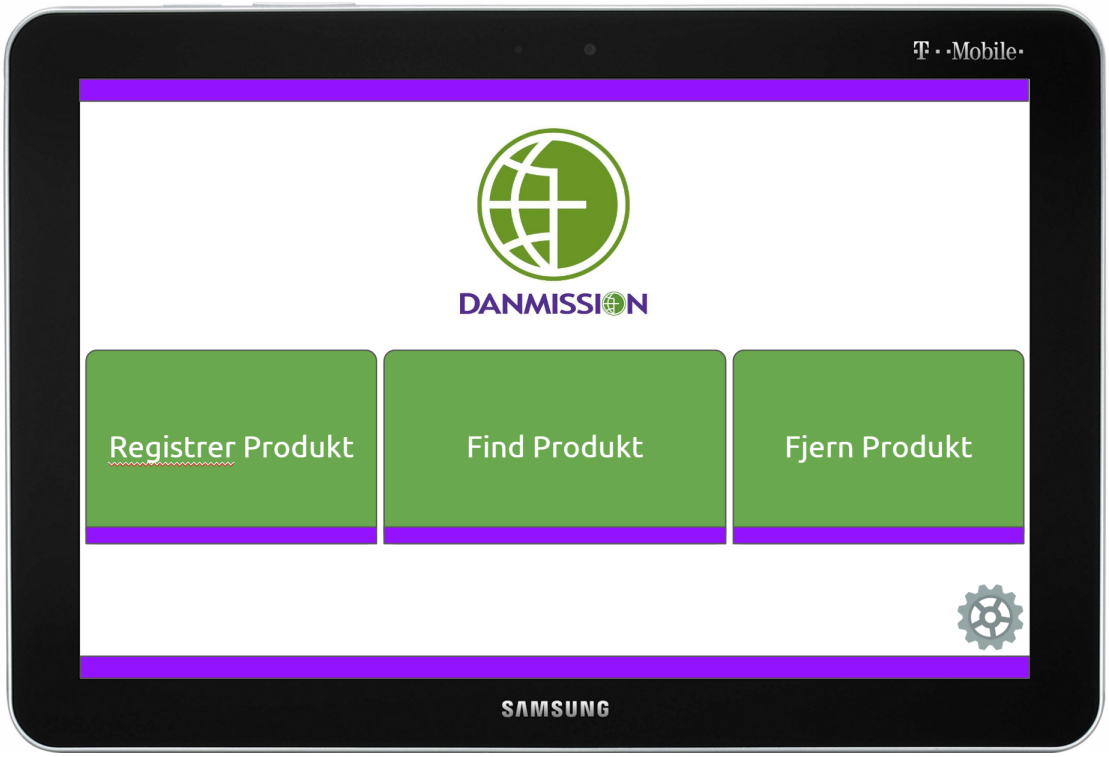


Figure 3.2: Mockup GUI prototype, made with Google Slides

Every screen in the interface has a "Back"- and a "Cancel"-button. This allows the user to either take a single step back, or cancel the current operation and return to the home screen. The interface also makes use of icons that are representative of certain actions, such as the cog-wheel on the home screen and the camera button for taking pictures. This helps to make the interface more intuitive and straightforward to use for someone with less experience with IT systems. Systems with text heavy interfaces can be difficult and time inefficient to navigate for users users, and we have therefore tried to keep to using simple wording and icons.

3.3 Evaluation of the Prototype

During Friday the 4th of November, we visited the Danmission store once again, and talked with Lone Alstrup to receive feedback about our prototype.

Overall, her feedback to the prototype was mostly positive, however with a few mentions about it might be improved. As a whole, she found that our heavy focus on solving the issue of lacking statistics about the products was good. Specifically she thinks it will be good for the management of the store, as well as for the management of the organization. It can help answer questions and give information about questions that she, at the moment, can only guess about.

One of these questions is obviously which kind of products sell well, and which ones do not. The management in particular has been very interested in this information. She also mentions that, for instance, it has been difficult to track where the store's income is coming from, ever since they stopped selling furniture. Additionally, a system like this might allow for more unification of product prices across the different Danmission stores. At the moment, the product prices seem very much dependant on the employees' tastes in products. This might cause products, that could have been sold, if they were given a chance, to just be sent to charitable purposes.

She also mentions how the layout of the prototype is well and logically constructed, and that, for instance, the names/icons of the buttons are fine for giving understanding of what they do. She reckons that as long as the user of the system knows but a little about using a computer, smartphone or tablet, then the layout, as it is at this point, should not be a problem.

She did also had some feedback to very specific parts of the system. For one, she thought it was important that the system allow the defining and redefining of product categories. More importantly, it needs to be possible to add many products into a singular product ID. This is important, as it's not feasible for the user of the system to be asked to scan 500 pairs of children's socks separately.

We could solve this by allowing the user to create products in the system, which are not exhausted if the item is bought. For instance a pair of children's socks might not be checked into the system as a product with name "Blue Children's Socks, Size S" and a unique ID, but instead simply added as additional entity under the product "5 DKK Children's Socks". When one of those entities is the bought, the employees simply scan the ID of the collection of wares. The ware then remains unchanged in the system, but it is noted in the statistics that one entity has been sold. It is of course also possibly to count the amount of entities in the collection of items, but it does not appear to be worth the effort to micromanage the wares, that the employees do not wish to check in separately.

As we presented our prototype, Lone evaluated it to be very practical to use either QR codes or barcodes on products. She suggested to only use a written number either in decimal or hexadecimal. In the same vein, Lone mentioned that the system should not necessitate taking a picture of each product while adding it to the system.

Additionally, she mentioned that the labeling of the products should only contain basic information such as price, ID and perhaps the fact that the product is not returnable. Furthermore, she emphasized that the system should allow for labelling to be done for

each category and not necessarily for all individual products. She also mentioned that they at the moment label their products, mainly clothes, simply by men, women or children.

Lone mentioned that this system would be difficult to teach to some of the elderly employees. Some of the elderly employees, however, do own smartphones or tablets themselves, and those should be able to understand how such systems work. Along with these, the younger employees should also be willing to learn the system. The difference between these, Lone reckons, is the mentality of trial and error. She reckons that they are scared to try the system mainly because they are scared of doing something wrong. The way that we are combating this, and will be, during further development, is to limit unnecessary choices and make the functionality that needs to be accessed as obvious and easy to access as possible.

Lone also mentioned that the system, that we are developing, should be able to be used in multiple Danmission stores, as most other stores probably have employees that have knowledge about IT too.

Although this prototype was made with a tablet/smartphone device in mind, we decided to narrow our focus onto the statistics and inventory management capabilities of the program, as these are the most essential requirements. For this reason we will solely be developing the system as a Windows application. We will use feedback from this chapter, and apply it to our Windows application, as the same design decisions will apply. A mobile client may still be a future extension of the system.

We will now be working towards the design of the actual system through an analysis of the problem domain and the application domain, spanned by our prototype.

Chapter 4

Modeling the Store

4.1 Problem-Domain analysis

In this section we will model, analyze and thereby describe the problem-domain. We model the classes and events, that exist in the problem-domain. This gives us an overview of what it contains, and gives us the ability to structure the domain. We also analyze the behavior of classes, such that we can model what happens within each class, and more importantly, model their interaction and relations. Within the problem-domain, we are especially keen on elements we wish to administer or observe. All this achieves an overview, through modelling, of the problem domain. These three activities, *classes, structure and behaviour*, will all come together and form a cohesive model of the problem domain. The model is then going to form the foundation for the Application Domain Analysis in chapter 4.2, as it's models will be based upon those in the problem-domain. Through these activities we wish to further our understanding of the problem and all its intricacies, to avoid making costly mistakes later on.

4.1.1 Classes and Events

In this subsection we will cover the description of the various classes, events and objects we have derived from the user stories 2.4 and the system definition 2.5.3. This section will also include an event table where the events and objects will be characterized as well as the relation between the various objects and events will be shown. Classes will have a description of why it has been chosen and the operations and attributes will be defined too. Although brands are a part of the problem domain we have chosen not to model them in this project due to them not being an active part of the interaction between events and objects.

The following list is an evaluated list of events:

- Product registered

- Product unregistered
- Product returned
- Product sold
- Product outdated
- Daily sales accounted
- Monthly sales reported
- Store section created
- Store section closed
- Standard prices revised

The following is an evaluated list of classes:

Product

The system needs to keep track of which products are in stock. Because of this we have chosen to include the class *Product* which contains information about the product's ID, type, price and the date it was added to the shop's inventory.

Transaction

The system needs to keep track of the transactions that occur when products are bought or returned and determine which types of products are selling well. Because of this we have chosen to include the class *Transaction* which contains information about the transaction's product, type, the day the transaction occurred, and how many vouchers were included in the transaction.

Standard Product Prices

The system needs to keep track of the standard prices for the different types of products that the shop sells. Because of this we have chosen to include the class *Standard Product Prices* that contains information about the standard prices of different types of products that will be helpful when adding new products to the shops inventory.

Voucher

The system needs to keep track of the value of a voucher that is given when a product is returned. Because of this we have chosen to include the class *Voucher* which contains information about the voucher's value and date it was given.

Store Section(Categories)

The system needs to keep track of which categories the different types of products can belong to. Because of this we have chosen to include the class *Store Section* which contains information about how many products fall under the different categories such as male/female clothing, jewelry or porcelain.

The evaluated list of events and classes are based on candidate lists, which were generated through a group brainstorming session. These un-evaluated lists can be found in Appendix F. The specific evaluation criteria can be found in [20].

To get an overview of how the events are related to the classes, we have made an event table 4.1. For each event, if there is a + this means that the event occurs 0 or 1 times per object of the class and * means that the event can occur 0 or multiple times for each object of the class.

	Classes				
Events	Product	Transaction	Standard Product Pricing	Voucher	Store Section (Categories)
Product Registered	+		*		*
Product Unregistered	+				*
Product Returned	+	+		+	
Product Sold	+	+			
Product Outdated	+				
Daily Sales Accounted		*		*	
Monthly Sales Reported		*		*	
Store Section Created					+
Store Section Closed					+
Standard Prices Revised			*		

Table 4.1: Event table for the evaluated classes and events

4.1.2 Structure

This subsection will cover the structure of the classes from the class list we evaluated in section 4.1.1 and after structuring the classes we look for the relationships and multiplicity between the classes. This activity will result in a class diagram overview over the classes, their relation and multiplicity. This activity is done to achieve an overview over the classes.

Structuring Classes

The class diagram shown in figure 4.1 shows the relation between the classes we have. The relations we make use of are aggregation, association and the multiplicity between the classes.

Transaction

To modulate Danmission's method, we look at the process from when a transaction happens and its relation to the voucher Danmission issues when a customer returns a product. While the voucher is, in this case, a part of the transaction, we model it by having "Transaction" aggregate "Voucher" as shown in figure 4.1. The multiplicity for the classes is 1 for "Transaction" and 0 to many(*) for "Voucher", because there is the possibility of no vouchers issued or several ones issued when a transaction occurs.

The class "Transaction" aggregate "Product". The reason for that is that we want "Product" to be a part of "Transaction", because it is important to have information about the product when conducting a transaction. For each transaction there can be many products involved and therefore, as shown in figure 4.1, the multiplicity is 1 for transaction and 1 to many(*) for products. This also means that a transaction has to contain at least 1 product.

Store Section

The "Store Section" class is a collection of categories for a product and therefore an aggregation is established since "Product" is a part of "Store Section". Therefore for each Store Section(Category) we can have several products, therefore the multiplicity in the class diagram is 1 for "Store Section" and 0 to many(*) for "Product". The reason we place a zero multiplicity for "Product" is because a category can either have products or be out of "Products", but still exists.

Product

Danmission places a standard price for each product and in order to modulate the

relationship between the two, "Product" aggregates "Standard Price" as shown in figure 4.1 and because there is only one standard price for each product type, e.g. shoes, we set the multiplicity to 1 for "Product" and 1 for "Standard Price". Furthermore "Product" also aggregates "Store Section" because 1 "Store Section" can contain 0 to multible products.

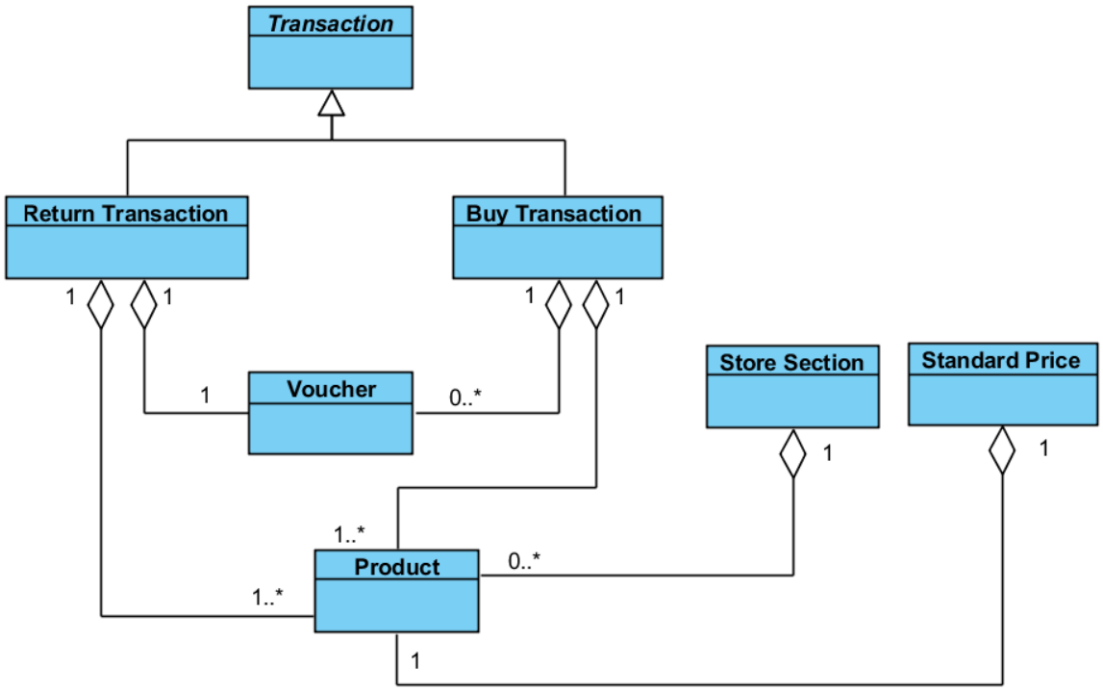


Figure 4.1: An overview over the structure of the class diagram and the multiplicity between the classes

4.1.3 Class Behaviour

In this subsection we will be describing the behaviour of the different objects in the most essential classes that we found to exist in the problem domain. The essential classes in this case will be *Product*, *Transaction*, *Standard Product Price*, *Voucher* and *Store Section* The descriptions for each class along with their class state diagram can be found in the descriptions below.

Product

The product class is used to contain data on each individual product. This information includes data such as naming, pricing, identification, category and possibly a description. This provides a framework for each product to be stored and managed in the store, using the system. This class is essential to the system, as it defines the central objects in the system; the products themselves. It's an issue that products remain in the store for extended periods of time. Therefore, after a certain amount of time, a product should be removed. The product is registered in

the system, when it's processed in the store (pricing, categorization, etc.). Then later removed, either due to a purchase or due to being outdated. In the case where a product would have to switch to another store section, the product would have to be unregistered and registered to another section.

As shown in the state chart 4.2 for the product class. The first thing to happen is that the product is registered and then reaches the active state. From this state the product can be edited and the date it was added is checked. From here the product can be sold or become outdated which both will result in the removal of the product.

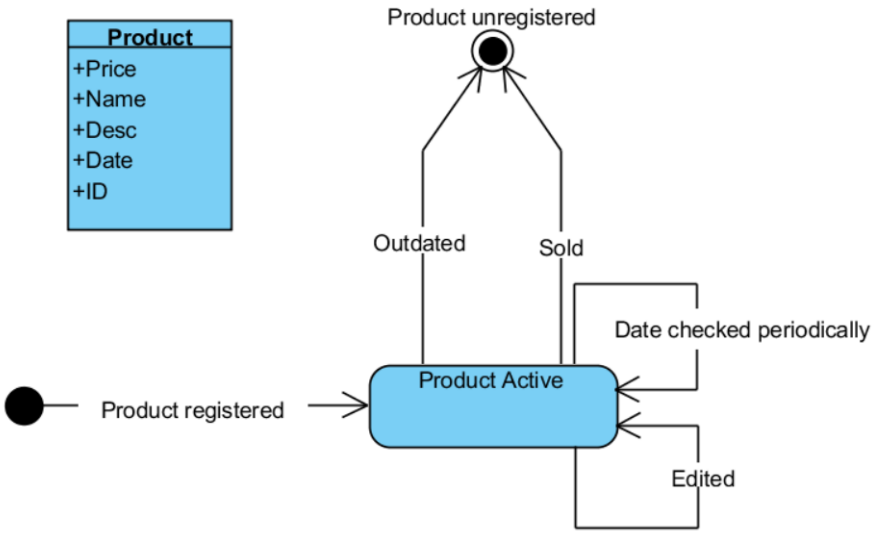


Figure 4.2: An overview of the behavior of the product class

Transaction

Originally we had a single *transaction* class, but a consequence of this would be a more vague behavior. Therefore we decided to make to specializations of this class, even though the behavior of these two specializations will be very similar in most aspects. The consequence for our model, is that in principle, a customer cannot buy something and return something at the same time.

BuyTransaction

This class' responsibility is to contain data about how many products are sold, at which time, and how many vouchers were used in the transaction. An ID is also assigned to each transaction, so that they can be uniquely identified. A state chart diagram of this class can be seen in figure 4.3.

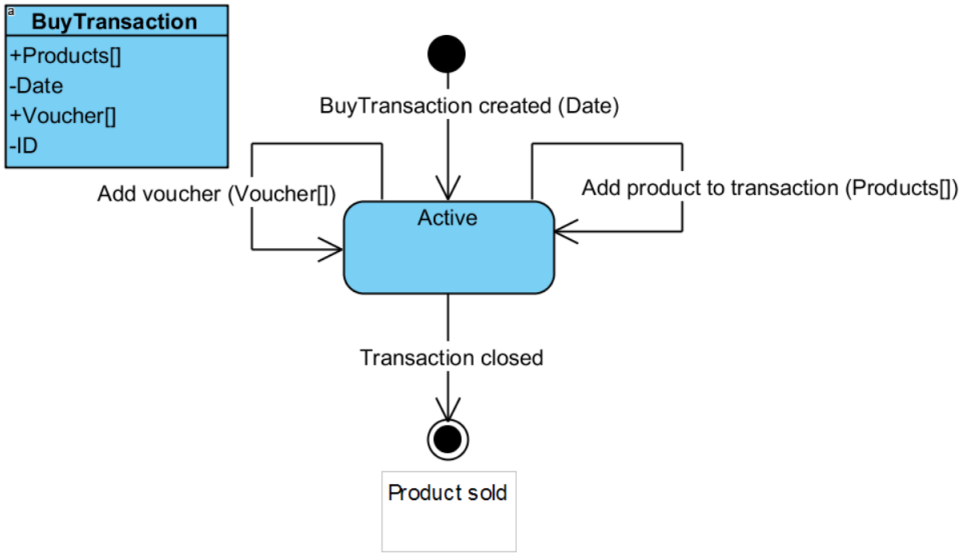


Figure 4.3: An overview of the behavior of the BuyTransaction class

The first thing that happens is that a *BuyTransaction* is created, after which products and vouchers can be added to the transaction. The transaction closes, and products are considered sold.

ReturnTransaction

The only difference between this class, as seen on figure 4.4 and the *BuyTransaction* class is that only one voucher is part of this class, since only one voucher will be generated, and the problem domain event that occurs afterwards, is that the returned products will be registered.

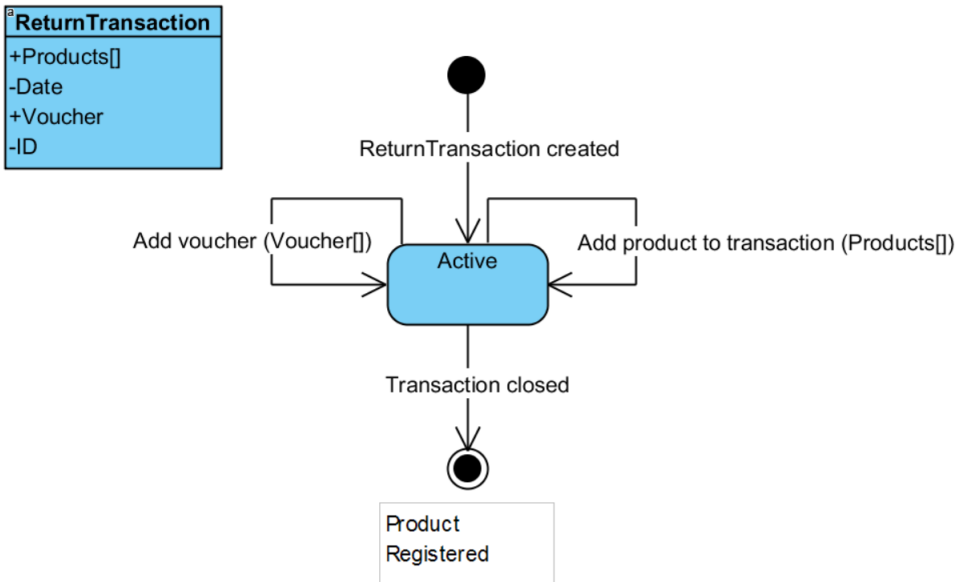


Figure 4.4: An overview of the behavior of the ReturnTransaction class

Standard Product Pricing

The class seen in 4.5 is a list of customary prices for products of the specific categories (whichever store section the products are in). The standard product prices are being used by employees as suggestions on how to price the products. This is done for all products and and can be changed if it is necessary.

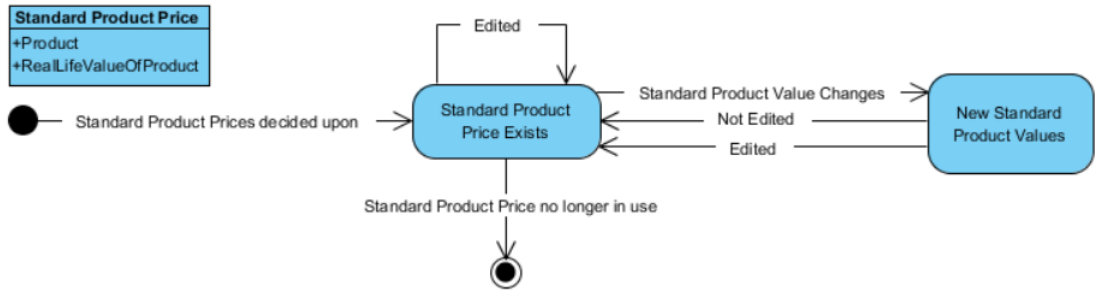


Figure 4.5: An overview of the behavior of the standardprice class

As shown in the state chart 4.5 for the standard product price class. A standard product price is decided on and the state *Standard Product Price Exists* is reached. From here the product price can be edited from the *New Standard Product Values* state, or be removed when the standard price is no longer in use because the product category is no longer used.

Voucher

The voucher is issued to the customer, when a customer returns a product. Instead of a cash refund, the customer receives a voucher for the returned product, as per store policy. This voucher contains information regarding the date of issuing and the credit amount. The voucher can only be instantiated by returning a product.

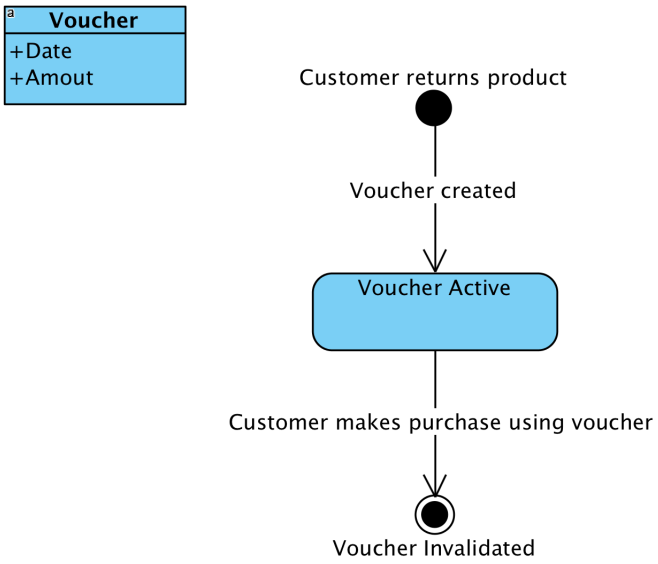


Figure 4.6: A state diagram of the voucher class

As shown in the state chart 4.6 for the voucher class. the first thing to happen is that a product is returned and a voucher is created. Then the voucher reaches the active state, and when a purchase is made with the voucher it is invalidated.

Store Section

The store section class is used to keep track of the main categories that exist within the store. It is important for the system, because it provides a separation of products by category that is equal to how the store is structured, which will make it easier searching for products manually in the system, and also with regards to adding new products to the store. An individual store section object has to keep track product objects, that belong to it. A store section object also keeps track of how much physical store space is allocated to it, and by doing it this way it will be easier to generate statistics about how efficient each section of the store is, which would lead to the possibility of optimizing the limited space that the charity shop has available. The behavior of this class is described in the following figure 4.7

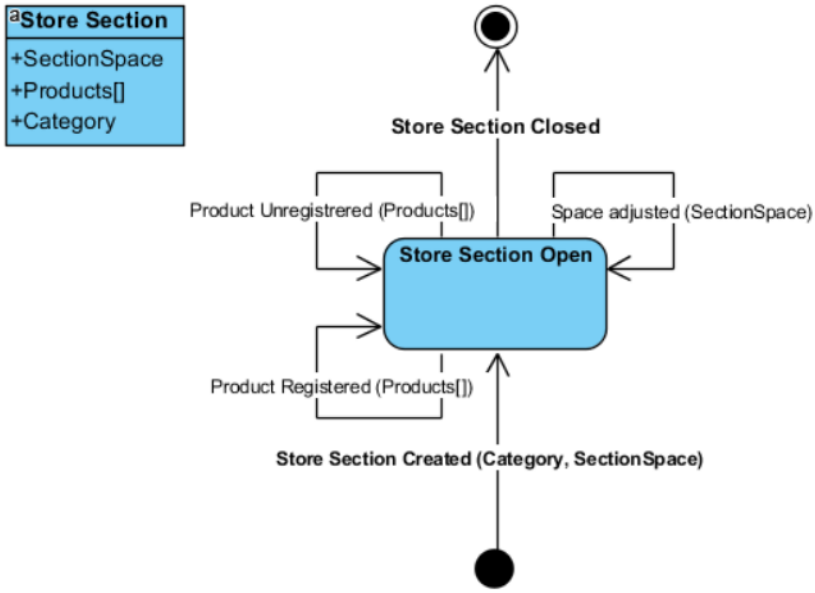


Figure 4.7: An overview of the behavior of the store section class

As shown in the state chart 4.7 for the store section class. The first thing to happen is that a store section is created and it reaches the *Store Section Open* state. From here products can be registered or unregistered from the store section and the space the store section uses can be adjusted. This state ends when the store section is closed.

Now that we've gained a better understanding of the problem-domain and created a class diagram, event table and described the behaviour patterns of our classes, we can begin analysing the application-domain to gain a better understanding on how the system should work.

4.2 Application Domain Analysis

At this point in the project we are ready to create the application domain analysis. Here we will take a look at the *Actors* who are going to use the system, *Use Cases* for the system, *Functions* that are going to be part of the system and the *Interfaces* that the actors are going to use in order to monitor and control the problem domain. The actors in the system are the employees and the manager in the charity shop, as described in section 2.4.

4.2.1 Actor Table

To be able to determine which tasks the actors in the application domain take responsibility of we will create an actor table. The actor table can be seen in figure 4.2

Table 4.2: Actor table for the charity shop

	Actors	
Use Cases	Employee	manager
Register product	✓	✓
Sell product	✓	✓
Modify product	✓	✓
Unregister product	✓	✓
View statistics		✓
Administer store section		✓

From the actor table 4.2 we can see that the manager of the shop is going to use the system in the same way that a regular employee is going to. On top of this the manager of the shop is also going to be the one who gets information about when specific products are not sold despite being available for a determined amount of time. The old products in question can be removed by the manager or an employee. Furthermore the manager can also view statistics about the different product types and administer the different store sections in the shop.

4.2.2 Use Cases

In this subsection we will be creating an actor table for the project. Here we will be taking a look at the actors in the application domain and what their use cases are. As mentioned previously the actors in the application domain are the manager and the employees that work in the charity shop.

Register product

When we conducted the interview with Lone Alstrup, as shown in section 1.6, we learned that when the shop receives new products, an employee would have to make sure that each individual product was intact, and as for electronics they would have to function as intended, and estimate the price of the product before it could be set for sale in the charity shop.

Because we wish to fulfill the system definition in section 2.5.3 where we wish to create statistics to increase the revenue of the charity shop as well as help with the registration and bookkeeping of products. We will also have to register these products in the system we are going to create. A use case specification for registering products to the shop's inventory could be the following:

Use case

Register a product to the shop's inventory. This is done by an *employee* or the *manager*. The following information is gathered so that the product can be registered and set for sale in the shop:

- Date and time
- Condition of the product
- Product type
- Product price

Objects

Product, Standard Product Prices, Store Section

Functions

Register product

Sell product

The shop also sells products to costumers in the shop. A use case specification for selling products from shop's inventory could be the following:

Use case

Sell product from the shop's inventory. This is done by an *employee* or the *manager*. To be able to sell a product the costumer who wants the product must bring it to the cashier. Then the cashier can sell the product and use the IT-system to mark the product as sold through the products unique ID.

Objects

Product, Transaction

Functions

Sell product

Unregister product

From the interview in section 1.6.1 we learned that the employees in the store would like to remove products that were not being sold despite being on the shelf for a long time. A use case specification for removing products from shop's inventory could be the following:

Use case

Remove product from the shop's inventory. This is done by an *employee* or the *manager*. To be able to remove the product an employee or the manager must be notified by the system when a product has been for sale for a long time, but has not been sold. Then the product will have to be found through its ID and then taken of the shelf to be replaced with another product.

Objects

Product, Store Section

Functions

Unregister product, remove product

Modify product

If, when you create a product in the IT-system and make a mistake, you will have to be able to modify the product in some ways within the IT-system so that it reflects the physical product. This could, for example, be adjusting the price of the product because it might be worth more, or less, than first estimated. Or change the description of the product. A use case specification for modifying products from shop's inventory could be the following:

Use case

Modify product from the shop's inventory. This is done by an *employee* or the *manager*. To be able to modify the attributes of the product an employee or the manager must use the product ID to be able to find the product within the system. Once this is done the employee or the manager are free to change the attributes of the product, such as price or item category and so on.

Objects

Product, Standard Product Prices, Store Section

Functions

Modify product

View statistics

From the interview in section 1.6, we also learned that the manager would like to be able to view statistics about which types of products are in demand, and which are not. A use case specification for viewing the previously mentioned statistics could be the following:

Use case

View statistics from the desktop application. This is done by the *manager*. To be able to view these statistics the manager must use the desktop application that will be part of the result for this project.

Objects

Product, Store Section, Transaction

Functions

Generate statistics

Administer store section

From the interview in section 1.6 we learned that the employees and the manager sometimes change what types of products the different sections in the store are used for, by removing the types of products present to exchange them with some other types of products, effectively *closing* a section of the store and *opening* a new one. A use case specification for administering the store sections could be the following:

Use case

Administer store section form the desktop application. This is done by the *manager*. To be able to change what a store section is used for the manager must use the desktop application and change, create or close a store section.

Objects

Store Section

Functions

Add store section, remove store section, modify store section

4.2.3 System Functionality

In this subsection we are going to be looking at the functions that are going to be part of the system for the Danmission charity shop. The result of the section is going to be a list of functions, their type and their complexity.

We base the functions of the system on four different types of functions: *Update*, *signal*, *read*, and *compute* functions [21].

Update

This function type is an update, caused by a change in the problem-domain, which requires a change in the system.

Register product

Whenever a product is put for sale it will have to be modeled in the system.

Sell product

The system will also have to reflect when a product is sold.

Return product

When a product is returned the system will have to reflect this.

Unregister product

A product is sometimes removed because it is not sold, the system will have to reflect this.

Edit product

The price of a product can also be changed, the system will have to reflect this as well.

Create voucher

When a product is returned a voucher is given, this will also have to be modeled in the system.

Create category

Different products are registered and these belong to different categories. For products that do not belong to a category a new category can be created, this will have to be modeled in the system.

Edit category

An existing category can also be edited to encapsulate more or less products. This will also have to be reflected in the system.

Edit standard pricing

When standard pricing is revised, the system will need to update a part of the model. This includes adding new pricing categories, as well as removing or changing older ones.

Create store section

When a store section(category) is opened, the system will need to reflect this change.

Close store section

The system will also need to reflect when a store section is closed.

Edit store section

A store section can also be edited so the system will have to reflect this as well.

Signal

This function type represents a change in the model, which should signal to the actors, that a change has occurred.

Product outdated

When a product is determined to be outdated, a notification should be displayed to the actors, so they can take action. This is signaling a change in the model to the actors.

Read

This function type happens when actors query the system for information.

Find product

When an actor is viewing the inventory of products, read-functions are taking place between the client device(smartphone, tablet or laptop) and the data server. Actors can perform search queries to navigate the data.

Compute

This function type is defined by the need for computations to be executed by the system, to help actors.

Generating statistics

When an actor wishes to view statistics from the system, the system will need to compile data in the model. This will result in a comprehensive statistical overview, but will, as a result, require some computation to take place. Statistics contain data concerning sales, displaying a detailed overview. Data about products, store sections, etc. will be included in the statistics.

Function list

In figure 4.3, we have compiled a list of functions, as well as an assessment of their respective complexity and type. Generally, changes applied to databases are considered simple in their complexity i.e. product changes. This is due to the simplicity with interacting with databases, relative to more complex data processing, like generating statistics. The subfunctions contained in *Product changes* are all assumed to have same complexity and type. Therefore, we have abstracted this to a collection.

Function	Complexity	Type
Register product	Simple	Update
Find product	Simple	Read
Sell product	Simple	Update
Return product	Simple	Update
Unregister Product	Simple	Update
Edit product	Simple	Update
Create voucher	Simple	Update
Create category	Simple	Update
Edit category	Simple	Update
Edit standard pricing	Simple	Update
Create store section	Simple	Update
Close store section	Simple	Update
Edit store section	Simple	Update
Product outdated	Medium	Signal
Generating statistics	Complex	Compute

Table 4.3: Function list

In table 4.3 the function *Generating statistics* is listed as a complex function. To make the function less complex we have chosen to split the function into a number of less complex functions that has the same functionality when combined, these functions can be seen in table 4.4.

Function	Complexity	Type
Fetch data	Simple	Read
Compute statistics	Medium	Compute
Display statistics	Simple	Read

Table 4.4: Functions that replace *Generating statistics*

The function *Fetch data* will be used to gather the data that is requested by the user, this data could be gathered from products, transactions, store sections, etc. This data will then be handed over to the function *Compute statistics* that computes the requested statistics. *Display statistics* will then display the computed statistics to the user.

4.2.4 User Interface

In this subsection we will decide upon a user interface for the desktop system. We will also look at what type of user interface we have chosen for our solution as well as the

dialog in the system. We have chosen to ignore the interface of the hypothetical Android platform, as we have determined this to be outside of this project's scope, earlier.

Dialog

For this project we have chosen the menu selection pattern [22], both for navigating and giving the system input. Put simply the menu selection pattern centers around the user navigating through different menus in a program to complete different tasks. The reason we have chosen the menu selection pattern is that it is easier to navigate through, and also significantly shortens the learning of how to use the program for novice users, i.e. our personas 2.3.1, when compared to something like the command-language pattern [22] which would require instruction and training of the users before they could use it efficiently.

The Windows application is going to have a main menu, where the user can choose from a variety of sub-menus, depending on what the user wishes to do. Each sub-menu will have its own sequence of sub-menus, and contain exclusive data. This means that data shown in one menu, is only relevant to the context of this menu. This will make the interface easier to navigate, as redundant information is not shown to the user. This application will allow the user to fully manage the system. This means registering, managing and removing products, as well as categories and system configuration. Additionally, the application can compile and show statistics, all this is handled by a system of menus.

4.2.5 Overview

To get an overview of how the system is going to be navigated we have created a navigation diagram 4.8. This describes the navigational patterns, that the user can take. Thereby plotting the navigation of buttons and menus, here the boxes in the diagram represent different menus or views and the arrows lead to different views depending on which button is clicked. This means we can create an overview of what the application should take care of, and how functionality should be structured in the user interface.

An example to clarify the navigation diagram could be that from the main menu we click the options button which takes us to the options menu. From here we can change the options in the system and/or go back to the main menu.

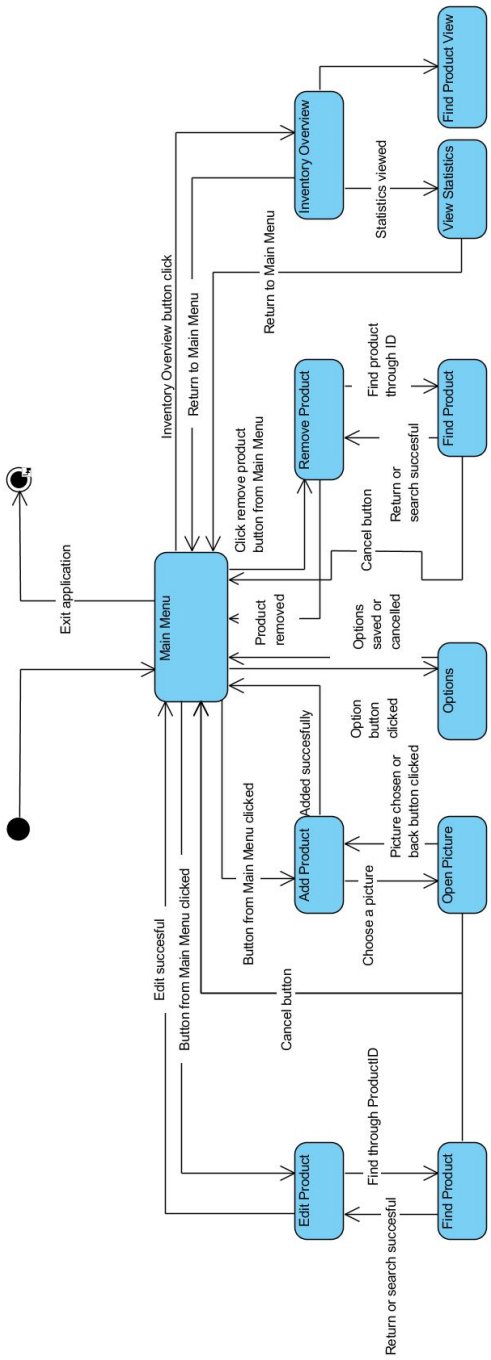


Figure 4.8: Navigation diagram for the desktop application

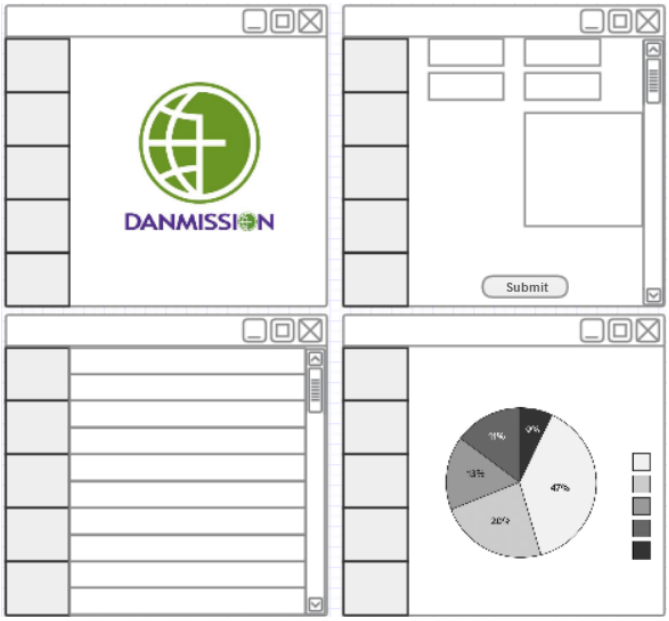
The blue boxes are states, which the system navigates through. While we start at the Main Menu the arrows describe how many patterns the system can take from thereon. Since the system has to navigate to other states and still be able to return to the initial Main Menu, the arrows describe this action through the comments written above them. While navigating through the states there will be some states that have single or more states that can be navigated to and accessed. Some states in the system has the functionality to return to the Main Menu without going through every single state that has

been navigated through.

Examples

We are going to look at some of the essential screens in the user interface, on the Windows platform. The following screens are designed and inspired from similar programs, and lightly based on our Android prototype from chapter 3. During implementation some changes will likely occur because of *Windows* design standards/conventions. The most essential screens can be seen in figure 4.9. These UI examples are black and white, except for the logo, and in our final desktop application we will use colors similar to that of the logo to have it resemble the color that Danmission uses.

Figure 4.9: Examples of the different views in the User Interface for the desktop application



Now that the application-domain has been concluded and we have defined the functions that our system has to use, we can begin designing the system, and decide how its architecture should be.

Chapter 5

System Design

In this chapter we will cover the aspects of designing a system that is appropriate for solving the problem statement, and thereby the issues mentioned by the target audience. This system should abide by the requirements as well as the store leader's feedback to the prototype in chapter 3. We will be looking at the technologies and platforms we will be using, as well as system architecture. This will form the basis for the implementation in the following chapter, where we document the actual system implementation in detail.

5.1 Technologies

The prior descriptions of our system, as well as the functions described in our application domain analysis heavily suggest the use of certain technologies. In this section we will describe which advantages they offer, and how these may be used in the system.

5.1.1 Database Configuration

In this subsection we will cover the database configuration. Which platforms and technologies are we using, and why we are using them.

Choice of Database

For our project, we will need a database of some sort. We need this to centralize data, and achieve data persistence. Meaning our data will be stored in a central place, where the data will not depend on whether or not a client program is running. This also ensures that the same data is accessible, if multiple clients demand access to the data.

To achieve this, we are using an SQL server. There is a huge array of options, in choice of an SQL server. Such as: Microsoft SQL Server, MySQL, PostgreSQL, SQLite etc. We have chosen to use a MySQL database [23]. Mainly because this specific platform

is open-source, and widely used for projects of all sizes, and also cross-platform, so it will run on almost any given server environment. MySQL is in the top 3 most popular database solutions, amongst Oracle and MS SQL Server [24]. Therefore, MySQL is more than capable of providing what we need from our database. Furthermore, as it is open-source and as a result of its popularity, finding useful documentation is easily accessible. Our demands from the server are very limited, as we are only expecting to store simple data and images. But as we can pull data from this server, regardless of our client platform, the data can be accessed from multiple types of clients if other platforms were to be introduced later in the product's lifetime. Such as mobile or web based applications.

5.1.2 EntityFramework

We need to communicate between our program and the database. Communication with the database happens through SQL queries. By using a MySQL .NET Connector interface [25], provided by MySQL, we can handle this communication manually in .NET.

But, we have settled on using a framework to take care of the SQL-based communication, meaning we will never write or define any actual SQL commands/queries in our code. This also means that security measures have been taken care of, as well as a slew of other things making it easier. We are using EntityFramework (version 6.1.3), which is an abstraction layer for communicating with SQL databases in .NET. EntityFramework uses the .NET Connector for MySQL to communicate with the database, just as we would normally do manually. But the EntityFramework lies between our code and MySQL Connector, taking care care of all the interfacing inbetween. This makes it so that we will not have to consider SQL syntax. The EntityFramework works behind the scenes, in order to procedurally generate SQL queries for the database. It takes LINQ expressions (lambda), in order to construct SQL queries, and automatically communicate queries to the database.

In listing 5.1 is an example of how the EntityFramework can be utilized to query data. The *ServerContext*-object *ctx* is instantiated. It contains all the connectivity data, as a connection string, etc. This object contains properties for each table in the schema. Therefore we can construct a LINQ expression, as normally when working with Lists. This is the expression that EntityFramework uses to construct a custom SQL-query, which is then sent, using the connection data contained in *ctx*.

Listing 5.1: Example of using EntityFramework

```
1 using (var ctx = new ServerContext())
2 {
```

```
3       List<Product> productlist = ctx.Products.Where(x => x.id ==
            10).ToList();
4   }
```

5.1.3 Windows Presentation Foundation

Windows Presentation Foundation(WPF) is an integrated part of Microsoft's .NET framework and is widely used to create Graphical User Interfaces (GUI) for applications. WPF is fairly new compared to its counterpart WinForms, which, too, is an integrated part of the .NET framework. [26]

We have decided to use WPF for the GUI as it allows a more clean separation between the GUI code and the code of the program, that defines its functionality. If done right, this allows for changes to the GUI code to be applied without having to change any of the functionality code. This is especially important for this project, as it allows for swift changes to the UI, if the target audience disagrees with design choices, without having to rewrite much of the code for functionality. [27].

The layout of the different windows and pages that make up a WPF GUI are written in the XAML markup language. Through this code, the elements on the GUI, (a button for instance) their layout, their positioning is specified. If we then desire to add functionality to a GUI button, for instance, that this button shows how many items are in the store, then we utilize data bindings.

Data bindings work by tethering displays in the GUI to properties defined by the code. The values of the properties can then be shown on the GUI, and even changed, if this is desired. To allow for a property to be used for data binding, we do, however, have to implement the interface *INotifyPropertyChanged* in whichever class the property lies. This interface declares one event, and the point of it is to notify the GUI of changes in the property, if we so desire. Through bindings, we can also allow the user to execute commands in the code, like calculating a certain value every time the user clicks on a button.

5.2 Architecture Design

To design our solution it is first necessary to decide upon the architecture of the system. In this section we will describe which components the system is made up of and which responsibilities they are each intended to manage. Throughout this section we will be transitioning from an abstract description of the system to a more concrete one. We will be focusing on describing the boundaries of the components and what information

they individually have access to, while initially disregarding how this may be implemented. Lastly we will describe how the architecture may be implemented in practice by showing how the system's main would have to look.

5.2.1 Modularity

Because we intend to heavily adapt the GUI to the target audience, the architecture needs to allow for a great deal of uncertainty during the development of the program. As such, the main objective of the architecture is to ensure a high degree of modularity.

A modular design is achieved by separating program concerns into modules that have high internal cohesion, and by limiting coupling between the different modules. The point of this is that the individual program modules can be modified without it affecting the other modules, or requiring changes in these. By following a modular architecture we ensure that the program can easily be changed if the wishes or requirements of the target audience change, for instance, after we test the usability of the system. [28]

To start with, we will be separating the system concerns into components.

5.2.2 Components

The system will follow a client-server architectural style, where the data managed by the system (the products that are currently in the store, for instance) is centralized on the server. The entire system can be separated into components and dependencies, as seen on figure 5.1. [29]

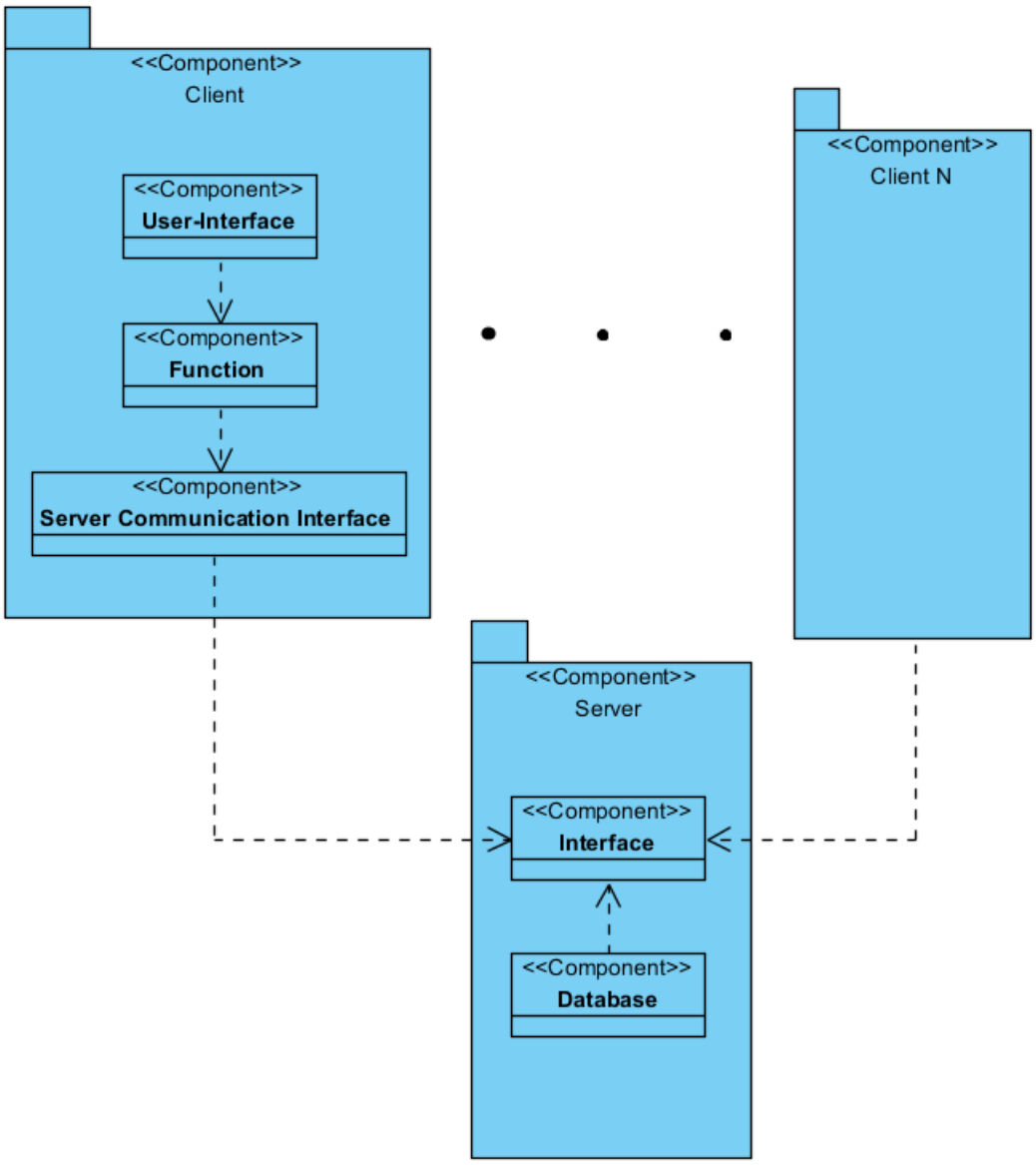


Figure 5.1: The architecture of the system's main components

The system as a whole can be can be split into the subsystems client and server (which contains the database that holds our data). The user is intended to only ever interact with the user interface component. On user interaction this component informs the function component, which starts the specific functions, as mentioned in the function list in section 4.2.3. The functions which require data from the problem domain (for instance the products that are currently in the store) then access the database component on the server subsystem. They do this by sending commands through an interface on the client, which makes sure that the commands to the server are valid and allowed, and an interface on the server, which simply receives and executes the commands. The data gained is then passed back through the components, the result of the function is calculated, and lastly presented to the user interface.

To allow this communication between the components and ensure system modularity in practice, we will be using the MVVM (Model, View, ViewModel) pattern to structure how the program's components interact with each other.

5.2.3 MVVM Pattern

This pattern was not chosen solely because it aims to ensure modularity, however, as other patterns like MVC (Model, View, Controller) serve the same purpose. MVVM in particular was picked, as the way it binds functionality with the user interface synergizes well with WPF, the chosen GUI framework, see 5.1.3.

While the Client-Server pattern heavily focuses on how the data is presented and stored, the MVVM pattern provides us an overview of how we intend to ensure high cohesion and low coupling between the components.

The MVVM pattern splits the program into three separate main modules: the model, the view and the viewModel. See figure 5.2 for an overview of how the different components are intended to interact.

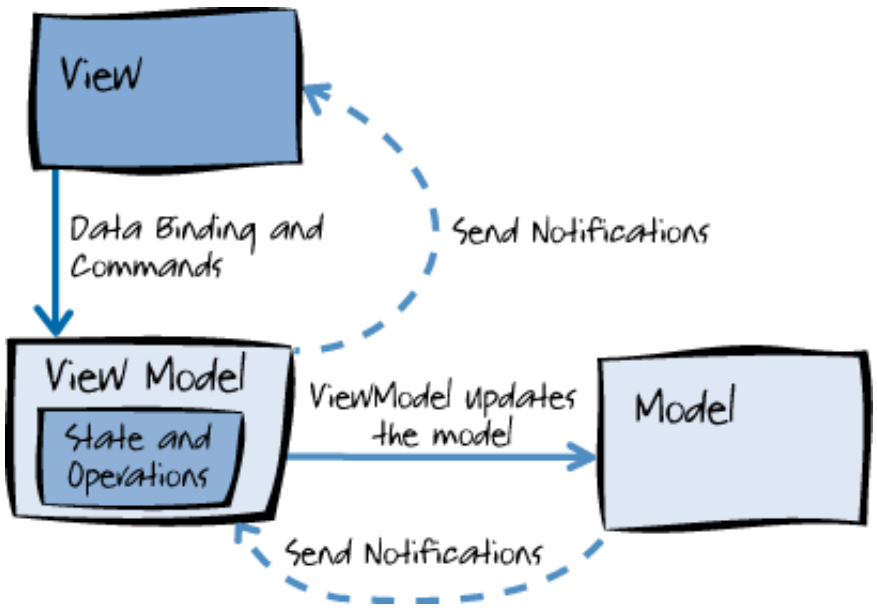


Figure 5.2: An overview of the MVVM Patterns [30]

The model is the implementation of the application domain and contains the system's core logic, algorithms and data. The model's primary functionality is to receive a command from the viewModel and return a value or method, as in figure 5.3.

The View contains the user interface, which, in this case, will be made through WPF. The View-component will ideally not contain any kind of business and View logic, but

only the structure, layout and appearance of what the user sees on screen.

The viewModel is responsible for handling the view-logic and interconnecting the view and model components. The viewModel component is where we may, for example, implement the commands that are triggered when the user clicks on a button in the View component. It may then retrieve data from the model component by invoking certain methods and relay the data to the view component. The viewModel is similar to a simple controller component, which handles requests from the view component and invokes methods on the model component. A controller component is intended to be modular by not having any awareness of which specific view implementation it is displaying its data to. A viewModel is set apart from this by the fact that the viewModel additionally is intended to not have any awareness of which specific model implementation its data comes from either.

Using the MVVM pattern, we can specify how the system component will be interacting with each other and how the system should be separated in practice, to ensure modularity of the finished program, see figure 5.3.

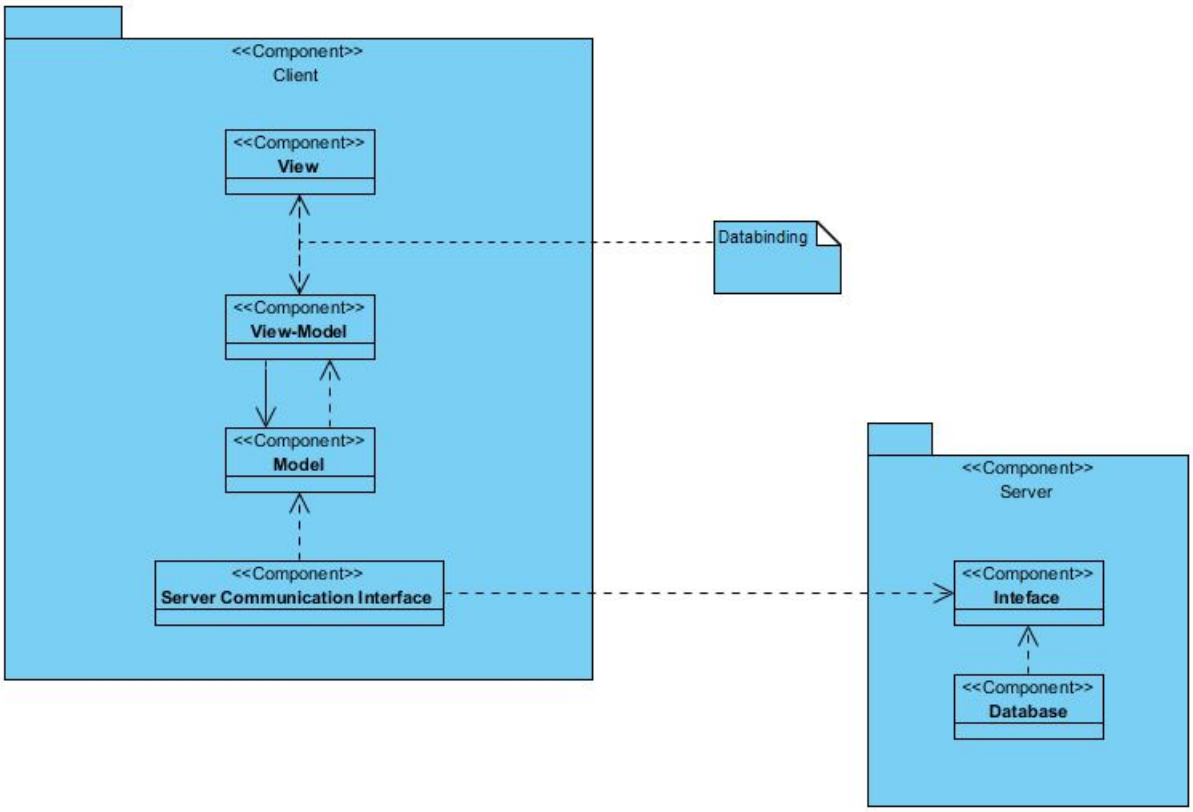


Figure 5.3: The architecture of the system's main components

Here it is specified that the server component will only be used for distributing data from its database. Through the dashed arrows it can be seen that the view and viewModel

will be communicating both ways, while still only being loosely connected, which will be achieved with WPF's databindings.

5.2.4 Program Setup

To illustrate how we will be using this architecture in practice we will show an example of how the modularity and connections are to be ensured in the implementation of the program. Through an example we will show how we intend to separate the view component code from the rest of the program during the final implementation and thereby achieve high modularity. We will be using C# for as the programming language. The View component will be created through the WPF framework, and will therefore be written largely in the XAML markup language.

WPF views are separated into XAML files, which dictate how the view looks, and a code-behind file, which allows for the coding of functionality and logic. For the sake of high modularity and proper separation of view and business logic, we will attempt to keep these code-behind files as empty as possible.

Contrary to WinForms view frameworks, WPF applications are not started through a traditional program main. The view is instead executed in an App.xaml file, which is what mostly resembles a main in WPF. The App.xaml file is automatically generated by Visual Studio, and can be programmed in as normal, but usually does not require any modifications for simple programs. The main use of the App.xaml file, except for starting the WPF view, is for defining resources (view styles for instance) that are used across the entire application. When creating a new application, the automatically generated App.xaml will look similar to listing 5.2. [31]

Listing 5.2: WPF App.xaml

```
1 <Application
2     x:Class="WpfTutorialSamples.App"
3
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5     StartupUri="MainWindow.xaml">
6
7     <Application.Resources>
8     </Application.Resources>
9 </Application>
```

The StartupURI property on line 5 defines which view is started when the application is launched, in this case the MainWindow.xaml view. In this view we may then design

buttons or textboxes, through which we intend to expose functionality to the user. To then connect button clicks to the viewModel classes, without any direct referencing, that may harm the program's modularity, we utilize observable objects.

See listing 5.3 for a typical implementation of the *INotifyPropertyChanged* interface, which allows us to expose properties that the View component may then use. [32]

Listing 5.3: INotifyPropertyChanged

```
1  class PersonViewModel : INotifyPropertyChanged
2  {
3    public event PropertyChangedEventHandler PropertyChanged;
4    protected virtual void OnPropertyChanged(string propertyName)
5    {
6      this.PropertyChanged?.Invoke(this, new
           PropertyChangedEventArgs(propertyName));
7    }
8
9    private string name;
10   public string Name
11   {
12     get { return this.name; }
13     set {
14       if (value == this.name)
15         return;
16       this.name = value;
17       this.OnPropertyChanged(nameof(this.Name));
18     }
19   }
20 }
```

On line 17, the Name property is exposed to the View, which allows us to tether elements in the View to it. The data binding itself is defined by the XAML code on listing 5.4 in the View [32].

Listing 5.4: Data binding

```
1  ...
2    <Window.DataContext>
3      <viewModels:PersonViewModel />
4    </Window.DataContext>
5    <TextBlock Text="{Binding Name, Mode=TwoWay}" />
6  ...
```

The DataContext on lines 2-4 instructs where the view should be searching for valid bindings, and line 5 of code defines that the text contained within the view's TextBlock should be two-way bound to the Name property.

This is the gist of how we are attempting to ensure modularity in the program. In practice this approach will work for properties, but will be different for commands (specifically button clicks). Commands will be bound to the viewModel through an additional layer, the RelayCommand class, which forwards ICommands to the viewModel, and will then execute these.

The program's raw data will be kept in a database on a server, but the program's functionality and GUI will be kept locally on every program client. Prior to implementing the program, we will be exploring the components of our client and their contents in greater detail.

5.3 Component Design

Using the design of the system's architecture, we will in this section plan the design of the individual components. We will cover how we choose events and functions to include, and which ones we chose. Lastly, we will describe how we are planning to connect the different components. The result of this section is a detailed class diagram of each component, including their attributes and operations, which will be used as a foundation for the implementation of the components.

Events

The first thing we are going to create is the design for the model component. To do this we are going to take a look at the common and private events from the table 4.1. These events will either be modeled as attributes in the different classes seen in figure 4.1 or be put in new classes as attributes.

Private Events

Private events are events that involve only one object from the problem domain.

- *Product outdated*
- *Store section created*
- *Store section closed*
- *Standard prices revised*

Product outdated happens in sequence and only once, which means that we will need another attribute for our product class. The same thing is true for store section created and closed, therefore this class will also need a new attribute. Standard price revised is an iterative event, which means that we need a new class.

Common Events

Events that occur only once

- Product returned
- Product sold

Can add attribute to Product, or transaction class. We added them to product class.
Events that occur once and multiple times, depending on the object

- Product registered
- Product unregistered

These two events are very similar and can represented as attributes for the product class, in the form of a "state" variable.
Events that occur multiple times for multiple objects

- Daily sales accounted
- Monthly sales reported

Functions

We will also have to incorporate the functions listed in table 4.3 and 4.4. The functions listed in these tables will be split up and be incorporated in both the model component and the viewModel component. The reason why some of the functions will be contained in the model component instead of only being in the viewModel component is because some functions only operate on the one object where the function is contained, e.g. the *Edit Product* function in the *Product* class. A unique case where a function is stored in the model component is the *Create Voucher* function in the *Return Transaction* class. The reason why this function is here is because a *voucher* object is always going to be created when a return transaction occurs.

The more general functions that create objects in the model component, other than the *Create Voucher* function, will be stored in the viewModel component. An example of this could be the *Register Product* function in the *Product Changes* class which is used to create a new *Product* object that is stores in the model component.

Connecting Components

Now that attributes and functions have been assigned to the different classes the the model and the viewModel component can now be connected through their classes. The dashed lines seen in figure 5.4 represents the creation and removal of objects of the different classes in the model component as well as the fetching of data, in the form of objects, from the model component, through this the components are connected.

The classes on the figure can be used as a starting point for interfaces for the program during implementation.

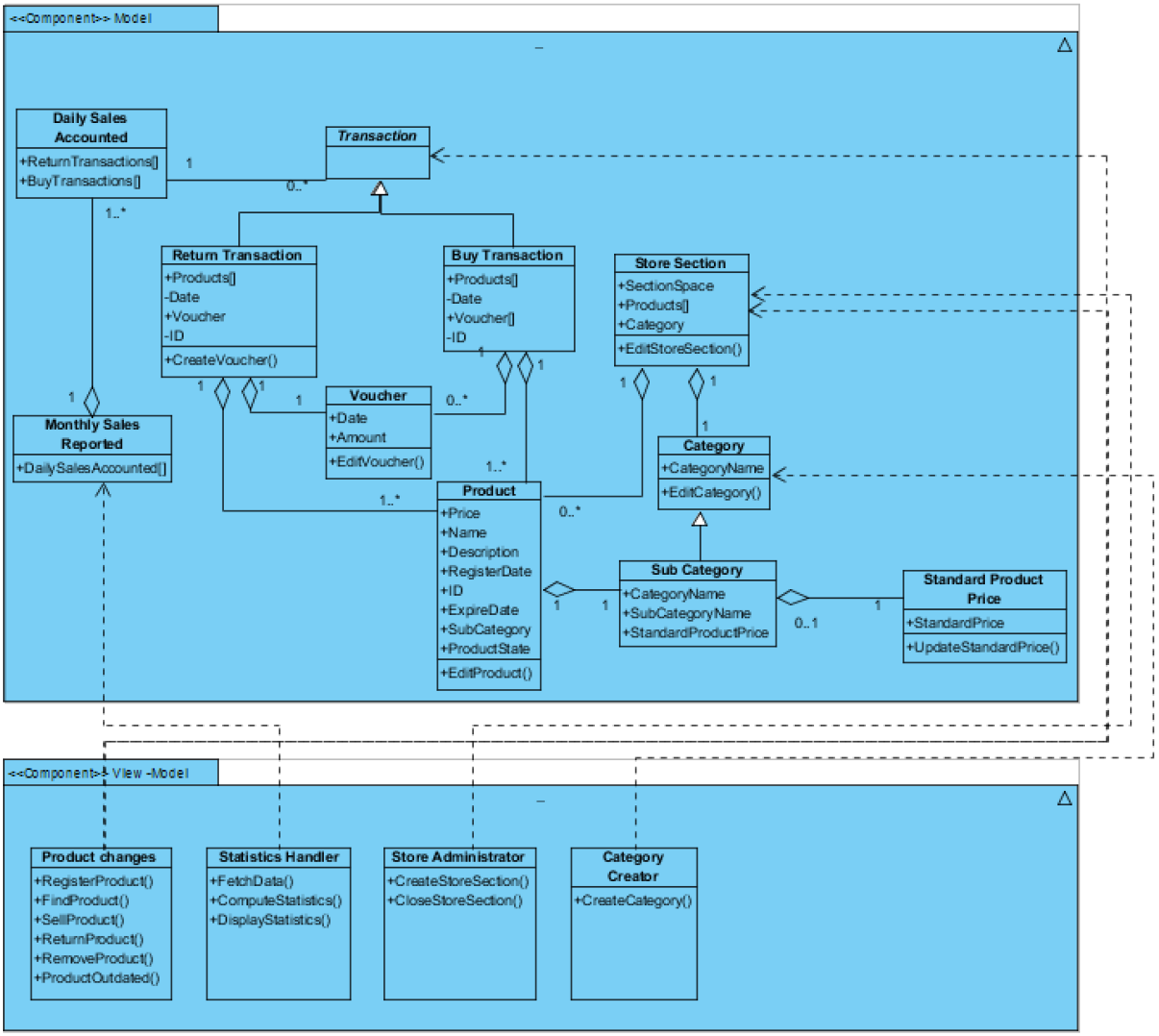


Figure 5.4: Detailed class diagram with attributes, functions and connected components

Chapter 6

Implementation

This chapter will consist of the documentation of the implementation of the inventory management system. The documentation will primarily be split into sections: 6.1 for the model (database) in the system, 6.2 for the viewModel in the system and 6.3 for the user interface in the system. Some of the sections in the chapter will contain snippets of code from the system which will be used in a relevant context. Screenshots of the user interface from the system will also be used to explain some of the functionality in the system.

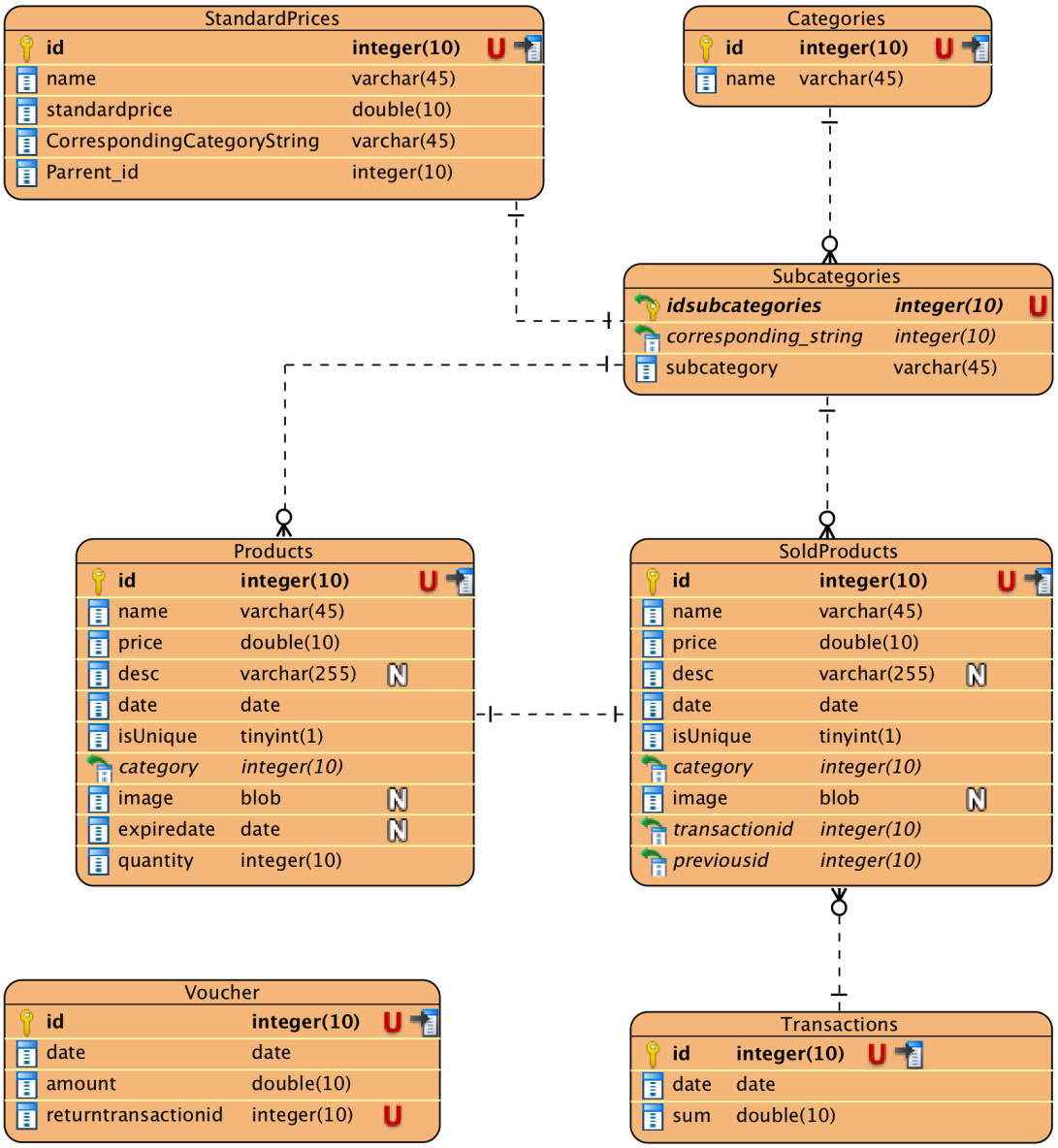
6.1 Model Implementation

In this section we will cover the database setup and the interaction between the database and our codebase.

Data Types

We are defining data types for each column in each table. These types define whether the field contains, *string*, *int*, *double*, *blob*, etc. We also set up rulesets for each of these data columns, such as whether or not they can be *NULL*, or define if they should *auto increment*, or some custom behavior, such as generate a *timestamp*. These rules are enforced by the database server, so that we cannot enter invalid data, that will later cause issues. This means that columns like *name*, *id* and *date* cannot at any point be *NULL*, as they are either required or set to be automatically generated. These tables are represented in C#, by auto generated classes, generated by the EntityFramework, based on the schema setup.
In figure 6.1, an overview of our tables can be seen, as well as their relations to each other. Meaning, whether it is *one-to-one*, *one-to-many* or *many-to-many*.

Figure 6.1: Overview of database structure. Boxes represent tables, and relations between these are shown with lines.



Communication

We need classes in our program, which reflect the tables in the database, which are auto generated. ADO.NET is a Microsoft .NET data access technology, which is used for communicating between relational and non-relational systems. Our *Product* class is used within *ObservableCollection<Product>*. This is due to our use of WPF for the GUI, which means we have to work within the confines of what the technology is bound to, by convention.

Data returned from the database is stored in classes. For example, we have a *Product* class, containing properties mapped to mirror those in the *Products table* on the

database. This class can also contain unmapped properties, which will then be ignored by the EntityFramework. Our *Product*-class is shown in listing 6.1. This class is largely auto generated, meaning ADO.NET has mirrored the database setup as mapped properties in classes. This is what EntityFramework takes, and uses as a mapping for the database communication.

Listing 6.1: Product class - largely auto generated by ADO.NET, based on an existing database table

```
namespace DanmissionManager
{
    [Table("data.products")]
    public partial class Product
    {
        public int id { get; set; }

        [Required]
        [StringLength(45)]
        public string name { get; set; }

        public double price { get; set; }

        [StringLength(255)]
        public string desc { get; set; }

        [Column(TypeName = "timestamp")]
        public DateTime? date { get; set; }

        public bool isUnique { get; set; }

        public int? quantity { get; set; }

        public int category { get; set; }

        [Column(TypeName = "mediumblob")]
        public byte[] image { get; set; }

        public DateTime? expiredate { get; set; }

        [NotMapped]
        public BitmapImage productImage { get; set; }
    }
```

```
}
```

Several places in the program, we are querying the database. In listing 6.2, is an example of one of the central places where we use the EntityFramework to request data from the database, and store it within C# datatypes. This specific implementation is inspired by a workshop we attended, by our 2nd semester OOP lecturer. We start by instanciating *ServerContext()* as *ctx*. The instance of *ServerContext* contains properties (as *Lists<T>*). We use this to 'download' to a local list, where we then process some image data, and finally assigning the data to the *ObservableCollection<Products>*, which the WPF listview is bound to.

Listing 6.2: An example of a database query, using EntityFramework

```
using (var ctx = new ServerContext())
{
    List<Product> list = ctx.Products.Where(x =>
       x.name.ToLower().Contains(SearchParameter.ToLower()) ||
       (x.id.ToString()).Contains(SearchParameter.ToLower()) ||
       (x.price.ToString()).Contains(SearchParameter.ToLower())).ToList();

    foreach (Product x in list)
    {
        if (x.image != null && x.image.Length > 0)
        {
            x.productImage = ImageFromBuffer(x.image);
        }
    }

    ObservableCollection<Product> collection = new
       ObservableCollection<Product>(list);
    this.Products = collection;

    if (this.Products.Count == 0)
    {
        MessageBox.Show("Din soegning gav ingen
           resultater.","Desvaerre");
    }
}
```

6.2 ViewModel Implementation

The viewModel component in the program consists of classes that operate on the model component to fetch, upload and calculate relevant data which is then displayed in the different views of the program. Relevant code, such as the different types of methods, from the different classes in the viewModel can be seen and will be explained in the subsections below. Not all of the viewModel classes will have their own subsection, the reason for this is because the contents of some of the different classes are very similar but used for different tasks.

6.2.1 Register Product

In this subsection we will be taking a look at the interesting sections of code from the *CreateProductViewModel* class. This class contains the functionality for registering/creating a product within the program. The way this is done is that the user inputs the required information about the product through the *addProductPage* in the program, which then is stored on the server and given a specific id. The method *AddProduct* that is used to do this can be seen in listing 6.3.

Listing 6.3: Method for Registering products in the store

```
1  public void AddProduct()
2  {
3      Product product = new Product(this.ProductName,
          this.SelectedSubCategory.Parent_id,
          this.Product.isUnique, this.ProductDesc);
4
5      if (product.isUnique == false)
6      {
7          product.quantity = this.AmountOfProducts;
8      }
9      else
10     {
11         product.quantity = 1;
12     }
13
14     if (this.Weeks > 0)
15     {
16         product.expiredate =
              product.date.Value.AddDays(this.Weeks * 7);
17     }
18     else
```

```
19      {
20          product.expiredate = null;
21      }
22
23      if (Image != null)
24      {
25          product.image = this.imageToByteArray(Image);
26      }
27      if (this.Product.price.Equals(0.0) == true)
28      {
29          product.price = this.SelectedSubCategory.standardprice;
30      }
31      else
32      {
33          product.price = this.Product.price;
34      }
35      try
36      {
37          using (var ctx = new ServerContext())
38          {
39              ctx.Products.Add(product);
40              ctx.SaveChanges();
41              MessageBox.Show("Assigned ID: " + product.id,
                    "Success!");
42          }
43      }
44      catch (System.Data.DataException)
45      {
46          PopupService.PopupMessage(Application.Current
47          .FindResource("CouldNotConnectToDatabase").ToString(),
                Application.Current.FindResource("Error").ToString());
48      }
49  }
```

In the method *AddProduct* the first thing to happen is that a product is created with the relevant attributes such as *name*, *id* and so on, this can be seen in line 3 of listing 6.3. After this the different properties for the product is checked, for example if the product has a *price* of 0 it gets the *standard price* for the category it belongs to. Finally the product is added to the server and if everything goes well a message box will be shown with the note *Success!* as well as the product's *id*, if not a messageBox will show telling otherwise.

6.2.2 Statistics

Here we will cover the implementation of the viewModel class for calculating statistics, the *StatisticsViewModel*. This class contains methods for the calculation of the different types of statistics. The different types of statistics are: How much revenue the different categories have generated, how many items that have been sold within the different categories and the number of items belonging to the different categories are in the store. A method for the latter statistic called *ShowChartInventory* can be seen in listing 6.4. The other two types of statistics have methods that look very similar to the one seen in listing 6.4.

Listing 6.4: Method for finding products in the store

```
1  Private void ShowChartInventory()
2  {
3      List<KeyValuePair<string, int>> inventoryValue = new
           List<KeyValuePair<string, int>>();
4      foreach (Category category in AllCategories)
5      {
6          int numberOfProducts = 0;
7          foreach (Product product in AllProducts)
8          {
9              if(category.id == product.category)
10             {
11                 numberOfProducts++;
12             }
13         }
14         if (numberOfProducts != 0)
15         {
16             inventoryValue.Add(new KeyValuePair<string,
                   int>(category.name, numberOfProducts));
17         }
18     }
19
20     PieChart = inventoryValue;
21 }
```

A brief explanation of the method seen in listing 6.4 is that to begin with we declare a list *inventoryValue*, as seen in line 3. Then we begin to iterate over the different categories contained in the list of categories *AllCategories* in line 4. For each of the the categories we iterate over all of the products found in the list of products *AllProducts* from line 7, and if the category's *id* and the product's *category* match, we increment the

integer *numberOfProducts* in line 11 and then adding this value along with the category *name* to the list *inventoryValue* successfully counting the number of products belonging to each category. And finally the list is assigned to the property *PieChart* seen in line 20.

The code for the property *PieChart* in the program can be seen in listing 6.5. As seen in the listing the *PieChart*-property is a list with a backing field. When the property is set, for example on line 20 in *ShowChartInventory* in listing 6.4, the backing field is assigned to a value in the form of a list and the method *OnPropertyChanged* is called with the string *PieChart*. This changes the pie chart in the relevant View to its newly assigned value if it exists.

Listing 6.5: The pie chart property in the program

```
1  private List<KeyValuePair<string, int>> _pieChart;
2  public List<KeyValuePair<string, int>> PieChart
3  {
4      get
5      {
6          return _pieChart;
7      }
8      set
9      {
10         _pieChart = value;
11         OnPropertyChanged("PieChart");
12     }
13 }
```

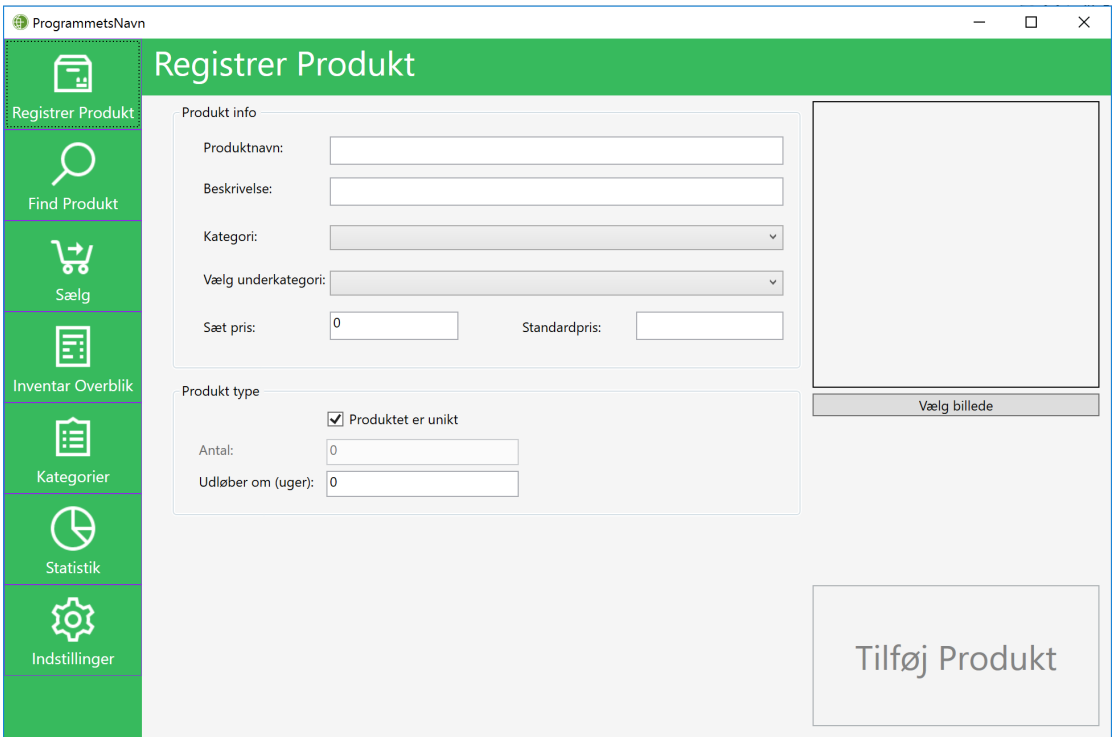
6.3 View Implementation

A view is made up by elements such as buttons which are made using Windows Presentation Foundation and the XAML markup language which we will go over.

6.3.1 Use of XAML

As previously mentioned XAML is used. In our case the use of XAML is required because it is used with Windows Presentation Foundation with which all the views in the program were made. To demonstrate the use of XAML we can take a look at the menu used to add products from the program, this menu can be seen in figure 6.2, this menu was chosen because it contains a lot of the different elements used in many of the other menus.

Figure 6.2: Image of *addProductPage* from the program



An example of the use of XAML code for an element on the could be for the *Choose image* button located to the right on figure 6.2, this code can be seen in listing 6.6.

Listing 6.6: XAML code for *choose image* button

```
1     <Button
2         x:Name="btn_ChooseImage"
3         Command="{Binding Path = CommandGetImage}"
4         Content="Vaelg billede"
5         Margin="0,312,10,0"
6         Width="252"
7         Height="20"
8         VerticalAlignment="Top"
9         HorizontalAlignment="Right"
10    />
```

To briefly explain the XAML code seen in listing 6.6 the first thing to happen is that a button is declared, this button is then given a name, which can be used to find it in the code behind file. The button is then given a property path so that it can be linked to a property in other classes within the program. The content of the button is then assigned, this determines the text shown on the button. A margin is also set for the button, this determines the distance from the four edges of the window. In this case the button is 10 units from the right side of the window and 312 units from the top at all times regardless

of the size of the window. Where the values are zero mean that they are not bound to the corresponding edge. The horizontal and vertical alignment determines where the margin is calculated from and the width and height determine the dimensions of the button.

Text boxes are also used, for example to give a product a selling price, the XAML code for this text box from the menu seen in figure 6.2 can be seen in listing 6.7.

Listing 6.7: XAML code for a text box being used to set a price

```
1   <TextBox
2       x:Name="TextBox_setPrice"
3       Text="{Binding Product.price, Mode=TwoWay,
            UpdateSourceTrigger=PropertyChanged}"
4       HorizontalContentAlignment="Left"
5       Margin="133,168,0,0"
6       Height="25"
7       Width="113"
8       VerticalAlignment="Top"
9       HorizontalAlignment="Left"
10  />
```

The XAML code seen in listing 6.7 is very similar to the code for the button seen in listing 6.6 in that this also has a name and their design features such as margin, alignment and so on. One difference in listing 6.7 can be seen in line 3 where a *Binding*, *Mode* and *UpdateSourceTrigger* is set. The binding determines which property the text box will be bound to, in this case it is *Product.price*. The mode is in this case set to twoway, this means that if the value of the text box is changed the code will update and the other way around. This works because of UpdateSourceTrigger, this is assigned to the event *PropertyChanged* which ensures that the property *Product.Price* is changed in the code when the text box is written in.

To select things such as a category when you want to register a product a combo box is used. The XAML code for a combo box can be seen in listing 6.8.

Listing 6.8: XAML code for a combo box being used to select a category

```
1   <ComboBox
2       x:Name="comboBox_category"
3       ItemsSource="{Binding Path = Categories, Mode=TwoWay,
            UpdateSourceTrigger=PropertyChanged}"
4       DisplayMemberPath="name"
```

```
5           SelectedItem="{Binding SelectedCategory, Mode=TwoWay}"
6           Margin="133,92,10,0" VerticalAlignment="Top"
7       />
```

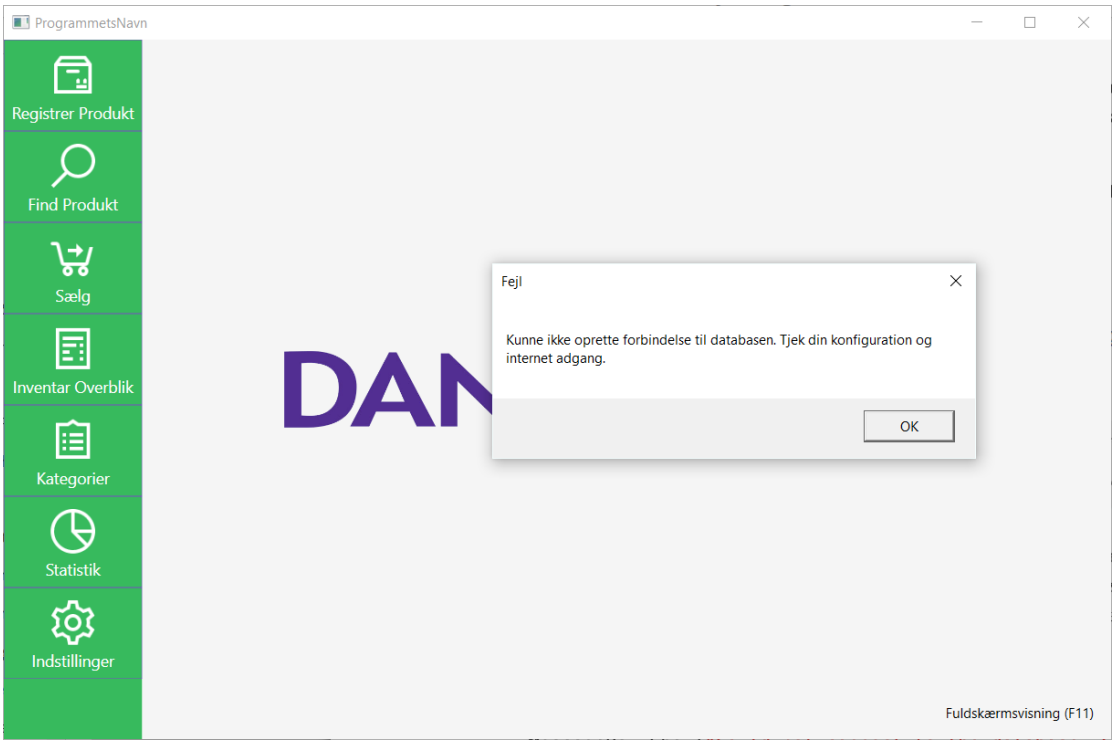
We see that the comboBox in listing 6.8 is also given a name and some design features as well as a binding path, a mode and so on. The things that are new in this XAMl code is *DisplayMemberPath* and *SelectedItem*. *DisplayMemberPath* determines which property is displayed in the combo box, in this case it is the name property of from Category. *SelectedItem* determines what the combo box displays once a category has been selected.

6.4 The Program Architecture in Retrospective

The program architecture was meticulously planned, but naturally this was done using only the knowledge and information that we had beforehand. However, because of the sheer amount of tasks to be implemented, as well as their individual complexity, it is very easy to run into architectural issues that were not apparent initially. These may cause simple and/or mundane parts of the program to become a lot more complex than expected. We will describe the shortcomings and problems that we encountered in regards to the chosen program architecture, and how we circumvented the problems. Specifically we will focus on the MVVM architecture pattern and describe if and how we were forced to break the rules and guidelines of the pattern. We will be using our issues during the implementation of error messages as an example of this.

We met the real first shortcoming of the MVVM architecture pattern while attempting to utilize the standard windows implementation of MessageBoxes to notify the user of errors during program execution. See figure 6.3 for how we intended for the popup boxes to look in the finished program, which we did succeed with, if the user tries to access the database without Internet access.

Figure 6.3: Popup message box showing on failed attempt to connect to database



The main point of using the MVVM pattern for our program architecture was to facilitate a complete separation of business logic from the GUI logic. This does however mean that the usual implementation of message boxes, see listing 6.9, is not allowed.

Listing 6.9: Calling the static MessageBox.Show method

```
1 using System.Windows;
2 public MainViewModel()
3 {...
4     MessageBox.Show("Could not connect to the database. Check
          the database configuration and Internet access.",
          "Error");
5 }...
```

The reason it is not allowed is because it is pure View logic that is placed within the viewModel. There are many ways of implementing message boxes properly into programs following the MVVM architecture pattern, but the fact remains that MVVM does not inherently work well with message boxes. Many of these solutions still breach the MVVM rules, however, or bring other issues with them. Those that do not, seem to be significant coding projects on their own or require major restructuring of the program. This includes the use of a navigation service or an implementation of the mediator pattern. As the message boxes were not a major concern during the implementation, it is a task we did late, which meant that restructuring of the program was not a very enticing prospect. The least invasive method of doing it, which we ended up using, appeared to

be injecting the Windows MessageBox methods into the viewModels in the App.xaml.

First we changed the program to start through the App.xaml's code behind, see the implemented code-behind on listing 6.10

Listing 6.10: App.xaml's code behind

```
public partial class App : Application
{
    private void OnStartup(object sender, StartupEventArgs args)
    {
        // MessageBox
        var popup = (Action<string, string>)((msg, capt) =>
           MessageBox.Show(msg, capt));

        // confirm box
        var confirm = (Func<string, string, bool>)((msg, capt)
           =>
            MessageBox.Show(msg, capt, MessageBoxButton.YesNo)
               == MessageBoxResult.Yes);

        var popupService = new BaseViewModel.Popups(popup,
           confirm);
        views.MainView view = new Views.MainView(popupService);
        view.DataContext = new MainViewModel(popupService);
        view.Show();
    }
}
```

Here we inject the static windows MessageBox methods as delegates into the viewModel. As such, the MainViewModel knows nothing about how the popup boxes are handled, which was the intend. As one may notice, however, we also inject the MessageBox methods into the View itself. Now, injecting it into the View is fine, as the MessageBoxes are purely GUI logic. The problem resides within the reasoning for this injection.

When using the MVVM architecture pattern, a general guideline is to keep the code behind the Views as empty as possible, as it ensures that no business logic is contained within it. Before we added MessageBoxes, our MainView's codebehind contained only the fact that each button click would open a new page of the program (as well as adding a keybinding that toggles fullscreen, which is of little interest here). See listing 6.11 for one of the MainView's button events, contained in the code behind prior to the

implementation of the popup messages.

Listing 6.11: Code behind MainView for clicks on the inventory overview menu button

```
private void btn_inventoryOverview_Click(object sender,
    RoutedEventArgs e)
{
     Main.Content = new inventoryOverviewPage();;
}
```

The individual pages would then, through their XAML code bind themselves to their corresponding viewModels as DataContext, see listing 6.12 for example:

Listing 6.12: XAML Code instantiating and binding to a viewModel

```
<Window.DataContext>
    <local:MainViewModel/>
</Window.DataContext>
```

This instantiation was no longer possible through XAML, because we now require the popupServices as parameters in the viewModel constructors. Instantiating the View DataContext with parameters is tricky, though seemingly not impossible, using only XAML. For this reason we removed the XAML code on listing 6.12 and changed the MainView's code behind to contain the binding of viewModels to the Views, see 6.13.

Listing 6.13: MainView's new code behind, containing viewModel being bound to Views

```
private void btn_inventoryOverview_Click(object sender,
    RoutedEventArgs e)
{
     var newpage = new inventoryOverviewPage();
     newpage.DataContext = new
        InventoryOverviewViewModel(_standardPopupService);
     Main.Content = newpage;
}
```

The problem with this approach in regards to the MVVM architecture pattern is that we have added additional code into the code-behind, and that the MainView now contains references to the viewModels. The code for instantiating and binding the viewModel to the View, see 6.12 for the XAML version and 6.13 for the C# version, are equivalent in all but the exact syntax. Using the C# version over the XAML version does not cause higher coupling (lower modularity) of the view and viewModel components. The View would have to be bound directly to the viewModel at some point. Whether this happens in declarative XAML code or object-oriented C# code should not cause one to be coupled more tightly than the other. For this reason we deemed that both of these MVVM breach issues were minor.

There are other places in the program where we similarly do not keep as closely to the MVVM pattern as we had planned, but, barring mistakes, the reasoning for those choices is very similar to the reasoning behind the breach of guidelines while implementing these popup MessageBoxes.

6.5 Internal Testing

Internally we have been white box testing the program all throughout its implementation. This means that we, who are well versed in the program's inner workings, have been testing the different parts in an unstructured manner whilst creating them.

Additionally, as intended by proper MVVM modularity, we have attempted to make room for unit testing. A unit test is a piece of code intended to test only a very small part of the program's behaviour. For instance, a unit test may verify whether the method EmptyList(list<T>[]) is truly an empty list. As such, the program is broken down into small behaviour "units", which can then be answered with either a yes or a no. Having an amount of unit tests covering a large amount of the program code helps keep detection of where bugs do (or do not) originate from simple, even if the program grows complex. The downside of unit testing is how time consuming it is.

The issue with unit testing in a program like ours is that a lot of functions require user input, which is not allowed for unit testing. The solution is to allow substituting the functions that require user input with functions that do not. In the program we have, for instance, the MessageBoxes as mentioned in section 6.4. To then test functionality that usually requires the user to go react to a MessageBox first, we can, for instance, substitute "empty" methods during the creation of the viewModel instance. See listing 6.14

Listing 6.14: Dummy delegates unit for the popupService

```
1 ...
2 var dummyPopup = (Action<string>)((a) => {return;});
3 var dummyConfirm = (Func<string,string,bool>)((a,b) => {return
     true;});
4 var popupService = new BaseViewModel.Popups(popup, confirm);
5 var view = new Views.MainView(popupService);
6 ...
```

The user is now no longer prompted when the popupService methods are called. Although we had this possibility, we did not actually end up using unit tests at all. One reason for this was that the different parts of the system were very modular and separate to begin with. There was no complex algorithm, where we had to make sure that the

data stays intact through all the steps involved, for instance.

For one, the pages of the View held practically no view functionality that had to be tested. Additionally, the logic in the different viewModels was often trivial, as the viewModels and their pages in general only had one job each. It was, as such, very simple to deduct whether the different viewModels and pages were functional or not. The issues that we did encounter while implementing the system were mostly concerning how to implement functionality without breaking the MVVM rule set.

6.6 Usability

To ensure that our solution was properly usable by the people at the Danmission store, we prepared and conducted a test and an interview. Here we attempted to gain feedback concerning the program's ease of use, as well as feedback on existing or missing functionality. We also had some questions regarding the GUI as a whole, and how to make it more aesthetically pleasing.

The following is an example of how we formulated the assignments that we would ask the participants to carry out. The first part is a brief description of the assignment and the *assignment text* covers what we would actually tell the participant.

Assignment 5 - Statistics and Transactions For this assignment the participant will be asked to provide some information about the stores' sales and transactions. Currently the system has some generated data, which we will use to calculate these statistics. The transactions can be found in the inventory overview tab.

Assignment text:
For this assignment you need to find out how much money the store made from selling male clothing from 12./1. to 12./12. Afterwards find the most profitable transaction.

The remaining assignments and questions can be found in the appendix G. The assignments were designed based on the strict requirements from section 2.6. For instance the requirement that the program can gather statistics has a corresponding assignment, the example assignment, in which the participant has to use the program's statistic page to gain information concerning the products in the store.
The test itself was done in three steps. First a short introduction to the program and its functionality, which took around 10 minutes. Then the participant would go through the assignments and discuss these with us, in which we would sometimes guide the participant. The usability test/interview was scheduled to have included 4-5 people from Danmission, but all of them except one had to cancel due to reasons such as sickness,

and we did not have time to reschedule. Consequentially, our test will probably not uncover all usability problems, as this usually requires at least 5 participants [33]. The test/interview was conducted in the store itself.

6.6.1 Results and Findings

This section will cover the results and findings from our usability test.

1 - Multibuy

When unregistering products that had been sold, the participant mentioned a situation in where the store had sold 30 plates to the same customer. The participant commented that it would be very difficult to unregister this amount quickly and efficiently, because it is only possible to add 1 product at a time. The participant also found it difficult to see how many of the same items were added to the virtual basket.

2 - More ways of searching

To find the correct product in the system, the participant would have to know the id of a product, which would be difficult to remember, if there are more than just 10 products.

3 - Color coding of buttons

With regards to the interface, the participant mentioned that color coding buttons might make it easier to remember what they did. For example, buttons that add things to the database would have a certain color, while buttons that remove things have another color.

4 - Never expiring products

The participant mentioned that it should be possible to register a product without this product having an expire date.

5 - Font sizes

During the entire usability test the participant had trouble with reading the text on screen.

6 - Expired products tab

When the participant had to find an expired product and remove it, the participant commented that it would be nice with a button in the expired products tab that could quickly remove expired products.

6.6.2 Implementing changes

We have implemented changes to try and remedy usability problems 1, 4, and 6 from section 6.6.1. We have not, however, retested usability with the participant to ensure

that these problems have truly been solved.

For the first problem we added an extra textBox, in which a user can input the amount of items to sell. For the fourth problem we have added the ability to set a product's expiration date to null, which would cause the product to never expire in the system. For problem 6 we implemented another button to the expired products tab, which should make removing expired products more convenient, because one would not have to change pages.

6.6.3 Evaluation

The type of test we aimed at conducting is called an assessment test, which often takes place in the beginning/mid part of building the system. This test is typically followed by a validation test and is conducted at the end of product development [34]. However, we would not be able to perform a validation test, considering the limited data we acquired from the assessment test, because it is not enough data to create a usability benchmark. On the other hand, the assessment test was useful, since we did identify critical usability problems.

Chapter 7

Project Assessment

We will evaluate the program and project as a whole and discuss to which degree we have satisfied our system requirements, system definition and problem statement. Through this we will decide whether the problem can be considered solved. Additionally, we will explore what we see as the largest issues of the project and what could or should have been done differently to avoid these.

7.1 Program Evaluation

We will sum up and evaluate how the finished program ended up. On figure 7.1 the finished program is seen.

Figure 7.1: Page shown on program execution

The pages can be selected on the left side. The program will then, if necessary, connect to the database, its details configured in the options. Information about the items of the store are kept on the database and this can be browsed, edited, added and removed through the different pages of the program. Below, we will mention functionality that may not be immediately apparent by looking at the names of the pages.

The inventory overview page, for one, not only shows information about the store's current inventory, but also shows information about outdated products, transactions and the products that have been sold in the past.

The statistics page shows information about the store's current inventory and how it is divided into the different categories. Additionally, it shows information about the store's transactions, to allow the store to figure out which items are the most profitable.

The options page allows the user to switch the language of the program between English and Danish, as well as customize the default expiration date of products.

7.1.1 System Requirements

Here we will go through the system requirements from section 2.6 to determine whether they have been met.

Strict: Gather statistics

The program does generate statistics, which may be very interesting to the management of the store. However, the shortcoming of the program's functionality concerning statistics is that it's not very customizable. There are at this moment only three things that the user may sort by, and the results are only ever shown in a pie chart. In retrospect, we should have put greater emphasis on allowing the user to show statistics for more metrics. As a whole, however, this requirement can be considered fulfilled.

Strict: Notify user about outdated products

The finished program does not actively notify the user of outdated products. That said, the outdated products can be easily found and removed on the inventory overview page. Strictly speaking, this requirement has not been met. This functionality was, however, intentionally omitted, as we found it ran a high risk of appearing annoying to the user. We assess that the way we present information about outdated products in the finished program is a better solution.

Strict: Bookkeeping of transactions

This requirement has been met, as the system is made to keep track of both transactions and previously sold products indefinitely.

Soft: Allow showing items on a website

The program does not contain any functionality to specifically cooperate with a website, and as such, this requirement has not been met. In the way the program architecture and the server have been set up, however, there does not appear to be anything that may make website functionality any difficult to implement either.

Soft: Printable identification of products

As we ended up using simple numbers as identifiers for the products, this requirement has sort of been circumvented. The program does not contain any specific functionality that may allow the printing of identifier labels. The time it takes to label a product with an id, however, is very low. This requirement and the reason we had for circumventing it will be discussed in greater detain in section 7.2.

Soft: Allow the switching of languages in the program

This requirement has been fully met, as the user may switch languages in the program options.

Additionally, the system definition in section 2.5 mentions that the program has to be able to run on a cheap laptop. We did not test the program on the Danmission store's laptop, so we cannot determine whether this quality has truly been achieved. However, the program has run consistently on 5 laptops with varying specifications and quality. At no point during this process, have we had issues with the program that were caused by computer performance.

All in all, there are a few requirements that have not been properly fulfilled. However, we do not find that these unmet requirements significantly decrease the quality of the product.

7.2 Discussion

We will discuss the weaknesses of the project and the final program. We will be examining the severity of these issues, their causes, possible ways of mitigating them and how they affect the viability of the system. See section 6.4 concerning the weaknesses concerning the program architecture, as we will not be discussing these further.

Program Loading Times

The program suffers from a slow loading time once for each program execution. The first time the database connection is being opened takes a lot longer than subsequent operations. We expect that it has something to do with the inner workings of the Entity framework, which we are using to establish database connections. The speed of this first connection seems to be very heavily affected by the speed of the device's connection to Internet. This was especially noticeable when we

were usability testing the program at Danmission, using only a mobile 3G connection to the Internet.

We do not, however, intend for the program to be constantly opened and closed on the store's computer. Instead, we expect that employees would simply leave it running and ready to use for the registration and unregistration of wares. This means that the loading time would only rarely be a problem. The most problematic part about the loading times is that it will appear to be a lack of responsiveness to the employees, who are not used to IT-systems. This issue could, however, be remedied by adding a loading screen or something similar to show that the program is still working as intended.

The MVVM Pattern

We have used the MVVM system architecture pattern throughout the design and implementation. We have, however, gone against the principles of the pattern in certain parts of the program. For most of these breaches, the reasoning was that keeping to the MVVM pattern would have taken a lot longer and increased the complexity of the program more than the alternative would. We never did end up having to implement large changes to the View-component, however. This means that we do not know whether the MVVM breaches would have caused issues in that regard. In general, the program architecture may be weaker than it should have been.

Missing voucher

The program is at the moment not able to model a voucher. This will have some consequences for the statistics that the program can provide, namely that if an item is bought using a voucher the item will, in the system, be modeled as if it has been sold for cash and not for a voucher. This means that the statistics where this might have an effect, such as revenue generated by each category, will not be completely accurate.

The problem within the problem

The main problem that the manager at Danmission had was the problem of statistics, with regards to what kinds of products they sold. While trying to solve this problem we encountered several other problems, which we to some degree discussed in section 3.1, *Printable identification of products*. However, due to the uniqueness and uncertainty of what products/donations a charity shop receives, this problem was not solvable within this project's scope, meaning that our program does not handle physical identification labels, the consequences of which are also described in section 3.1. The way we handle this uncertainty is through abstraction, and that abstraction is exemplified by the *Category* class. This has

some advantages, which are that registration and unregistration of products becomes easier, which lessens the workload of employees, although the drawback is that products are not uniquely identifiable, which means that the requirement *Notify user about outdated products* becomes difficult to manage in practice. This abstraction also imposes another limit on our program's capacity to provide statistical detail to the user, although we know that our current level of detail *is* the level that the end user requested, this would be a challenge to improve upon if a higher level of detail is required in the future, without significantly increasing the menial labour of employees.

7.3 Conclusion

We will conclude the project, answer how much has been achieved, and determine if the goal of the project has been reached. To evaluate this we will take a look at the problem statement from section 1.8 and answer the question stated therein. The problem statement is the following:

The employees in the Danmission shop do not currently have any way to see which items that sell well and which ones do not. Furthermore, there is currently no dedicated system for keeping track of the amount of time a item has been part of the shop's selection. How can an IT-system help the employees keep statistics of what they sell the most and least of, and determine for how long they have had the specific items?

The answer to the problem statement consists, for our solution, in the creation of a system that consists of a client program, relying on a remote database, that can reflect the situation in the physical shop to an extent. This has resulted in a system where key objects such as items, categories and transactions are modeled and stored so that it becomes possible to generate and view a variety of statistics as well as determine for how long specific items have been for sale. As mentioned in the discussion in section 7.2 the system is not able to model a voucher which means that some types of statistics will not be completely representative unless vouchers are avoided.

The system has also been tested in the store by the manager. The feedback from the test was generally positive with only a few minor changes requested. The manager of the store was confident that if these changes were to be made, it would become possible to test the system in the charity shop for actual use.

We can from the above answer to the question from the problem statement conclude that the project has reached its goal, with a few shortcomings. Namely the lack of support for the use of vouchers in the system.

7.4 The Project in Retrospect

In this section we will reflect upon possible changes that we could have made to make the system more appealing to the target audience. Additionally, we will consider whether our choice of development method in section 1.9 benefited the project.

7.4.1 Webshop

Since the Danmission store advertises for their store through commercials or via Facebook, we could have put a greater emphasis on the interaction that our system might have with a website. Through this they could make their entire inventory easily accessible, reach a greater target audience, and perhaps even sell their products through the net.

7.4.2 Smartphone and Tablet Application

We had initially envisioned the system as an application well suited for smartphones or tables. This was meant to make it easier for the user to add products with pictures to the system, as such devices usually contain a camera. Also, it seemed very natural to unregister products from the system by scanning in their labels, just like cashiers in supermarkets do with barcode scanners.

In the end, as we turned our focus away from a web-shop, the pictures of products were suddenly a lot less important. More focus on a web-shop, however, as mentioned in the previous subsection, would make pictures important again, which in turn makes the system a lot more enticing and easy to use if it were compatible with smartphones or tablets.

7.4.3 The Development Method

The development method that we planned on using was largely the Waterfall method with Iterative qualities. This method has been very useful, because we had a clear overview over the different development phases throughout the project. It forced us to set a lot more clear requirements for the system, which may have been, through the use of the Iterative method, revised unnecessarily throughout the project.

However, more focus on the Iterative method, would have also forced us to communicate a lot more with our target audience. This was, at times, a problem, for instance during usability testing, as seen in section 6.6. Here we did not end up doing a validation test, only an assessment test, which means that we do not know whether our

revised program is better than prior to the test. In short, more iterations of the program and more communication with the target audience might have been an advantage, because in hindsight there was more uncertainty than we expected.

Bibliography

[1] *Forbrugere forvandler deres affald til genbrug*. https://politiken.dk/forbrugogliv/ECE2203421/forbrugere-forvandler-deres-affald-til-genbrug. 2014-07-02. (Visited on 09/21/2016).

[2] *Gode Råd*. http://www.loppemarked.nu/priser/. Unknown. (Visited on 09/21/2016).

[3] *Røde Kors*. https://www.rodekors.dk/det-goer-vi/genbrug. Unknown. (Visited on 09/21/2016).

[4] *PRESSEMEDDELELSE: Stort potentiale i genbrug af tekstiler*. https://dakofa.dk/element/stort-potentiale-i-genbrug-af-tekstiler/. 2014-12-09. (Visited on 09/21/2016).

[5] *90.000 ton tøj ryger årligt i skraldespanden*. http://www.dr.dk/nyheder/viden/miljoe/90000-ton-toej-ryger-aarligt-i-skraldespanden. 2015-04-08. (Visited on 09/21/2016).

[6] *Wikipedia*. https://da.wikipedia.org/wiki/Genbrugsbutik. 2015. (Visited on 09/21/2016).

[7] *GPS'en Sundsholmen - Brugte ting til gode formål*. http://aalborgforsyning.dk/renovation/borger/af-med-affaldet/genbrug-til-genbrugsbutik.aspx. Unknown. (Visited on 09/28/2016).

[8] *Projekt GenbrugNord*. http://www.aalborgforsyning.dk/media/197871/projektet_genbrugnord_er_etableret_i_et_samarbejde_mellem_familie.pdf. Unknown. (Visited on 10/04/2016).

[9] *Organisation*. http://english.danmission.dk/about/the-history/. Unknown. (Visited on 10/05/2016).

[10] David Benyon, Phil Turner, and Susan Turner. *Designing interactive systems: People, activities, contexts, technologies*. Pearson Education, 2005, pp. 156 –162.

[11] Lars Mathiassen et al. *Object-oriented analysis & design*. Marko, 2000. ISBN: 8777511530.

[12] *What is Waterfall model- advantages, disadvantages and when to use it?* http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/. (Accessed on 12/18/2016).

[13] Craig Larman. *Agile & Iterative Development*. 12th. Craig Larman & Pearson Education, inc, 2010, pp. 1–25. ISBN: 0-13-111155-8.

[14] *Danmission Bjerringbro*. http://danmissiongenbrugbjerringbro.dk/. (Visited on 10/07/2016).

[15] *Danmission Genbrugsland Aalborg*. http://genbrug.danmission.dk/blog/butik/danmission-genbrug-aalborg-genbrugsland/. (Visited on 10/07/2016).

[16] *Adult Vision: Over 60 Years of Age*. http://www.aoa.org/patients-and-public/good-vision-throughout-life/adult-vision-19-to-40-years-of-age/adult-vision-over-60-years-of-age?sso=y. (Visited on 11/09/2016).

[17] Lars Mathiassen et al. *Object-oriented analysis & design*. Marko, 2000, p. 24. ISBN: 8777511530.

[18] Lars Mathiassen et al. *Object-oriented analysis & design*. Marko, 2000, p. 22. ISBN: 8777511530.

[19] *Genbrugsindekset 2016*. https://guide.dba.dk/media/378494/dba_genbrugsindekset_2016.pdf. Page 12. Spring of 2016. (Visited on 09/12/2016).

[20] Lars Mathiassen et al. *Object-oriented analysis & design*. Marko, 2000, pp. 60–64. ISBN: 8777511530.

[21] Lars Mathiassen et al. *Object-oriented analysis & design*. Marko, 2000, p. 138. ISBN: 8777511530.

[22] Lars Mathiassen et al. *Object-oriented analysis & design*. Marko, 2000, pp. 154–155. ISBN: 8777511530.

[23] *MySQL database*. http://www.mysql.com/. (Visited on 10/10/2016).

[24] *db-engines.com*. http://db-engines.com/en/ranking. (Visited on 10/10/2016).

[25] *MySQL Connector .NET documentation*. http://dev.mysql.com/doc/connector-net/en/. (Visited on 10/10/2016).

[26] *What is WPF*. http://www.wpf-tutorial.com/about-wpf/what-is-wpf/. (Accessed on 11/26/2016).

[27] *WPF vs Winforms*. http://www.wpf-tutorial.com/about-wpf/wpf-vs-winforms/. (Accessed on 11/22/2016).

[28] *Modularity*. https://msdn.microsoft.com/en-us/library/ff921069(v=pandp.20).aspx. (Accessed on 11/18/2016).

[29] *Architectural Patterns and Styles*. https://msdn.microsoft.com/en-us/library/ee658117.aspx. (Accessed on 11/23/2016).

[30] *The MVVM Pattern*. https://msdn.microsoft.com/en-us/library/hh848246.aspx. (Accessed on 11/17/2016).

[31] *Working with App.xaml*. http://www.wpf-tutorial.com/wpf-application/working-with-app-xaml/. (Accessed on 11/23/2016).

[32] Nikolaj Mariager. *Object-Oriented Workshop 2 - DAT3/SW3*. https://www.moodle.aau.dk/pluginfile.php/877499/mod_folder/content/0/WPF_Nikolaj_Mariager.pptx. The file is not available for download unless one has an AAU logon (Accessed on 11/23/2016).

[33] Jakob Nielsen. *How Many Test Users in a Usability Study?* https://www.nngroup.com/articles/how-many-test-users//. (Accessed on 12/15/2016). June 2012.

[34] Jeffrey Rubin. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1994, pp. 28–36. ISBN: 0471594032, 9780471594031.

[35] *Reklamationsret - forbrug.dk*. http://www.forbrug.dk/Artikler/Test-og-raad/Forbrugerleksikon/mangelsbefoejelser?tc=F8AD094C7E3F4E469700CC3A7C4BAA2B?SC_itemid=f8ad094c-7e3f-4e46-9700-cc3a7c4baa2b. (Visited on 09/28/2016).

[36] *Retsinformation*. https://www.retsinformation.dk/Forms/R0710.aspx?id=142961. 2014. (Visited on 10/05/2016).

[37] *Er en garanti og en reklamationsret det samme? - forbrug.dk*. http://www.forbrug.dk/Hotlinen/garantiogreklamationsret?tc=AD677AB99D35428489BBBCD0083B71A1. (Visited on 09/28/2016).

[38] *Køb brugt: Her er dine rettigheder | Forbrugerrådet Tænk*. https://taenk.dk/raadgivning-og-rettigheder/koeb-brugt-her-er-dine-rettigheder. (Visited on 09/28/2016).

Appendices

A Sales Law

In this chapter we will cover how warranty and consumer rights are handled legally in Denmark and how this affects thrift stores. This may in turn have effects upon the further design of the final product.

When a consumer buys a product that turns out to be defect or inadequate compared to the promised product, then the consumer's rights to warranty is protected by the Sales Law. The Sales Law dictates that the consumer has two years of warranty, which can neither be negotiated nor decreased. As a general guideline, while this warranty holds, the consumer has the right to choose one of the following options if the product turns out to be of poor quality:

- The product must be repaired
- The product must be exchanged for a new one
- The price must be decreased
- The contract must be rescinded

The seller has the right to decline the choice of option if the execution of that option is disproportionally expensive compared to the other options. Additionally the seller can decline the choice of option if the product in question is no longer being produced, or if the inadequate product has already been worn out. [35] The specific laws descried here can be found in paragraphs 42, 43, 49 and 54 of the Danish Sales Law [36].

A.1 Additional Warranty Offered by the Seller

Sellers may offer consumers additional warranty, that covers more time or situations, where the seller might not usually be forced to help the consumer. The difference between a warranty offered by the seller and the warranty, that is defined by the Sales Law, is that the seller-offered warranty is not part of the law, and is just an additional commitment that the seller has chosen to offer the consumer. From this point onwards, when we mention warranty, we will be referring to the warranty, that is provided by the

Sales Law. If we wish to mention the additional warranty that a seller may commit to, then we will explicitly state this. [37]

A.2 Warranty and Second-hand Goods

The warranty rights apply to second-hand goods as well. If a consumer buys a product from a secondhand store, he will be covered by the Sales Law warranty of two years. As always the quality of the product by the time of purchase will be kept in mind if the consumer wishes to chose one of his four options. In case a consumer buys a product off another consumer, it is important to note that the warranty does not begin anew. [38]

B Mail Questionnaire

B.1 Questionnaire: Translated

Thrift shops and computer software for managing inventory
Section 1:
Are you currently utilizing computer software to manage your inventory? (Yes/No)

Section 2:
What computer software is used and how does it aid the work?

Section 3:
Do you manually manage your inventory in writing? (Yes/No)

Section 4:
How are you managing your inventory?

Section 5:
Would you be interested in computer software that could improve your work? (Yes/No)

Section 6:
Why not?

Section 7:
Would you be interested in computer software that could improve your work? (Yes/No)

Section 8:
Are you displaying your inventory on the Internet? (Yes/No)

Section 9:
Are you interested in displaying your inventory on the Internet? (Yes/No)

Section 10:
Would you be interested in computer software that couples inventory management together with displaying the inventory on the Internet? (Yes/No)

Section 11:
Do you currently use computer software where it is easy to add and remove items in the inventory? (Yes/No)

Section 12:
Would you like to use computer software to do this? (Yes/No)

Section 13:
Are there any difficulties regarding the use of computer software in the day to day? (Yes/No)

Section 14:
Please describe these difficulties.

Section 15:
Would you be interested in further collaboration regarding the development of computer software? (Yes/No)

Section 16:
Whom should we contact? (Please include phone number or e-mail)

Section 17:
Thanks for answering.

B.2 Questionnaire: Original

Genbrugsbutikker og IT-systemer til inventarstyring

*Required

1. **Bruger I aktuelt IT-systemer til administration af jeres inventar?** *
Mark only one oval.
Ja *Skip to question 2.*
Nej *Skip to question 3.*

Hvilke IT-systemer bruger I til dette, og hvordan hjælper disse i arbejdet?

2.

Skip to question 13.

Holder I skriftligt styr på varene i butikken?

3. *
Mark only one oval.
Ja *Skip to question 4.*
Nej *Skip to question 6.*

Hvordan holder I styr på jeres inventar?

4.

Skip to question 5.

Ville I være interesserede i et IT-system til forbedring af processen?

5. *
Mark only one oval.
Ja *Skip to question 8.*
Nej *Skip to "Mange tak for svar."*

Hvorfor ikke?

6.

Skip to question 7.

Ville I være interesserede i et IT-system til denne process?

7. *
Mark only one oval.
Ja *Skip to question 8.*
Nej *Skip to "Mange tak for svar."*

Viser I jeres inventar på en hjemmeside?

8. *
Mark only one oval.
Ja *Skip to question 11.*
Nej *Skip to question 9.*

Er I interesserede i at vise jeres inventar på en hjemmeside?

9. *
Mark only one oval.
Ja *Skip to question 10.*
Nej *Skip to question 15.*

Ville I derudover være interesserede i et system, der kobler lagerstyring sammen med fremvisning af varerne på hjemmesiden?

10. *
Mark only one oval.
Ja *Skip to question 15.*
Nej *Skip to question 15.*

Skip to question 6.

Har I et effektivt system til at administrere tilføjelse og sletning af varer på denne?

11. *
Mark only one oval.
Ja *Skip to question 15.*
Nej *Skip to question 12.*

Ville I kunne gøre brug af et IT-system til dette?

12. *
Mark only one oval.
Ja *Skip to question 15.*
Nej *Skip to question 15.*

Skip to question 15.

Opstår der besværligheder ved brug af IT-systemerne i arbejdet?

13. *
Mark only one oval.
Ja *Skip to question 14.*
Nej *Skip to question 8.*

Skip to question 14.

Beskriv venligst disse besværligheder.

14.

Skip to question 8.

Er I interesserede i yderligere samarbejde vedr. udviklingen af et IT-system?

15. *
Mark only one oval.
Ja *Skip to question 16.*
Nej *Skip to "Mange tak for svar."*

Hvem skal vi kontakte?

Tilføj venligst tlf. nr., emailadresse eller lign.

16. **Kontaktinformation:**

Skip to "Mange tak for svar."

Mange tak for svar

B.3 Questionnaire Results: Original

Google Forms does not allow proper PDF printing of the result summary, which is what has caused some of the formatting to be off.

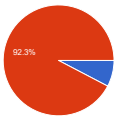
Edit this form

13 responses

Publish analytics

Summary

Bruger I aktuelt IT-systemer til administration af jeres inventar?



Ja	1	7.7%
Nej	12	92.3%

Hvilke IT-systemer bruger I til dette, og hvordan hjælper disse i arbejdet?

Intern, Danmission

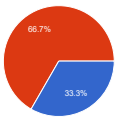
Holder I skriftligt styr på varene i butikken?

Ja	0	0%
Nej	12	100%

Hvordan holder I styr på jeres inventar?

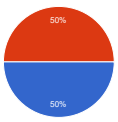
No responses yet for this question.

Viser I jeres inventar på en hjemmeside?



Ja	1	33.3%
Nej	2	66.7%

Er I interesserede i at vise jeres inventar på en hjemmeside?



Ja	1	50%
Nej	1	50%

Ville I derudover være interesserede i et system, der kobler lagerstyring sammen med fremvisning af varerne på hjemmesiden?

Ja	1	100%
Nej	0	0%

Har I et effektivt system til at administrere tilføjelse og sletning af varer på denne?

Ja	0	0%
Nej	1	100%

Ville I kunne gøre brug af et IT-system til dette?

Ja	1	100%
Nej	0	0%

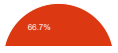
Opstår der besværligheder ved brug af IT-systemerne i arbejdet?

Ja	0	0%
Nej	1	100%

Beskriv venligst disse besværligheder.

No responses yet for this question.

Er I interesserede i yderligere samarbejde vedr. udviklingen af et IT-system?



Ja	1	33.3%
Nej	2	66.7%

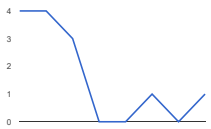
Hvem skal vi kontakte?

Kontaktinformation:

Real contact information blurred out.

Mange tak for svar

Number of daily responses



Ville I være interesserede i et IT-system til forbedring af processen?

No responses yet for this question.

Hvorfor ikke?

vi er en lille butik hvor det er let overskuelig

fordi det ikke er muligt på forhånd at vide hvilke varer vi har i butikken

Bestyrelsen i butikken vil gerne anvende kasse rapparatet til at dokumentere hvilke produkter der er mest omsætning i, men da vores butikspersonale udelukkende består af pensionister uden it kundskaber, må vi erkende at yderligere brug af it-systemer, som butikspersonalet skal anvende, er en plan for fremtiden.

der er ikke 2 varer der er identiske, så arbejdet vil være for tidskrævende i forhold til salgspris og hermed indtjening

Det er umuligt, da vi er mange forskellige, som arbejder der. Det vil være alt for tidskrævende

Varerne er typisk unikke med kun EN af hver og samtidig mangel på kompetancer

Synes, det er uoverskueligt at registrere. Og måske også unødvendigt.

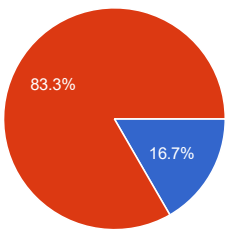
Føler det er unødvendigt dog sætter vi dato på møbler der indsættes i buyikken

Fordi det vil være fuldstændig uden mening at holde styr på den varmemængde vi har og hvor en stor del ryger ud fordi der ikke er købere til dem. det vil efter min mening være fuldstændig spild af tid.

vil tage for lang tid

Vi ønsker hurtig omsætningshastighed; samme vare kommer aldrig igen; vi anvender ikke stregkoder og det er frivillige med meget forskellige kompetencer, som styrer butikkerne.

Ville I være interesserede i et IT-system til denne process?



Ja	**2**	16.7%
Nej	**10**	83.3%

C Mail Questionnaire and Results in English

D Mockup Of Prototype

This section will cover the rest of the mockups that was introduced in section 3.2.2 and used to illustrate the GUI for the prototype. The following mockups are going to illustrate the rest of the prototype's GUI. The mockups are divided into three subsections which are the three overall functionality the prototype has.

D.1 Register product

Figure 2: This is the main page on how to register a product looks like.

On figure 2 it is possible to pick a category, a price, date added, an ID and take a picture of the product. The following figure 3 will feature how to choose a category.

Figure 3: How to choose the category of the product.

Clicking on the category list you can choose between the featured parent categories. After choosing one, subcategories are visible to choose from. The following two figures 4 and 5 will feature how to edit the price of a product.

Figure 4: This features the products standard price with no modifications.

Figure 4 features the category, the price, that can be modified, the date and the ID of the product. The following figure 5 will show how to edit the price of the product with the two arrows shown on the figure 4.

Figure 5: How the price is edited through using the two arrows.

Figure 5 features how the price of a product can be edited. It is done through the two arrows shown in the figure. The following figure 6 send a notification to user that the product has been successfully added.

Figure 6: Notification that a product has successfully been added.

D.2 Remove Product

Figure 7: Search engine for ID or via QR Code.

Figure 7 features how to remove a product through the use of either a products ID or QR Code.

Figure 8: Removing through the use of QR Code.

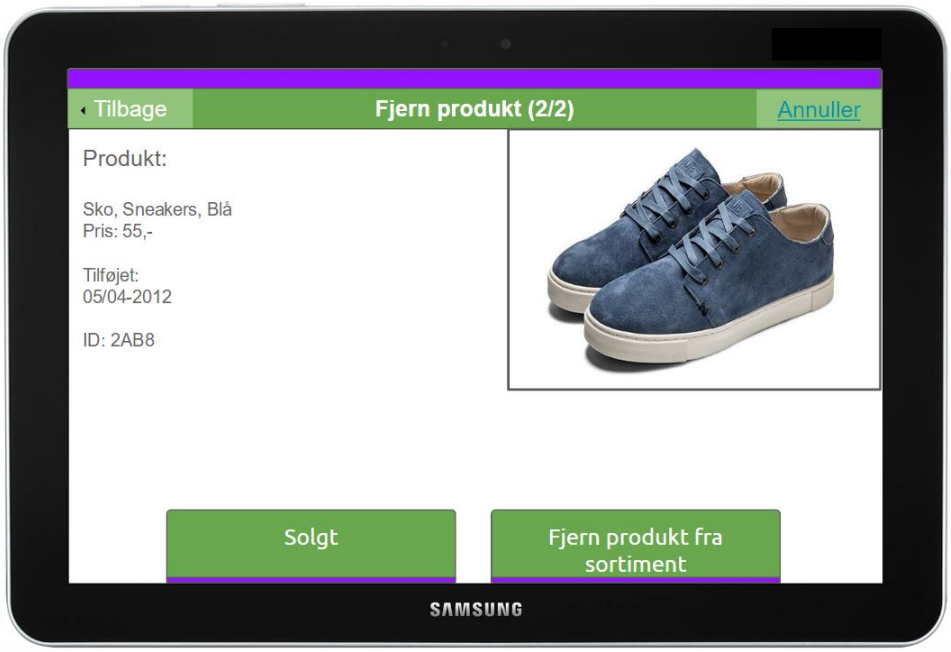


Figure 9: Description of the product with a remove product and sell product button.

Figure 9 features how to remove a product after finding it, either ID or QR Code, as sold or removed.

D.3 Find Product

Figure 10: Find product either by ID, description or QR Code.

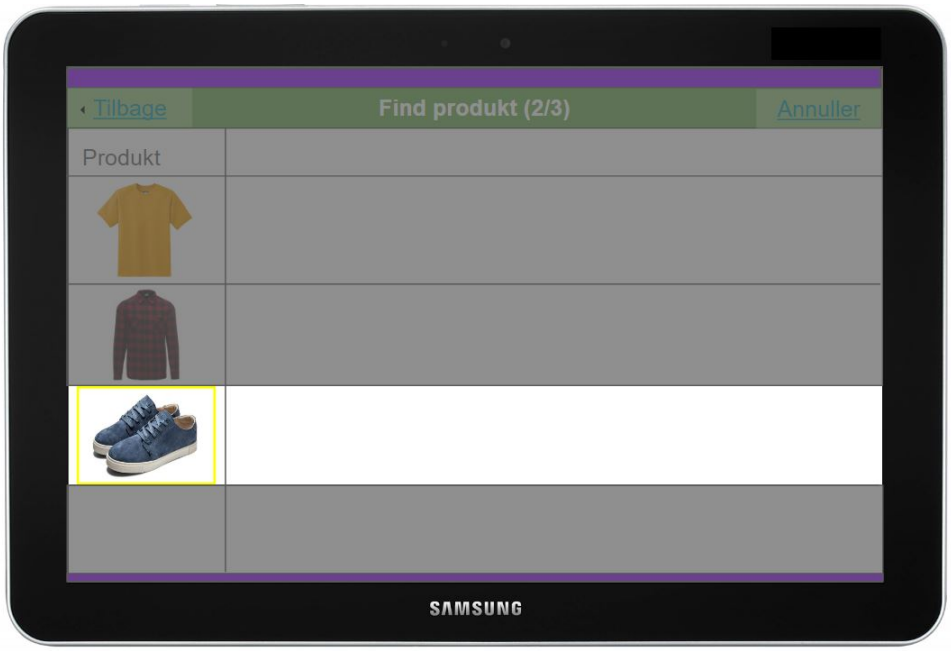


Figure 11: Picture chosen depending on the information given in figure 10.

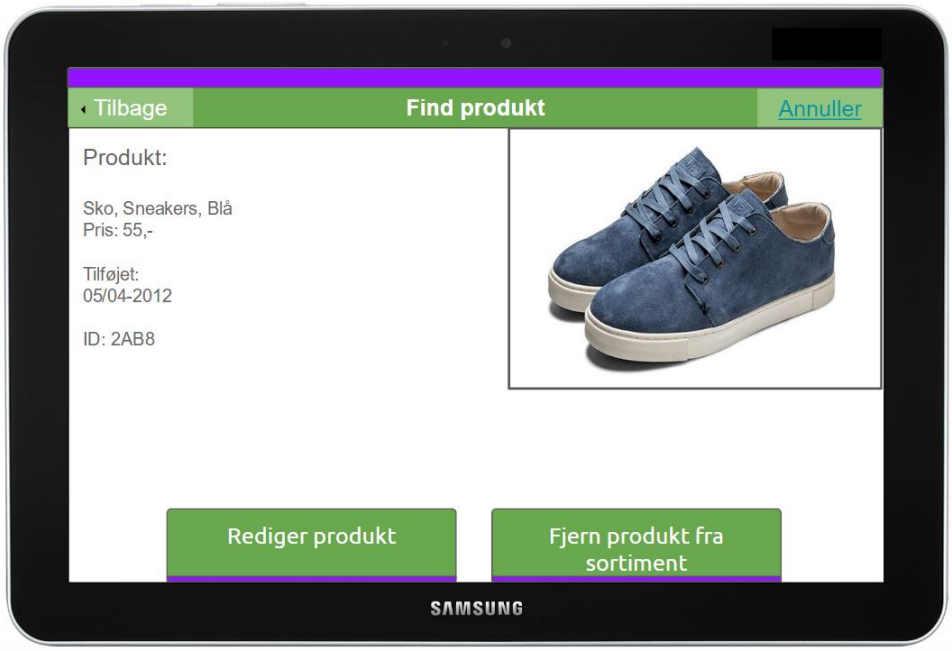


Figure 12: Product information with two buttons to edit or remove the product.

E Summary of Interview Notes

E.1 Varer

Alle varer i butikken er donationer, fra diverse kilder. Fx privat personer eller dødsboer.

Hvis varerne ikke bliver solgt i butikken, bliver de sendt videre i organisationen. Først til butikken i Svenstrup, og herefter til vejle, hvor varerne sorteres. Herefter bliver de sendt til Afrika.

Alt personale i butikken prissætter varer. Dette sker ved vejledende priser på varegrupper. Dog hænder det at butikken får mærkevarer, som skal prissættes højere. Ofte af en ekstern vurderingsperson. Butikken har sedler hængt op til personalet, hvorpå visse mærker står, som skal have en højere pris. Fx Lego, George Jensen, Herstal osv.

Butikken yder ikke en ”penge tilbage”-service, da det ikke er muligt med deres varer og struktur. Kunder som ikke er tilfredse med deres køb kan dog få et tilgodebevis.

Alle varer sælges som beset. Dvs. Vare sælges i den stand som de står med i butikken, ved køb. Butikken gør dog opmærksom på evt. Fejl eller mangler ved et produkt.

Alle varer bærer et individuelt prismærke, som er håndskrevet af personalet.

Varerne i butikken fyldes op, eftersom det mangles. Der findes et baglokale i butikken med et lille lager af usorterede eller ikke-prissatte. Varer som ikke bliver solgt efter en længere periode (omtrent 3 måneder) bliver afskaffet, da de højst sandsynligt ikke bliver solgt. Det er vigtigt at have rotation i butikkens varer, da der er mange stamkunder.

Nogle kunder går meget op i varernes historie. Der er også mange af de folk som kommer og donerer varer som går meget op i at fortælle historierne bag varerene. Derfor gør butikken, af og til, noget ud af at skrive disse historier ned, for senere at kunne fortælle om det. Dette er dog sjældent, da det hurtigt bliver uoverskueligt med de mange frivillige og de mange varer. Hvis det er mærkevarer, kan personalet også læse om det, hvis de har interesse for det.

Butikken modtager af og til kopi-varer. Fx modtog de forleden et Rolex ur, som så åbenlyst viste sig at være en kopi-vare. Dog modtager butikken ofte dyre smykker, som guld, rav eller ure, fra folk som enten ingen idé eller interesse har om det.

Størstedelen af donationer består af tøj (primært dame- og børnetøj).

Butikken benytter farvekoder på nogle varer.

E.2 IT

I butikken er det udelukkende Lone som benytter nogen form for IT. Hun [Lone] er ansvarlig for at kommunikere opad i organisationen via telefon/mail. Grundet aldersgruppen af personalet, er det en kæmpe udfordring at indføre IT i butikken. De ældre kan ikke se meningen med IT-systemer, og flere af dem er bange for at bruge det. Flere frivillige har "truet" med at sige op, når nye IT initiativer er blevet præsenteret for dem.

Butikken har haft gode erfaringer med at benytte Facebook til at promovere deres inventar. Flere gange er specielle varer eller udsalg blevet reklameret på Facebook, med relativ stor succes. Butikken ønsker at benytte Facebook i større grad, for at nå et bredere marked. Derudover ønsker de at promovere varer med billeder, pris og beskrivelser af varer.

Dog er personalet ikke på Facebook, og dette er en problematik for deres forståelse for platformen. Lige nu udføres arbejde med pen og papir, hvorefter det indskrives i en Excel formel senere. Det kunne være godt hvis man kunne digitalisere hverdagen i butikken. Dvs. Registrere salg af varer digitalt.

Danmissions butikker deler en intern online platform. På denne platform kan butikkerne kommunikere med hinanden, over et online forum. Her deler butikkerne erfaringer og viden omkring varer eller butikken. Fx hjælp til prissætning af varer. Derved kan butikker konsultere hinanden, hvis nødvendigt.

E.3 Butikken

De daglige opgaver består af sortering og prissætning af varer i butikken. Derudover er betjening af kunder også en væsentlig del af hverdagen i butikken. Der bliver indleveret varer, hvorefter de bliver sorteret efter behov. I visse tilfælde gøres varerne rene, hvis det menes at kunne betale sig, før de bliver hængt ud i butikken.

Butikken på J. F. Kennedys Plads, har tidligere haft salg af møbler. Det blev dog besluttet af stoppe salget af møbler, da det ikke kunne betale sig i forhold til pladsen og udgifterne associeret derved.

Butikkens hoveddel af kunder er faste kunder. Det skyndes at butikken har 100-200 faste stamkunder som regelmæssigt vender tilbage i butikken. Disse kunder kommer oftest ud på eftermiddagen.

De 3 butikker i Aalborg området, deler samme bil. Dette er dog en uofficiel bil, som kun sjældent bliver brugt.

Butikken har ca. To store udsalg om året, hvor det meste af butikken er -50%.

Butikkens omsætning i kontanter bliver sat i banken periodisk. Dvs. Butikken til tider ligger inde med større beløb, som fysisk skal tages i banken.

Hver butik har deres egen interne struktur. Men hver butik skal have en formand, daglig leder samt en kassere. Derudover bruger Danmission en revisor.

Butikken har netop lige fået Internetforbindelse, i form af Wi-Fi. Butikken fik først kortterminal for 2 år siden. Men kassen (det kontante system) og kortterminallen er fuldstændigt separeret. Dette er en problemstilling for bogføringen, da de to betalingssystemer er separate. Der er mange kvitteringer, idet hvert køb har deres egen dokumentering, som senere skal skrives ned. Dette resulterer i dobbeltarbejde. Butikken benytter et såkaldt "Tæller" system til kortterminalen.

De primære daglige problemstillinger består i det decentraliserede system, samt det faktum at størstedelen af de frivillige ikke er venligtsindede overfor IT.

Eftersom butikken ikke længere modtager møbler, bliver disse donationsanmodninger henvist til andre af Danmissions butikker. Hvis ikke Danmission kan tage imod disse, bliver de oftest smidt ud.

Danmission foretrækker ikke selv at skulle smide inventar på lossepladsen, da de skal betale en erhvervsafgift på deres affald.

Butikkens varer eller overskud bliver ofte doneret til lokalmiljøet i Aalborg.

Butikken består af forskellige afdelinger. Fx En afdeling til tøj; en til børneting; en til LP'er og bøger, osv.

Butikken har haft Wi-Fi siden sidste fredag (noteret d. 30-09-2016).

Skilte og mærker i butikken bliver printet i butikken, af lederen (den eneste som i forvejen har noget med IT at gøre). Mange af disse skilte skal bruges mange gange. Derfor har Lone lavet skabeloner og kopier af designs, som let og hurtigt kan printes.

Den centrale del af organisationen (baseret i København), ved ikke specifikt hvordan hverdagen fungerer ude i butikkerne. Derfor har hver butik deres eget miljø, og måder at gøre visse ting på. Derfor kan priser variere meget, alt efter om butikken ligger i

Nordjylland eller i København.

Butikkerne modtager ikke finansiel hjælp fra Danmission. Derfor skal hver butik tjene penge nok til at betale dets husleje og andre udgifter. Danmission på J. F. Kennedys plads' månedlige husleje er ca. 30.000 kr. Dertil kommer diverse driftsomkostninger (dog ingen løn), som også skal finansieres af butikken selv. Danmission finansierer et arrangement for personalet én gang om året.

Butikken får tit folk i jobtræning, eller flygtninge. Dette kan give problemer, da folk i arbejdstræning kan have problemer med at arbejde selvstændigt. Sprogbarrieren mellem personalet og flygtninge er også en stor problemstilling, som gør kommunikation svært. De flygtninge som butikken får tildelt, er ofte enormt overkvalificerede. Men de kan ikke indgå på en dansk arbejdsplads, så de får sprogtræning ved at arbejde i butikken.

E.4 Personalet

Personalet består udelukkende af frivillige. Danmission har omkring 8000 frivillige fordelt i organisationen. Disse frivillige er ikke betalte. Dog finansierer organisationen arrangementer for personalet. Danmission har svært ved at finde unge frivillige, da disse oftest søger et betalt job. De oplever dog opmærksomhed fra unge.

Personalet vil gerne udfører forskellige arbejdsopgaver, men de har dog en fortrukket opgave i butikken. Derfor er det oftest fx de samme personer som står ved kassen. Aldersgruppen for personalet er omkring 50-80 år. 4-5 personer i butikken er under 50. Ellers er hovedparten af personalet over pensionsalderen.

Butikken har én leder. Dvs. En som står for den daglige drift, i form af arbejdsplaner, ansættelser, osv. Butikken har herunder et hierarki, som man også kunne finde på en traditionel arbejdsplads. Lone beskriver butikken som en "mini-arbejdsplads". Butikken får også besøg af arbejdstilsynet, idet de er klassificeret som erhverv.

Butikkens leder, har til ansvar at rapportere regnskab hver måned. Dog må dette gerne ske oftere. Dette bliver, via en Excel form, sent afsted til Danmission.

Der findes en personliste over alle de frivillige, med deres navn, tlf.nr. og e-mail. Der benyttes dog mest telefoner til intern kommunikation mellem personalet, idet de fleste enten ikke har, eller benytter sig af e-mail.

Noget af personalet har deres egne eksterne kontakter som bliver brugt til vurdering af ting, som ikke umildbart kan vurderes af butikkens eget personale. Fx guld og ure.

Det kan ofte være et problem at visse kundesegmenter har en tendens til at ville 'prutte om prisen'. Dette kan det ældre personale godt blive sure over, idet varerne forvejen er meget billige. Dette er en kulturel problemstilling, som kolliderer med kulturen i butikken.

Ansatte har selv frokost med. Dog betaler butikken for diverse kage, kaffe, osv.

F Candidate Lists of Events and Classes

Employee added to contact list	Employee removed from contact list	Product gift wrapped	Employee arrived	Employee left
Employee hired	Employee fired	Employee resigned	Daily sales accounted	Monthly sales reported
Product label created	Product label removed	Work schedule created	Work schedule edited	Supplies purchased
Customer receipt printed	Product delivered	Product discarded	Product added	Product removed
Product returned	Product sold	Product received	Profit sent	Statistics accessed
Employees notified	Notification added to bulletin board	Item added to display case	Item removed from display case	Store section created
Store section closed	Auctioneer priced product	External value-setter consulted	Fitting room used	Cash register used
Credit card terminal used				

Table 1: Overview of the candidate list of events from the brainstorming session

Manager	Cashier	Employee	Organization	Product
Charity shop	Regional manager	Transaction	Product label	Work schedule
Cash register + terminal	External value setter	Laptop	Employee contact sheet	Auctioneer
Lists of high-value brands	Standard product prices	Standard category signs (physical)	Storage room	Fitting room
Router	Bulletin board	Valuables display case	Store section (categories)	Transport car
Delivery person	Expenditure sheet			

Table 2: Overview of the candidate list of classes from the brainstorming session

G Interview Questions and Assignments

Assignment 1 - Registering a product

The participant will be given an item, which the participant has to register into the system. Taking a picture of the item is not necessary, considering that the assignments will be done using a laptop.

Assignment text:
You are running out of items in the store and need to put up some more products for sale. You would like to register the following product (participant is then given an item).

Assignment 2 - Finding a product

The participant will be asked to find an item based on some data. This data includes the name, id, and category of the item in question. The participant will have to use the search bar, and choose only one of the mentioned attributes to use as a search parameter. The assignment will be complete when the participant has

selected the correct item and changed its' price, and that the new price has been updated in the database.

Assignment text:
You will now be given a note with some identifying attributes of a product. Use one of these attributes to find the product in the system. Then change the price of the product to something of your choosing.

Assignment 3 - Selling a product

The participant will be given a physical list of id's that correspond to active products. The participant will have to find the products, add them to the system's basket and then confirm that the transaction succeeded.

Assignment text:
You will be given a list of products, which a customer wants to buy. Use the system to sell these products.

Assignment 4 - Find all the outdated products

Assignment text:
you find all the products that are currently outdated - and then remove one of them from the system.

Assignment 5 - Statistics and Transactions

For this assignment the participant will be asked to provide some information about the stores' sales and transactions. Currently the system has some generated data, which we will use to calculate these statistics. The transactions can be found in the inventory overview tab.

Assignment text:
For this assignment you need to find out how much money the store made from selling male clothing from 12./1. to 12./12. Afterwards find the most profitable transaction.

G.1 Questions about Design

Question 1

Was it easy to access the different functions in the program through the side-menu?

Question 2

Did the icons help with identifying the different functions?

Question 3

Did the user interface respond in the way you expected?

Question 4

Did you like the way the different functions were presented, like the register product page etc.?

Question 5

What would you like to be different in the design of the program, page layout etc.?

G.2 Questions about Functionality

Question 1

Does the program have enough functionality to work in the store in practice?

Question 2

Is there any present functionality that could be made better or implemented in a different way to suit your needs?

Question 3

What would you like to be different in the functionality of the program, more statistics, more product options etc.?