Business Software Engineering Processes: An Analytics Case Study

GEORGE E. KALLERGIS
Business Software Engineering Processes
An Analytics Case Study

Master of Science Thesis

Written by
George E. Kallergis

Supervised by
Ove Svenneke, Senior Manager, Customer Value Sweden AB

Examined by
Prof. Mark T. Smith

Software Engineering of Distributed Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
August 2015, Stockholm, Sweden
Abstract

Using Information Technology (IT) solutions to automate business processes has been the norm for many organizations in recent years. Despite the great benefits an enterprise can reap from adopting such solutions, developing ones own IT solution is not as easy as it might seem. If we take into account the ever increasing complexity of modern businesses and their operating environment as well as the fast pace with which the modern world is changing, development and maintenance of such systems can easily become a daunting task. Many software development processes have been proposed over the years aiming at increasing software projects’ success rates in terms of budget, time and requirements satisfaction. In this project (code named Helium), we propose a simple software development process customised specifically for Customer Value (CV) and we apply this process to develop a novel distributed IT system that automates the main business processes of the company. Our main goal is to reduce the operational costs of Customer Value (CV) and increase its capacity and provided Quality of Service (QoS). The ”start-up” nature of the company is taken under consideration, since it introduces a considerable amount of uncertainty, as well as the fact that the initial set of projects in their project road-map are going to be thesis projects carried out by students and not experienced professionals. The proposed distributed architecture aims at providing maintenance, expansion, performance and scalability benefits. A basic set of measurements, carried out on the implemented system, validate the correctness of our approach with respect to performance and a set of interviews carried out with senior developers and managers validate the importance of the benefits of the process and architecture from a business standpoint.
Dedication

This work is dedicated to my family and friends.
I would like to get the opportunity to express my gratitude towards all that helped in bringing this project to life. I really appreciate the invaluable insights, guidance and constructive criticism I received during the execution of this project and I consider them to be of utmost importance for the final results.

I express my kindest regards to Prof. Mark T. Smith for overseeing this project as the examiner and the Customer Value (CV) management team, Mr. Ove Svenneke, Mr. Per Weidenman and Mr. Gunnar Flossing, for giving me the chance to be part of what turned out to be the greatest learning experience of my life, so far. I would like to especially thank Mr. Ove Svenneke, co-founder and senior manager at Customer Value, for providing all the necessary details needed at the early stages of the project and for providing valuable feedback and guidance throughout its lifecycle.

I would like to thank my classmate and friend, George Paralikidis, Yanzi Networks, Software Engineer for his help in clarifying different aspects of the project and for the good job he did in the previous version of the system which served as the basis for Customer Value System (CVS). He literally saved me a lot of time understanding the requirements and it would have been impossible to do it otherwise considering the remote-work model adopted for the execution of this project. I would also like to thank my classmate and friend Theofilos Kakantousis for reviewing this text with me and providing valuable feedback.

Furthermore, I would like to thank Mr. Stefanos Kakarantzas, Tickethour Group, CEO for providing valuable man-hours of the Tickethour IT team to review and evaluate this project’s results. More specifically, I would like to thank Mr. Triantafyllos Gavrilas, Mr. Michael Michailidis, Mr. Emmanouil Motakis, Mr. George Triantafilidis, Mr. Michael Moudatsos, Mr. George Poullos, Mr. Eystratios Keramidas and Mr. Nomikos Mouzourakis for taking part in the interviews held towards the end of this project.

Last but not least, I would like to thank my friends and family for providing me with all the social aspects of life that are absolutely necessary to strike a balance between working and living.

Thank you all,
George E. Kallergis
Contents

List of Figures ix
List of Tables xi

1 Introduction 1
   1.1 Problem Description ........................................... 2
   1.2 Problem Statement & Purpose .................................... 3
   1.3 Goals & Benefits .................................................. 3
   1.4 Thesis Outline .................................................... 5

2 System Overview 7
   2.1 Main Features ..................................................... 7
      2.1.1 Company & Employee Profile Management ..................... 7
      2.1.2 Product Purchasing ......................................... 8
      2.1.3 Report Generation ......................................... 8
      2.1.4 Business Intel & Miscellaneous Features ................... 9

3 Methodology 11
   3.1 Philosophical Assumption ....................................... 11
   3.2 Method & Research Approach ..................................... 11
   3.3 Data Collection & Analysis ...................................... 11

4 Analysis & Design 13
   4.1 Software Development Process Model ............................ 13
      4.1.1 The Requirements Phase .................................. 16
      4.1.2 The Modeling Phase ....................................... 17
      4.1.3 The Construction Phase ................................... 17
      4.1.4 The Deployment Phase .................................... 18
   4.2 Process Application .............................................. 18
   4.3 Problem Analysis ................................................ 19
      4.3.1 Business Analysis ......................................... 19
      4.3.2 System Analysis ........................................... 27
   4.4 Problem Solution ................................................ 34
      4.4.1 System Design ............................................. 34
      4.4.2 System Integration Schemes ................................. 43

5 Implementation 47
   5.1 Software Stack .................................................. 47
      5.1.1 CVS SPA Software Stack ................................... 47
## Contents

5.1.2 CVS Core & Genny Software Stack ................................ 49  
5.2 Statistical Process Implementation ................................. 50  
5.3 System Deployment Schemes ......................................... 51  
  5.3.1 Development Deployment Scheme ............................... 51  
  5.3.2 Production Deployment Scheme ................................. 53  

6 Evaluation 55  
  6.1 Experimental Testbed .............................................. 55  
  6.2 Performance Profiling ............................................. 55  
  6.3 Use Case Testing .................................................. 58  
  6.4 Evaluation Interviews ............................................. 58  

7 Conclusions 61  
  7.1 Conclusion .......................................................... 61  
  7.2 Future Work ........................................................ 62  
    7.2.1 Leftovers ..................................................... 62  
    7.2.2 Formal Documentation of Statistical Processes ............ 62  
    7.2.3 Development Infrastructure ................................. 63  
    7.2.4 MySQL Database Sharding ................................... 63  
    7.2.5 Evaluate Different Database Technologies .................. 64  
    7.2.6 File Management ............................................. 64  
    7.2.7 Queuing for Generation Services ............................ 64  

A Full System Impact Analysis - Account Registration 67  
B NewBiz Report FRD 69  
C Predictive Report FRD 113  
D Sample Transaction Set 141  
E Interview Notes 143  
F NewBiz Report - Generated Sample 153  
G Enlarged Model Diagrams 163  
H Technical Background 167  
  H.1 Software Development Processes ............................... 167  
  H.2 Modeling & UML .................................................. 168  
  H.3 OOAD & Microservices Architecture ............................ 169  

Acronyms 177  
Glossary 179
## List of Figures

4.1 Top level project management process model. ........................................ 13  
4.2 The Helium software development process model. ................................. 14  
4.3 The Helium project burn-down rate chart. .............................................. 15  
4.4 Requirement status color coding scheme. .............................................. 16  
4.5 Manual process executed by CV to deliver its products. ......................... 20  
4.6 Conceptual level business domain model. ............................................ 21  
4.7 Requirements model, high level categories diagram. ............................. 22  
4.8 Requirements model, business intel requirements diagram. .................... 22  
4.9 Requirements model, miscellaneous requirements diagram. .................... 22  
4.10 Requirements model, company hierarchy management requirements diagram. ........................................ 23  
4.11 Requirements model, account management requirements diagram .......... 24  
4.12 Requirements model, product management requirements diagram .......... 25  
4.13 Requirements model, report generation requirements diagram ............... 26  
4.14 Non-functional requirements diagrams. .............................................. 26  
4.15 Primary actors identified during system analysis. ................................ 28  
4.16 Secondary actors identified during system analysis. ............................ 29  
4.17 Product purchasing use-cases. ......................................................... 30  
4.18 Activity diagram of the Log Into User Account use-case. ...................... 31  
4.19 User interfaces diagram for the account management category. ............ 32  
4.20 Business intel impact analysis diagram. ............................................ 33  
4.21 Customer Value System (CVS) Data model. See Appendix G for an enlarged version of this model ........................................ 35  
4.22 CVA SPA high level architecture. ..................................................... 36  
4.23 CVA SPA components impact diagram. .............................................. 38  
4.24 CVS Core & Genny high level architecture. ........................................ 39  
4.25 High level architecture, presentation layer. ........................................ 40  
4.26 High level architecture, business logic layer. ..................................... 40  
4.27 Business logic package, services components. .................................... 41  
4.28 High level architecture, abstraction layer. ......................................... 42  
4.29 Abstraction layer, external service abstraction. .................................... 43  
4.30 High level architecture, dependency layer. ........................................ 44  
4.31 Integration approaches, service exposure scheme. .............................. 44  
4.32 Integration approaches, shared storage scheme. .................................. 45  
5.1 CVS SPA Software Stack. .................................................................... 48  
5.2 CVS Core and Genny Software Stack. ................................................ 49  
5.3 CVS Core and Genny data access. ...................................................... 50
<table>
<thead>
<tr>
<th>Number</th>
<th>Figure Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>NewBiz report generation process as implemented in [8].</td>
<td>50</td>
</tr>
<tr>
<td>5.5</td>
<td>Deployment schemes, development deployment.</td>
<td>52</td>
</tr>
<tr>
<td>5.6</td>
<td>Deployment schemes, production deployment.</td>
<td>53</td>
</tr>
<tr>
<td>6.1</td>
<td>NewBiz report generation times.</td>
<td>56</td>
</tr>
<tr>
<td>6.2</td>
<td>Analysis time for 10k transaction relative to database table size.</td>
<td>57</td>
</tr>
<tr>
<td>6.3</td>
<td>Mean generation time for simultaneous report generations.</td>
<td>57</td>
</tr>
<tr>
<td>G.1</td>
<td>CVS Data model.</td>
<td>164</td>
</tr>
<tr>
<td>G.2</td>
<td>CVA SPA components impact diagram.</td>
<td>165</td>
</tr>
</tbody>
</table>
List of Tables

1.1 Goals for each high level artifact of the project . . . . . . . . . . . . . 4
4.1 Example requirements descriptions. . . . . . . . . . . . . . . . . . . 27
4.2 Example use-case pre and post-conditions. . . . . . . . . . . . . . . 29
6.1 Test server specifications. . . . . . . . . . . . . . . . . . . . . . . . . 55
6.2 Mean generation times and standard deviation for each system (in ms). 56
H.1 Semantics of Unified Modeling Language (UML) elements and dia-
grams used in Helium. . . . . . . . . . . . . . . . . . . . . . . . . . . 169
H.2 Semantics of UML relationships used in Helium. . . . . . . . . . . . 170
H.3 Semantics of UML elements and diagrams used in Helium. . . . . . 171
Chapter 1

Introduction

Using Information Technology (IT) solutions to automate business processes has been the norm for many organizations in recent years. Despite the great benefits an enterprise can reap from adopting such solutions, developing one's own IT solution is not as easy as it might seem. If we take into account the ever increasing complexity of modern businesses and their operating environment, as well as the fast pace with which the modern world is changing, development and maintenance of such systems can easily become a daunting task, especially for non-software organizations. Of course, this is not to say that manual execution of business processes is the way to go. The benefits of business process automation are too many to ignore and if a business wants to survive the fierce competition in today’s business environment, automation is usually something every business strategy should consider. The ability to monitor business processes, evaluate them and adjust them as the business is evolving are only a few of the benefits an organization can get with business process automation (for more see [1]). To state it differently, automation can increase the level of efficiency of an organization and along with it reduce its operational costs and increase its capacity.

In an attempt to make software development easier and increase software projects’ success rates in terms of budget, time and requirements satisfaction, many software development processes have been proposed over the years. Some formal, some informal and some specific for an application domain like medical applications (see [2] for a short list of methodologies). Generic process frameworks also exist in which case an organization can create its own software development process taking into account its specific needs [3, ch. 1.4]. In addition to a well defined software development process, some fundamental concepts of the software engineering practice play a critical role in the delivery of on time, on budget, quality software as well. As per the IEEE definition found in [4]:

“Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).”

This definition shows that software engineering in general and software development processes specifically, deal with the operation and maintenance of software, too. The word *disciplined* also makes clear the fact that, a successful application of a software engineering approach depends not only on the defined process, but on the software engineering team as well. The last statement confirms the necessity of
understanding the big picture, when designing a software development process for a specific project (or an organization), meaning that team dynamics, team constitution and other human behavioral aspects should also be taken under consideration. Research conducted by McKenzie in collaboration with Oxford University, on 5400 large scale IT projects [5], showed that, on average, software projects in that category run 45% over budget, 7% over time and they deliver 56% less value than initially planned. Furthermore, 17% of large IT projects go so badly that they can threaten the very existence of the company. In addition, many executives leading IT projects believe that lack of motivation, clear goals, schedule underestimations, communication overheads and lack of alignment play a crucial role in the outcome of an IT project [6]. Combining the high failure rates shown above with the influence that soft factors exhibit on the outcome of a project, we can see that for an organization to increase the possibility of delivering a software product on time, on budget a process that accounts for soft factors is a preferred alternative to ad-hoc development approaches.

In this project (code named Helium) we are going to focus on a start-up company that is active in the marketing consulting field and is currently executing its business processes manually. That company is Customer Value (CV) and is based in Stockholm, Sweden. CV is a Business to Business (B2B) company that begun operations in 2011 and their products’ base consists of reports on sales data. First, we will define a software development process model and then follow that to build a distributed system that will be used to automate the core business processes of CV. The system will be designed with scalability in mind and an aim to allow for higher capacity of the business and controlled expansion that will not incur large operational costs (team scalability). The system will be available to CV’s clients as an on-line web application.

1.1 Problem Description

Although the software engineering practice has been around for many decades, as we saw in the previous paragraphs, enterprises are still facing issues in applying those engineering techniques to enhance their business through IT solutions. Our case study organization (CV) is one of the companies that is executing its processes manually. The main process the company is using to deliver its products has four steps:

1. Their clients provide sales data (see Appendix D for a sample data set) via email.

2. CV staff analyzes the data using a statistical package called STATA.

3. The resulting data are imported to MS Excel for further analysis, graph generation and PDF report generation.

4. The report is delivered to the client via email.

As the co-founder and senior manager of CV, Mr. Ove Svenneke, quoted “it takes somewhere between 24 to 40 man-hours to deliver a report”. With the current number of employees (i.e. three), CV can handle up to 20 clients in total. This does
1.2 Problem Statement & Purpose

Based on the aforementioned estimations for report delivery on a per-client basis, it is obvious that the small number of CV’s current employees (i.e. three) can only deliver a small number of reports within a month. The situation gets worse, if we take under consideration that most clients will request their reports towards the beginning of the month (i.e. once the sales data for the previous month are all available). It is easily seen that to support more customers we need at least a proportional number of employees. Of course, such a strategy is not at all competitive and it will definitely incur high operational costs that will restrict CV from scaling up easily, if at all.

The purpose of this project is to define a software development process model and then follow that to build a distributed platform system that will automate the core business processes of CV in such a way that scalability will be easily achieved, when needed, allowing for higher capacity of the business and controlled expansion that will not incur large operational costs.

1.3 Goals & Benefits

The main goal of this project is to demonstrate that a distributed architecture utilizing a database for transaction data storage can further enhance the performance of CV by increasing its capacity and allowing it to experiment with different implementation strategies until it finds what best fits its needs and strategies. Since it has already been proven by [8] that automation will increase the capacity of CV, we make the additional assumption that moving away from files (which [8] uses extensively) to a database for transaction data storage and retrieval will enhance performance even more. The first column of Table 1.1 shows our main goal along with a set of resulting side-effects.

To complete this project a software development process is initially specified that aims to formally define project execution. The benefits of such an approach are summarized on the right column of Table 1.1, but it should be noted that those cannot be proven in the context of a thesis. To do so would require the application and evaluation of the process multiple times for any assumption to be statistically
Chapter 1. Introduction

<table>
<thead>
<tr>
<th>Distributed Architecture</th>
<th>Software Engineering Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increase CV Capacity</td>
<td>Use Model as Formal Documentation</td>
</tr>
<tr>
<td>Lower Operational Costs</td>
<td>Decrease Learning Curve</td>
</tr>
<tr>
<td>Decrease Human Errors Frequency</td>
<td>Improve Change Management</td>
</tr>
<tr>
<td>Increase Extensibility</td>
<td>Monitor Project Execution</td>
</tr>
<tr>
<td>Increase Maintainability</td>
<td>Reduce Communication Overheads</td>
</tr>
<tr>
<td>Increase Scalability</td>
<td>Document Communication</td>
</tr>
<tr>
<td>Increase Flexibility/Ability to Experiment</td>
<td>-</td>
</tr>
<tr>
<td>Improve User Experience (UX)</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1.1: Goals for each high level artifact of the project

proven. In Helium we only go to the extend of describing the benefits and validating that they attempt to address real world problems by performing a set of interviews with a real world software development team. After the process has been used by a series of thesis projects provided by CV a study could be performed to evaluate if it actually provides the benefits discussed here.

The proposed process is model centric utilizing UML to model the different concepts. The models created during the analysis phase of the project can be used as the formal documentation of the system. Given that nowadays every professional in the IT industry is familiar with UML modeling to some extend, the created models can become a basis for communication among the different stakeholders involved, either technical or managerial ones. The existence of formal documentation can also decrease the time needed by a new member of the IT or business teams to start producing meaningful results and ease out communication problems occurring due to a possible geographical dispersion of the team. Quoting from [9], the absence of UML “...may be OK in some cases, but the lack of documentation becomes a problem when the design has to be communicated across time (to a developer who will join the team six months later) or space (to a developer working in another country)”. Stressing the importance of understanding the human nature of the development team once again, we see how UML can help in reducing communication overheads and decrease the learning curve for new members of the team. In addition, the software development process proposes the models to be traceable. The notion of traceability is often referred to as a metric of the quality of a specification [3, ch. 23.2.2]. Traceability means that the models specifying the system should be navigable. In other words, we should be able to see at any point in time and at any abstraction level of the model, which elements relate to each other. For example, we need to be able to trace all the way from an abstract feature requirement in the model to the database table or software component that is responsible for its implementation. This way impact of changes can be easily analyzed and the change management process can be formally defined, reducing human errors. Finally, utilizing simple tools like burn down charts, task-lists and collaborative project management applications, the project execution is monitored by all of the interested stakeholders in (almost) real time, making sure we don’t deviate from our initial goals. After all, quoted from [10], “if our customer isn’t happy, we built the wrong software”.

The proposed distributed architecture follows a component based design approach and uses Object Oriented (OO) design principles [11]. The benefits we get
from a distributed architecture that uses these principles are increased capacity due to better performance, lower operational costs due to automation and better performance, elimination of human errors due to automation, extensibility, maintainability, scalability and flexibility (ability to experiment) due to the service oriented architecture and finally, improved user experience due to the aesthetically pleasing components used in the client subsystem as we will see in Chapter 4. Finally, security is also slightly enhanced since a distributed architecture allows for more advanced deployment schemes to be used. An extensive analysis of the different deployment strategies is outside of the scope of this project, but a brief reference is made in Section 5.3.

1.4 Thesis Outline

The present document is divided into seven chapters. Chapter 1 is an introduction to the problem domain. It describes the problem we are trying to solve and also the goals we set for this project. Chapter 2 provides an overview of the finished system from a user’s perspective. Chapter 3 describes the methodology used for this project and also describes how the data were collected and analyzed. Chapter 4 provides a comprehensive description of the proposed software development process model, the business analysis performed and the proposed distributed architecture. Chapter 5 shows the implementation chosen for the proposed architecture. Chapter 6 presents the experimental testbed, the measurements performed on a deployed test system as well as the interviews held with senior developers in an attempt to identify the problems development teams face. Finally, Chapter 7 ends this document with a discussion on the analysis results and possible future steps that can be taken to improve further on the architecture and performance of the system.
Chapter 2
System Overview

This chapter contains an overview of the completed system from a user’s perspective. The main features of the system are presented and then we continue with the analysis and design of the system presented in chapters 4 & 5 respectively.

2.1 Main Features

The CVS application is an automation tool for CV. Below we discuss each of the main features CVS provides to CV and its clients. A clickable demo of the front-end component of the application can be found in [12] and a presentation of all the use cases on the completed system can be seen on the video found in [13].

2.1.1 Company & Employee Profile Management

In CVS both a company that is a client of CV as well as every employee using the system on behalf of that company are represented by a profile. A company profile cannot be used to log into the system and thus an employee profile is required to access CVS features. Representing the real world structure of CV’s clients, company profiles are organized in hierarchical structures of parent companies and subsidiaries. A parent company can have any number of subsidiaries, but a subsidiary can only have a single parent. Currently, company relationships can have a maximum depth of one level meaning that subsidiaries cannot be parents to other companies. Employee accounts can be assigned to companies in three different positions: a parent company manager, a subsidiary company manager or a subsidiary company employee. Parent companies have no employees.

The hierarchy represented in CVS is used as a Hierarchy Based Access Control (HBAC) system to provide access to different resources for different employees based on the position they hold within a client organization. A manager of a parent company has access to all employee profiles of all subsidiaries as well as all companies’ profiles. A manager of a subsidiary has access to all profiles of the subsidiary company he is managing and a subsidiary employee can only manage his own profile. In addition to the HBAC system, a Role Based Access Control (RBAC) system is used to provide more granular access to specific features of CVS. Each employee profile has a set of roles assigned to it that grant access to the different features. An Admin role is reserved for the employees of CV itself granting access to all resources and features of the system regardless of their position in the hierarchy. Note that
features requiring additional roles can still be hidden from an admin account, if he
does not possess that additional role. An Admin role holder can attach that role to
any other employee account as well as remove it.

2.1.2 Product Purchasing

Client companies can purchase products (report generations) from CV’s product
portfolio directly through the system. A CV employee (an employee profile with the Admin role
attached to it) can buy a product and assign it to any company. This is allowed so that purchases can be carried out through phone support as well, in
case a client is having trouble completing a purchase on-line. Products bought by a
parent company manager are available for use on all subsidiaries. On the contrary,
products bought by a subsidiary manager are only available within that specific
subsidiary. CV employees can also provide clients with free products as a marketing
tool to promote their services and new reports.

When a purchase is made, an invoice is automatically generated and assigned
to the company that the employee who made the purchase is a manager of. Parent
employee managers can see all invoices for the parent and subsidiary companies, but
subsidiary managers can only see purchases made for the company they manage.
Subsidiaries’ employees do not have sufficient rights to purchase products on their
own and also have no access to the invoices generated for such purchases. Note that
the HBAC system takes precedence over the RBAC system and thus there is no
role that can be attached to an employee of a subsidiary in order to enable product
purchasing for him. Refunding of invoices is also possible as long as the purchased
product have not been used yet. An invoice, once paid by a client can be set to
paid by a CV employee once the appropriate paperwork and phone confirmation is
provided by the client company. Of course, a purchase can only be refund by a CV
employee. Similarly, order details are available for each purchase following the exact
same access scheme used for invoices.

2.1.3 Report Generation

Report generation sits at the heart of CVS and is the major feature of the system.
Once a company has completed a product purchase (even before the invoice is paid)
the products automatically become available for report generation to those with
sufficient rights (as described in the previous section). To generate a report, a
manager or employee needs to upload sales data into CVS. Uploading of sales data
is a company specific action and the data are available to anyone with sufficient
rights to generate a report for that company. Uploading of data is performed in
batches of monthly transaction sets. Once all data are available a report generate
can be requested. Once the report has finished generating the manager or employee
will be able to download it. Once again, subsidiary employees and managers can only
generate and download the reports of their specific company while parent managers
can do the same for any subsidiary. Note that a report generated by the parent
company manager for a subsidiary is also available to the manager and employees
of that subsidiary.
2.1.4 Business Intel & Miscellaneous Features

Business intel is a set of useful features provided for CV employees to monitor the system. An implementation of a complete set of monitoring tools was outside of our scope so we only added a logging example for demonstration purposes only as can be seen in [12] under the Business Intel menu. The Business Intel menu item is also an example of a more granular access control using our RBAC system. As can be seen on [13] at 41:30, even though a CV employee has the Admin role available an additional role is required to view system logs. This way we can have greater control over what each employee within CV sees. For example, some statistics on sales could be limited only for CV’s top level management to see.

Finally, CVS is a multilingual environment allowing for its users to select a specific language to view their user interface in with additional support for left to right languages like Persian. Additional features regarding the appearance of the user interface are provided to allow for greater control of menu positioning and color themes of the application.
Chapter 3
Methodology

In this chapter we are going to discuss the methodology used for this project and how the data collection process was executed and the data analyzed. Chapter 6 presents the data collected during the evaluation of the deployed system and Chapter 7 discusses the results.

3.1 Philosophical Assumption

The main question we attempt to answer in this thesis is whether the system design proposed will perform better compared to the design proposed in [8], in the context of Customer Value. Since the evaluation is focusing on the proposed designs, both systems are seen as black boxes and no comparison is made at the technology stack level. This means that the differences in performance are not attributed to a specific component of the design (i.e. the application servers used), but rather to the design as a whole. Our philosophical view of the world is that of positivism, since we are attempting to prove a cause-effect relationship between different designs and performance. Quantitative data will be collected to prove this as it is discussed in later sections of this chapter.

3.2 Method & Research Approach

The proposed systems are evaluated in terms of their performance following the quantitative research method. A series of test cases are executed on the deployed system and a set of data is collected consisting of time and memory measurements. The same set of tests are run on the system found in [8]. The results are then used to compare the two systems in an attempt to validate our assumptions regarding the two different designs. Our research approach (i.e. strategy) is that of experiments since we run deterministic tests with predefined outcomes for the provided inputs and repeat them a number of times to increase their statistical significance.

3.3 Data Collection & Analysis

To verify our initial hypothesis that the system will benefit from switching to the proposed design, we perform report generations on different sizes of transaction data
sets and measure the total generation time for each. Since the available infrastructure only consists of a single server it is not possible to evaluate the benefits that the distribution of the system gives. Statistical analysis is the data analysis method used to evaluate the system’s performance and Microsoft Excel was used to perform the analysis and generate the graphs included in this document.
Chapter 4

Analysis & Design

In this chapter, we are going to discuss the analysis and design of the CVS. Section 4.1 talks about the proposed software development process, which includes the different phases defined as well as the different tools needed. Section 4.3 presents the analysis of the problem we are trying to solve. We try to understand what the business does, understand its environment and also define and categorize the requirements for this project. Later in this section, we turn to system analysis where we define all the system use-cases that realize the specified requirements. The different user interfaces are also specified at a conceptual level of abstraction, to assist during the implementation phase (discussed in more detail in Chapter 5). Finally, section 4.4, specifies the system architecture and the different components that make up CVS.

4.1 Software Development Process Model

The Helium software development process model is utilizing the generic process framework, described in section H.1, to create a customized software development process specifically for CV. Our main goals in specifying this process are noted in Table 1.1 and specified in more detail in Section 1.3.

Usually, the software development process is part of a bigger project management process. Since providing a complete project management process model is outside of the scope of this project we use the simple, three-step waterfall model shown in Figure 4.1. Our software process model will reside in the project execution phase of this project management process model and this is going to be our main area of focus.

![Figure 4.1: Top level project management process model.](image)

The Helium process, shown in Figure 4.2, is a variant of the framework process. The five activities were reduced to four, having the planning phase merge with the
communication phase and those two named requirements phase. The planning phase was eliminated and some of its responsibilities were transferred to the communication phase (now part of the requirements phase), because in the context of a thesis project the resource management aspects of the project (i.e. monetary, human resources etc.) are not going to be the main focus. The approach followed is an iterative one to allow for a closer cooperation with the client and ensure that the project is on track at all times [10]. Furthermore, change is also handled more gracefully due to the iterative nature of the process and the traceability of the models used (described later in this section). The duration of an iteration (i.e. a complete execution of the whole process once) is set to twenty (20) working man-days (equivalent to eight man-hours per day). The weekends are considered non-working days and will be used as contingency time in case of an emergency or to move ahead of schedule in cases that this is possible. Since a thesis project lasts for twenty weeks, it is expected that five whole iterations will be completed. It is also expected that the requirements phase will take a lot longer during the first iteration than during the rest of them. In any case, the process will be followed as is. The orange arrows shown in Figure 4.2 were added to allow for the flexibility needed in the case of a thesis project (i.e. a one man project). This way we can jump from any step backwards in case an error is identified and needs to be fixed prior to a whole iteration completing. Although the process is geared towards the execution of projects as thesis projects (since this is the current strategy of CV), it is equally capable to handle real-world projects with minor changes to include monetary and other resources estimations.

![Figure 4.2: The Helium software development process model.](image)

Regarding the monitoring aspects of the process, a burn-down rate chart will be maintained during the execution of the project to monitor progress and perform adjustments in case of schedule slippage. Figure 4.3 shows the burn-down chart of this project. A burn-down chart monitors how fast the development team completes the work available against the remaining number of days until the deadline of the project [10, p.104]. The grey straight line shows the ideal execution rate. As long as the green line is above the gray line, the project is on track and proceeding as expected (or better). Once the green line goes below the gray line, we should be cautious and be on the lookout for possible slippages coming up. Being above the ideal line (gray line) is not always a preferred way to execute tasks. Moving forward really fast might burn out the development team and cause trouble in the project later on.

To reduce communication overheads and allow smooth execution of the project from a geographically distributed team, the process mandates the use of a collaboration tool for all communication. The minimum requirements for such a tool are the following

- **Task Lists.** The selected tool should allow the team to specify a task list
4.1. Software Development Process Model

Figure 4.3: The Helium project burn-down rate chart.

with all the tasks that are identified at the beginning of the project. Allowing sub-tasks to be defined is also a necessity.

- **Scheduling.** The selected tool should be able to set deadlines for the completion of each task (or sub-task) and also allow the assignment of a task to a team member.

- **Commenting.** The selected tool should allow commenting on the different tasks by the members of the team.

- **Uploading.** The selected tool should allow the team to share files as needed.

This way the team and CV managers will cover all their communication needs with a single tool reducing communication overheads and allowing for permanent storage of all communications for future reference. To achieve the same results with email a more complex process would be required to collect, sort and store all email for future reference.

In addition, the process also mandates the use of traceable UML models to specify the system. The requirement for formal UML models is there to act as a documentation to the system. Since thesis projects usually lack the time to provide full technical documentation (i.e. a developer’s guide) a model can play the dual role of clarifying requirements and also acting as a form of formal documentation in itself. In the future, when new members join the development team, working on a new project or an already executing one, going through UML models will greatly help reduce the time needed before a new member is able to offer valuable services to the company [9]. The modeling will mainly focus on Platform Independent Model (PIM) models (detached of any specific technology stack) to account for the limited time available for thesis projects. Platform Specific Model (PSM) models can be optionally included, if time permits, otherwise a top level model linking the PIM with the PSM will suffice. The reason for preferring the PIM models over the PSM
ones is that a PIM focuses on defining the system, from requirements to solution architecture, in a platform agnostic way. Technologies change really fast and a start-up company like CV will want to experiment with different technologies before selecting a more permanent software stack. For that reason, to avoid maintaining continuously changing models we primarily focus in PIMs. Finally, all models should be traceable to allow for better change management. When a model is traceable, each change that is requested can be evaluated in terms of the impact it will have on the already existing system. In Section 4.3 we will see examples of diagrams that can be used for impact analysis of the system to allow for a more graceful and safe accommodation of change. For the modeling needs of the process, any UML modeling tool will do. Next we discuss each phase of the process in greater depth.

4.1.1 The Requirements Phase

The first phase of the process deals with the analysis of the problem at hand. This includes the understanding of the problem domain as well as the requirements of the system that we are trying to build. Since it involves a lot of communication, it is considered the most difficult part of the Helium process. During the Helium project, input from CV executive, Mr. Ove Svenneke is received to define the scope and the initial requirements for the system. First, the functional and non-functional requirements of the system are documented in an attempt to reach an initial consensus. Secondly, the set of requirements specified are split into categories for easier requirements management and handed over to CV once again for approval. On every subsequent cycle the requirements are updated and re-submitted for approval by CV. The process defines a color code of six colors to specify the status of each requirement. Figure 4.4 shows the different colors in the scheme.

- **Proposed Requirement.** A requirement specified by the development team either as a new requirement or one retrieved from a formal document (like the FRDs in appendices B & C).

- **Unclarified Requirement.** A requirement that was previously approved, but later some kind of ambiguity was found. This can be due to the elicitation of new conflicting requirements or missing information in the documentation of an approved requirement.

- **Approved Requirement.** A requirement that CV management has approved and the development team can move forward with its implementation.
4.1. Software Development Process Model

- **Validated Requirement.** This is a state between proposed and approved. When a requirement is proposed and approved the development (or requirements analysis) team might add additional information or remove information. Such changes to an already approved requirement change its state to validated and is pending approval by the CV management team (or other appropriate stakeholders) to return to approved state.

- **Frozen Requirement.** A requirement that was not able to fit within the iteration it was planned for changes its state to frozen. Frozen requirements can return to approved state, when they are scheduled for implementation on an other iteration.

- **Implemented Requirement.** Once the use-cases realizing a requirement are all implemented and tested, the requirement is considered satisfied and is marked as implemented.

Apart from the requirements specification, the different actors of the system (i.e. users or other systems that interact with it) are identified and the use-cases that realize the documented requirements are also specified during this phase. Each use-case is defined at a high level along with its pre-conditions, post-conditions and activity diagram specifying the communication between the system and its actors to accomplish the use-case’s goals. Use-cases also share the same status states as requirements. Note that for a use-case to be able to be implemented, it must have all the requirements it realizes set to the approved state. Additionally, a high level user interface model is created to specify which user interfaces should be build to cover the needs of each use-case. Finally, a business domain model is created in an attempt to understand the business in its usual operating environment.

*Work Products Generated:* A formal specification of the system requirements and system use-cases along with a business domain model and a user interface model.

4.1.2 The Modeling Phase

The second phase of the process deals with the architecture of the system and the technologies that are going to be used in its implementation and deployment. The parts of the system that will satisfy the requirements set are identified and their interfaces to the rest of the system are defined. Information architecture and data modeling also happens at this stage, before the implementation begins. An additional goal here is to define a tool-chain and frameworks to use that will ease system maintenance in the future and allow developers to focus on business logic tasks rather than process management tasks [14].

*Work Products Generated:* A PIM of the system (formal to the extent possible by time limitations) as well as an overview of the technologies that are going to be used for the system implementation. The later also includes the tools used in the development life-cycle.

4.1.3 The Construction Phase

The third phase of the process deals with the implementation of the design specified during the *modeling phase.* Implementation of the design using the technologies
Chapter 4. Analysis & Design

specified will begin and the deployment platforms will be used for testing and evaluation of a properly functioning system. To facilitate easier sharing, a source code version control system should be used to allow for source code commits and change logging. This allows the company to have a common code-base from which all of its projects can begin. Infrastructure set-up and configuration (e.g. setting up and configuring a web server) are also activities executed during this phase.

Work Products Generated: The actual repositories of system source code and all deployment artifacts needed for proper operation.

4.1.4 The Deployment Phase

The final phase of the process deals with the deployment and testing of the system in a test environment. The client will test the system and comment on its proper operation, if possible, otherwise other ways for the customer to see the system in action can be substituted (e.g. screencasts). Note here that it is advisable that the test environment is as close as possible to the real world deployment infrastructure [15]. At the end of the iteration the client will comment on the progress and the next iteration begin with changes or amendments to the requirements.

Work Products Generated: Fully functional version of the system parts developed and optionally artifacts that describe current system deployment operation (i.e. screencasts).

4.2 Process Application

Having defined a software development process, we can apply it in order to analyze our problem and design, implement and test a solution for it. To apply the process, we first need to decide upon a tool-chain in order to carry out the different activities necessary. As specified in Section 4.1, a tool is necessary to facilitate easier communication. In this project we choose an on-line project management tool called Asana, found in [16], that covers all the requirements required by the process. For our UML modeling needs we choose Enterprise Architect from Sparx Systems [17]. The free version of Asana works perfectly for our case, but Enterprise Architect requires a paid license for commercial use.

As expected, five iterations were completed during the twenty-week long execution of the project, having each iteration set to a one-month long duration. Each of the four phases within an iteration (as described in 4.1) was scheduled for a duration of one week. In the following chapters and sections, we present the final results of each phase, that is after the completion of all iterations. Sections 4.3.1 & 4.3.2 discuss the business and system analysis respectively, performed during the Requirements phase. Due to the small size of the project, most of the analysis (business and system) was performed during the first iteration leaving the Requirements phase of the subsequent four iterations mainly for refinements and for dealing with requirements’ problems identified during implementation and testing. Since a lot of new technologies were used for implementation which we had no prior experience with, this fast head start in the Requirements phase was something desirable that gave us the ability to have some extra contingency time to deal with expected problems during implementation.
During the Modeline phase, the system architecture was defined as described in Section 4.4. During the first iteration an initial architecture was defined for the whole system focusing mainly on the architecture of the front-end subsystem. The next four iterations gradually moved our focus to the backed subsystems and the frameworks used there. The reason we began our design with the front-end part of the system was that we had the least experience with the technologies used there compared to those in the back-end. This way any important problems would show up early in the process and we would be able to deal with them following a safer, more controlled approach (i.e. not panic).

The second to last phase, the Construction phase, dealt with the actual implementation of the system. More precisely, the first two iterations were devoted to the implementation of the front-end client of the system (discussed in Section 5.1.1) while the rest three to the implementation of the back-end subsystems. The front-end client was initially implemented as a clickable demo and subsequently (second iteration) using a fake API to simulate the existence of a back-end. The back-end was implemented gradually as well with an initial deployment of the software stack used (third iteration - discussed in Section 5.1), the implementation of the business logic of the system (fourth iteration) and finally the implementation of a product (report generation - discussed in Section 5.2) during the fifth iteration. As we will see in Section 4.4.1, the proposed architecture allows for the isolation of the front-end and back-end subsystems giving us more freedom over when to implement the different components.

Finally, the Deployment phase, was used to test the system either on the development machine or by CVs manager Mr. Ove Svenneke himself. During the first iteration the deployment consisted of a clickable front-end client with hard coded data allowing for simple navigation in order for CV to get an overall view of how the final system UI will look like. A fake API to simulate the existence of a back-end was added during the second iteration to allow for CV to log-in and navigate as if the system was complete. The remaining iterations were devoted to the back-end, dealing with the testing of the back-end software stack, the testing of the implementation of the business logic (accompanied by a video presentation of the execution of all the identified use-cases of the system) and also the generation of a report using the system.

4.3 Problem Analysis

In this section, we are going to focus on the work products created during the Requirements phase of our process applied as explained in section 4.2.

4.3.1 Business Analysis

In business software engineering, we build software solutions in the context of specific businesses and those solutions are expected to provide some value to those businesses. The same way an architect or civil engineer designs a house before building it, software engineers design their systems prior to implementing them. A common problem in software engineering is that multiple stakeholders hold opinions regarding the system’s functionality and a consensus needs to be reached as to what
the system should actually do before any development effort begins. Quoting from [3], “a concerted effort should be made to understand the problem before a software solution is developed”. For that reason, we begin with an analysis of the business and a specification of its requirements.

**Customer Value Sweden AB**

Customer Value (CV) is a start-up company that aims to provide organizations with the tools to build loyalty relationships with their customers and do so in a proactive way. Analyzing sales data, CV provides businesses with a segmentation of their customer base into distinct categories, allowing them to target each category in a different, more suitable way, with the ultimate goal of increasing conversions and improving retention. As described in 1.1, CV follows the process shown in Figure 4.5. Receiving sales data through (unencrypted) email and aggregating manually has proven to be a time consuming process [8, p.43] and using multiple software packages for data analysis is error prone and requires multiple licenses as the number of employees that perform report generations increases. The need for “a few days” to generate and deliver a report to a customer is acceptable when CV has a small customer base, but when its customer base grows, “a few days” could be a serious threat to its sustainability and growth.

![Figure 4.5: Manual process executed by CV to deliver its products.](image)

In addition to the report generation and delivery process, other operations within the enterprise need to be carried out for smooth operation, such as collecting clients’ information, generating statistics to help with decision making processes, generating invoices and also sales and marketing processes.

**Customer Value Business Domain**

In an attempt to understand the operating environment of the business under consideration (CV), we model its business domain as shown in Figure 4.6. Figure 4.6 shows that CV, acting as an employer, employs one or more employees (account managers, sales people or managers). Account managers deal with CV’s clients. CV has one or more clients which are other companies acting as sellers to consumers (customers of CV’s customers). For each of its clients, CV is preserving some information such as contact info, addresses and others. A company can have one or more companies as its subsidiaries and each company can employ one or more employees. A consumer can execute one or more transactions with the company when buying the products or services offered by the latter and each consumer belongs to a specific market (for that specific transaction). Each company, that is a client of CV, generates one or more sales data sets for different time periods of sales. Additionally, each company (CV’s client) can be active in one or more markets. A market is defined as a geographical region that the company is selling in. CV can generate a report for the activity in a market and it can be
4.3. Problem Analysis

- One report for one market
- One report for multiple markets (summary report)
- Multiple reports for one market (in different time periods or different languages/currencies).

A report can be in only one language. Multiple languages are never mixed in a single report although a report for the same market can be generated in different languages. A report can be any of the two available products CV is offering (i.e. CV Predictive Report or New Biz report). Finally, the predictive report also comes with a Return Report which is a product specific artifact.

![Conceptual level business domain model](image)

Figure 4.6: Conceptual level business domain model.

As we can see, modeling already helps in capturing a big amount of information about the operational environment of CV in a single diagram. It should be noted that this model is not a complete model of CV’s business domain. What we model are the parts of it that are of immediate interest and add value to our development process. For example, the tax office in Stockholm is part of the environment of CV since it interacts with it, but it is absent for that model because our automation system automates internal processes and not accounting processes. If our project’s goal was to integrate with the tax office, then we would have included it in the model as well. In other words, it is important to be able to see the big picture in the correct scope [15]. For the complete set of descriptions of each entity in the diagram see the on-line model of the system available in [18].

Requirements Model

Going through the Functional Requirements Document (FRD) included in Appendix B and having a number of discussions with CV senior manager Mr. Ove Svenneke,
we created a requirements model for the business that describes its needs in the context of the software system to be built. Stated differently, we specified what the system should do in order for it to be valuable to the business.

The requirements were split into two major categories, functional requirements and non-functional requirements. Higher emphasis was given to the functional requirements due to the fact that most of the non-functional requirements require a production deployment to evaluate them, which is not available. Figure 4.7 shows the high level diagram of the requirements model.

Figure 4.7: Requirements model, high level categories diagram.

In each major category we can see the sub-categories the requirements were split into. For functional requirements, these are

- **Business Intel.** This category contains requirements that have to do with business statistics such as sales statistics, logging facilities etc. Figure 4.8 shows a diagram with the business intel requirements and their relationships.

Figure 4.8: Requirements model, business intel requirements diagram.

- **Miscellaneous.** This category contains requirements that have to do with miscellaneous features of the system that do not directly fall under any other category. For example, multilingual content, help facilities are such requirements. Figure 4.9 shows a diagram of miscellaneous requirements.

Figure 4.9: Requirements model, miscellaneous requirements diagram.
- **Company Hierarchy Management.** This category contains requirements that have to do with managing company accounts in the system as well as the hierarchies of companies supported (i.e., subsidiaries and company employees identified in 4.3.1). Figure 4.10 shows a diagram with the requirements of this category and their relationships.

![Figure 4.10: Requirements model, company hierarchy management requirements diagram.](image-url)
- **Account Management.** This category contains requirements that have to do with managing user accounts. Figure 4.11 shows a diagram with the account management requirements and their relationships.

![Account Management Diagram]

Figure 4.11: Requirements model, account management requirements diagram.
- **Product Management.** This category contains requirements that have to do with the management of products (reports) sold on-line by CV. Purchasing requirements are also in this category. Figure 4.12 shows a diagram of the requirements in this category.

![Figure 4.12: Requirements model, product management requirements diagram.](image-url)

Figure 4.12: Requirements model, product management requirements diagram.
• **Report Generation.** This category contains requirements that have to do with the automation of the report generation process. Figure 4.13 shows a diagram of the requirements in this category.

![Diagram of Report Generation Requirements](image)

Figure 4.13: Requirements model, report generation requirements diagram.

For non-functional requirements, the sub-categories are shown in Figure 4.14 where we also see the requirements in each sub-category (i.e. security and finance).

![Diagram of Non-Functional Requirements](image)

(a) Financial Requirements  (b) Security Requirements

Figure 4.14: Non-functional requirements diagrams.

All requirements model diagrams presented in this section depict the status of the requirements at the end of project. For that reason, requirements are either
implemented (satisfied) or frozen (left for the next iteration/project). Note that requirements that have sub-requirements defined stay in approved status until all of the sub-requirements are complete. Each requirement is accompanied by a short description that specifies it in more detail. Table 4.1 shows some example requirements’ descriptions. For the complete set of descriptions see the on-line model of the system available in [18].

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Account (REQ00025)</td>
<td>CVS should allow the registration of new user accounts on the system. A user account is possible to be created on demand by CV staff, the group manager of an umbrella company, the company manager of a child company or by an employee himself.</td>
</tr>
<tr>
<td>Free Products (REQ00023)</td>
<td>CVS should allow CV staff to offer products to a customer for free, if such a need arises (i.e. for marketing purposes).</td>
</tr>
<tr>
<td>Group Managing Account (REQ00076)</td>
<td>A single account attached to an umbrella company can act as its managing account and the overall managing account of the whole group of companies. A group managing account can be attached by a CV staff member only.</td>
</tr>
<tr>
<td>Account Authorization (REQ00037)</td>
<td>CVS should use a role based access control system. User accounts possess roles that define their access to system resources as well as their ability to edit and create information on the system in addition to their position in the defined hierarchy.</td>
</tr>
</tbody>
</table>

Table 4.1: Example requirements descriptions.

Having performed the above analysis of what we need to do, we are now in a position to proceed with the analysis of the system that will perform the desired functionality. The system actors and use-cases can now be defined and the user interfaces can be conceptualized at a high level.

### 4.3.2 System Analysis

The second step in analyzing the problem is to identify what the system should do in order to satisfy all the requirements set in Section 4.3.1. We are still concerned about the “what” of our problem, but now we shift our attention from the business to the system itself. In a way, we describe the same concepts, but from a different perspective. To accomplish this we continue with our modeling work and define the system actors, system use-cases and system user interfaces at a conceptual level. These artifacts, along with the requirements model created in Section 4.3.1, are going to be used to design a solution for our problem.
System Actors

An actor is an external entity to the system that interacts with it in any of the use-cases the system supports. An actor can be an actual person, for example, a user of the system or any other system that our system integrates with (e.g. an accounting system to log payments of on-line orders). In our use-cases model, we divide our actors into two categories, primary actors and secondary actors. Primary actors are the main users of the system that directly depend on it to carry out their daily work activities or utilize the services provided by CV. External systems that CVS depends upon to provide value to the business can also be found in this category. Secondary actors are people or external systems that play a secondary role when defining the system. For example, an accounting system will not guide the system design more than a client of CV would. It should be clarified that secondary actors are not less important than primary actors. An accounting system is still important for the proper operation of the system and CV itself. The only difference between the two categories is that from the development perspective we primarily care about the primary actors when defining the system. Figures 4.15 & 4.16 show the actors identified for the problem at hand.

![Diagram of System Actors](image)

Figure 4.15: Primary actors identified during system analysis.

All human actors are considered to be users of the system that either work for CV or for a client of CV. External systems like Point of Sale (POS) and Exchange Rate Provider are systems that CVS is using to receive information from during normal operation. A POS is there to cover the ability of the system to receive transactions directly from POS of clients and providing real time statistics instead of following a generation process for each month as clarified in [7]. The secondary actors that were identified, primarily deal with the financial aspects of the system and that is accounting and payment management for on-line purchases.
4.3. Problem Analysis

**System Use-Cases**

Having identified who (and what) is interacting with the system we can now generate a set of use-cases that will satisfy the elicited requirements for the identified actors. To make our model more readable and easy to follow we categorize our use-cases in the same categories as we categorized our requirements (account management, product management, report generation, company hierarchy management, product purchasing, miscellaneous and business intel). Figure 4.17 shows the product purchasing use-cases that were specified to satisfy the product purchasing requirements. Each use-case has an activity diagram that specifies the steps of interaction between the system and the actor involved in the use case. Figure 4.18 shows the activity diagram of the Log Into User Account use-case. In addition pre-conditions and post-conditions are specified for each use case. For example, in the Log Into User Account use-case, Table 4.2 shows the pre and post conditions. For the complete set of use-case diagrams and activity diagrams see the on-line model of the system available in [18]. Note that to see the pre and post-conditions, the use-case needs to be selected from the hierarchy on the left since clicking on it will open the activity diagram.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Pre-conditions</th>
<th>Post-conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log Into User Account</td>
<td>An active user account must exist &amp; User must navigate to the appropriate UI for the task.</td>
<td>A logged in user account with available options to use.</td>
</tr>
</tbody>
</table>

Table 4.2: Example use-case pre and post-conditions.

**System User Interfaces**

So far, we have identified what the system should do in order for it to be valuable to the business (requirements model), who the main people and systems are that CVS will interact with (actors from the use-case model), how the interaction between the system and the actors should happen in order to satisfy the requirements (use-case model). The last step in our system analysis process is to identify the user interfaces (in a conceptual level) that the actors are going to use in order to perform
the specified use-cases. Conceptual identification of user interfaces means that we
don’t care about the “look’n’feel” of the user interface yet, but only about the goals
of each user interface of the system in terms of the use-cases it is used in. Once again
we use the same categories used in requirements and use-cases to categorize the user
interfaces. Some early navigability information are also included in the diagrams to
show how the interface could be navigated. Figure 4.19 shows the screens needed for
the actors to be able to execute the use-cases defined for the account management
category and bellow are the goals defined for one of those screens (the Account
Details screen).

1. Deletion of the account.
2. Activation/deactivation of the account.
3. Attach account to an existing company.
4. Attach managing account to the company.
5. Edit account.
6. Add free products to account’s company.
7. View account statistics.
Figure 4.18: Activity diagram of the Log Into User Account use-case.

For the complete set of user interface diagrams and their goals see the on-line model of the system available in [18].

**Impact Analysis Diagrams**

As stated in the process description in Section 4.1, all models should be traceable to allow for better change management and maintenance. Figure 4.20 shows an impact analysis diagram for the requirements, use-cases and user interfaces of the *business intel* category. An impact analysis diagram is a cross section of the aforementioned three models to allow for better evaluation of requested changes to the system at a high level. The term high level means that we don’t evaluate the impact a change will have on our system in terms of system components such as database tables, but only at the high level analysis models like requirements, use-cases and user interface models. Although the process does not specifically define it, a simple change management process could be set to make sure all changes requested are evaluated first and implemented in a graceful manner that will not affect other operations of the system. This way maintenance of the system becomes a more streamlined activity and changes can be approved prior to being implemented. In Section 4.4 we will see impact analysis diagrams that span the whole system architecture from abstract requirements to database tables. A simple change management process could be the following
• **Step 1.** A change is requested by an involved stakeholder.

• **Step 2.** The business analysis team evaluates the change at a high level using an impact analysis diagram to find conflicting requirements, check which other use-cases will be affected as well as find which user interfaces might need to change.

• **Step 3.** The work required for the change is estimated by the development team.

• **Step 4.** The change is either approved or rejected by management.

• **Step 5.** If approved, the change is implemented following the specified process.

• **Step 6.** The requesting stakeholder is informed about the outcome of her request whether positive or negative.

The formality of the change management process could, of course, be adjusted depending on the complexity of the system at any specific point in time. More stakeholders and more frequent change requests could be an indicator that a more formal process is required to handle the incoming changes. For the complete set of impact analysis diagrams for all categories see the on-line model of the system available in [18]. An impact analysis diagram for the account registration process that spans all layers of the system can be found in Appendix A.
Figure 4.20: Business intel impact analysis diagram.
Chapter 4. Analysis & Design

4.4 Problem Solution

So far, we have completed the first phase of the proposed development process and we have available all the work-products defined for it (in reality this was completed over five consecutive iterations). That includes the business domain model, the requirements model, the use-case model and the user interface model. Moving forward, it is now time to design a solution for our system. The “what” question was answered and we shift our focus on the “how”.

4.4.1 System Design

The solution architecture chosen for CVS is based on principles of OO design and the microservices architecture as described in H.3. As we have already mentioned, it is really important to always keep in mind the human nature of the development team while designing a software system. For that reason, our system is split into three highly decoupled sub-systems that communicate with clearly defined interfaces. This way the different development teams can work independently from one another reducing communication overheads. As noted in [19, p.19] “since software construction is inherently a systems effort...communication effort is great...adding more men then lengthens, not shortens, the schedule”, thus reducing communication we can reduce the effect in schedule length an increasing number of team members can have and make project management smoother.

Focusing on principles like “separation of concerns” and “divide and conquer” the three major subsystems of CVS are the following,

- **CVS SPA.** This is the web front end of the application designed as a Single Page Application (SPA). This subsystem acts as a client to CVS Core.

- **CVS Core.** This is the core back end system that provides an interface to the CVS SPA in order to execute the automated business processes.

- **Genny.** This is the generation engine of CVS which is responsible for performing the calculations needed in order to generate a report and also graphs and PDF files. An interface is provided by Genny in order for interested clients to be able to request report generations.

Having a specified Application Programming Interface (API) between our three subsystems, enables the development of each in complete isolation from the others, making possible to increase team members count without having a great impact on project schedule. In Chapter 5, we will see how the software stack chosen also helps teams work as independently as possible from one another. The exposure of the core business processes through an API also makes the development of many different clients for CVS Core possible such as Android applications, iOS applications or even desktop applications. Again, the development of those can commence completely in parallel with all other development efforts due to the existence of an API.

Due to the utilization of Hypertext Transfer Protocol (HTTP) as our transfer protocol, since we are following a web services architecture, the frameworks or programming languages used for the implementation of the different subsystems can be independently decided upon different project factors. For example, the CVS SPA
4.4. Problem Solution

subsystem could be implemented using a framework that the front-end team is already familiar with in order to reduce learning delays. If the development of CVS SPA is outsourced to another company that has more experience in a specific framework, then that framework could be used instead to benefit from the teams prior experience. Performance benefits could also be a factor especially when implementing Genny. Designing is such a way we already see the benefits we get in project management and also the possibilities for experimentation that are now available to the company (CV).

The Data Model

One of the most important parts of any business software system, is its data model. The data model specifies the data the information system is going to manipulate and their relationships. Going through the models we have generated so far we can start identifying the data model for CVS. The business domain model shown in Figure 4.6 can also play an important role in identifying the data model entities for the system. Figure 4.21 shows the data CVS is capable of handling and their relationships.

![Figure 4.21: CVS Data model. See Appendix G for an enlarged version of this model.](image)

The main entities are the Employee, Company and Report entities. The Employee is related with the Role entity to enable the RBAC facilities specified in requirement REQ00037 (as seen in Table 4.1 and [18]). In addition, the relationship between the Employee entity and the Company entity as well as the relationship of the Company entity with itself enable the HBAC facilities specified in the same requirement (REQ00037). The Report entity models the reports the system will be able to generate. The rest of the entities are there to facilitate the rest of the features of the system based on the requirements model. The Activation entity is there to allow new accounts to be activated prior to using them, the Basket, BasketItem, OrderHeader, OrderItem, Invoice and Product entities deal with product management and purchasing and finally the entities SalesData, Transaction, Currency and Country deal with report generation and transaction data handling. System logging as per REQ00087 is handled by entity SystemLogEntry.
CVS SPA

CVS SPA is the Single Page Application (SPA) web client of CVS. Following the same principles of “separation of concerns”, “divide and conquer” and high decoupling of the different components we further benefit in terms of maintainability and team management the same way we described in the introduction of this chapter for the high level decomposition of the system into three major subsystems. Figure 4.22 shows the high level architecture of CVS SPA.

![Figure 4.22: CVA SPA high level architecture.](image)

CVS SPA was designed following the Model - View - View Model - Controller (MVVMC) design pattern. The different components of the front end client have a View which defines how the component is presented to the user (i.e. the colors, layout, images etc.), a View Model which contains the information shown on the View, a Controller which populates the View Model and provides additional functionality to the View and finally a Model which is the main source of data for the View Model. To eliminate any ambiguity between the View Model and Model of a component in the CVS SPA, we give the example of a user interface that shows account information. The Model is where all the account information is stored while the View Model is where only the subset of the information shown on the View is stored. This way a Model can be shared among many View Models instead of having each View Model maintain its own state. In other words, the Models provide a pool of information from which the View Model can retrieve. This way we know that the information shown on all Views are always in sync all across the application since they are retrieved from a common place, the Model. In our example of displaying account information, if a user visits a page to edit their account and change their name, next time they visit the account details page the name will have been updated since the account details view and the account edit view share the same Model from which their respective View Models get their information.

The Service part of the architecture is where the implementation of the CVS Core API is put in order to hide away the API implementation details from the rest of the SPA application. The service layer in the CVS SPA provides the calls available from the API to the rest of the SPA application in an opaque way. Having
this abstraction layer between the CVS SPA and CVS Core, any changes to the API of the latter will only affect the Service layer of the first. The rest of the layers are going to stay the same. Because of the way our dependencies are defined in the architecture, the same holds true for the Model layer. The Model layer acts as an abstraction layer between the Service layer and the rest of the application. To state it differently, each layer only needs to know about the existence of the layer below it. The dependency of the View layer and the View Model layer is bidirectional to ensure that information can also be retrieved from a View instead of just shown (i.e. a value typed in a text box).

The architectural model described above is consistent with our approach to ease maintenance efforts and the management process through proper design. The CVS SPA team can now be split into further sub-teams that deal with each layer (or set of layers) independently since the interfaces are once again specified. A possible breakdown could be the following,

- **Service Team.** Team that deals with the implementation of the CVS Core API that provides the API to the rest of the CVS SPA application following some convention.

- **Model Team.** Team that deals with the retrieval of information needed by the CVS SPA through the provided API and makes it available to the rest of the application.

- **Front End Team.** Team that deals with retrieval of information from the model to display it to the user and also retrieve it from the different forms of the application. This team created Views in Hypertext Markup Language (HTML) since this is a web application.

- **Web Design Team.** Team that deals with the colors, images and other artistic aspects of the Views. Due to the ability to separate content and appearance in web application (through Cascade Style Sheets (CSS)) this team and the previous one can work independently, too.

Such a division in human resources can further reduce the effect discussed in the beginning of Section 4.4.1 and [19] and allow the development team to scale gracefully without having a great impact in schedule from communication overheads. Of course, such a big set of teams is only valuable when the system is big enough to require it and in no case is necessary in a thesis project. A word of caution here is that this approach of dividing and conquering cannot be carried on forever. In order to divide the teams further, interfaces must be specified so as communication can be minimized (which is the point of dividing in the first place). The more we divide the teams the more interfaces are required, making the design process longer. Eventually, the interfaces will be so many that dividing further will prolong the design process and API specification process so much that it will outweigh any benefit we get from reduced communication overheads. For CVS SPA we apply this division only on its highest level (shown in Figure 4.22).

The different components needed by CVS SPA are divided in categories similar to those of the requirements and use-case models. All the components are depicted in Figure 4.23 (a larger version of this diagram is available in Appendix G). This is an impact diagram for the CVS SPA application specifically, that can be used, as
we explained previously, to handle changes and evaluate the impact they will have in the different parts of the application before actually applying them.

The components in the *Inbox* category although they have a *View*, a *View Model* and a *Model* they lack a *Controller*. That is because although we have implemented a messaging component in CVS SPA this is not functional (only its “look’n’feel” is available as a demo) and the corresponding requirement (*REQ00052 - Customer Contact*) is frozen. Finally, there is a dependency from the *Controller* layer to the *Service* layer. The only reason this exists is because the multilingual content facilities do not require a model for their operation. An alternative solution, an probably better, would be to include a model component for the multilingual content facilities as well so that all components follow the architecture with no exceptions. The *Model* layer could also be used as a caching layer to reduce the calls made to the CVS Core and thus reduce server load. For the complete model of CVS SPA see the on-line model of the system available in [18].
CVS Core & Genny

The next two subsystems (CVS Core and Genny) follow the exact same architecture so we present them together. Figure 4.24 shows the high level, layered architecture followed by our back end subsystems. The chosen architecture is a five layer architecture that contains the following layers

- **Presentation Layer.** This layer contains all the components that are responsible for exposing system functionality to the outside world in any way, whether this is an API, a web application (CVS SPA falls in this category) or even a Command Line Interface (CLI).

- **Business Logic Layer.** This layer contains all the components that implement the business logic defined by the requirements model and follows the flow defined by the activity diagrams of the use-case model.

- **Abstraction Layer.** This layer is here to implement the design pattern called Dependency Inversion (DI) that allows us to build our system as a platform instead of as a simple monolithic application.

- **Dependency Layer.** This layer implements the abstractions specified in the Abstraction layer as part of the DI design pattern.

- **Common Layer.** This layer contains some common artifacts that can be used in an application wide sense such as translations for multilingual content and configurations.

![Figure 4.24: CVS Core & Genny high level architecture.](image-url)
The **Presentation Layer** contains two major parts. The **User Interface** package, which in our case is just the CVS SPA application described in the previous section and the **System Interface** package which is the API provided by our back end system for consumption by its clients. As we can see, the single point of dependency between the **User Interface** and the **System Interface** (back end system) is what enables the aforementioned high decoupling of our subsystems. The only event that can trigger a change in the whole CVS SPA Application (or any other client application in that package) is a change in the component contained in **System Interface** (in our case just one component implementing the API). If an Android application or an iOS application was available, it would be put in the **User Interface** package.

![Figure 4.25: High level architecture, presentation layer.](image)

The **Business Logic Layer** contains two major parts. The **Services** package and the **Domain Model** package (Figure 4.26). The **Services** package contains a set of services that implement the business logic of the system as described by the requirements and use-case models. Each service covers a specific set of related business functions and it is exposed to the outside world through the **System Interface** as a service endpoint consumed by the applications in the **User Interface** package (in our case only CVS SPA). The complete set of services is shown in Figure 4.27 with the services of Genny marked in a separate frame.

![Figure 4.26: High level architecture, business logic layer.](image)

The **Domain Model** package contains the representation of the data model that the application is going to be handling as part of its normal operation. The approach followed here is that of an “Anemic Model”. What that means is that the **Services** package contains services that deal with all the logic for each business operation that needs to be carried out (e.g. registering a new account). The **Domain Model** is acting as a set of “property bags” that hold the information coming out of and going into the database. In other words, the **Domain Model** has a state, but no behaviour. An alternative to this would be a **Domain Model** that in addition to
state (i.e. data) it also has behaviour (i.e. functionality). This would make the Services package simpler, acting as an orchestrator for a Domain Model that knows how to perform business functions on its own. The latter approach offers higher re-usability since the business objects (the objects within the Domain Model package) contain all the related functionality.

![Figure 4.27: Business logic package, services components.](image)

For example, in the model shown in Figure 4.21 we see that an Employee entity can have one or more Role entities associated with it as part of the RBAC system requested. In the “Anemic Domain Model” approach we follow, every-time we need to check if an employee has a specific role, we need to retrieve that specific role and check if it is contained into the list of roles of that employee. In the latter approach, we could just ask the Employee entity itself if it has that role and it would handle internally everything that needs to be done to check that. It is obvious that a lot of repetitive code could be avoided in that case. The reason we followed the “anemic” approach is because we wanted to have a more straightforward mapping from the specified activity diagrams to the business functions. This way changes in the model will only fire a change in one business function. For the early stages of the system the “anemic” approach works well enough, but as the system matures a shift to the latter should be favored to make maintenance easier and promote re-usability within the business domain.

The Abstraction Layer also has two major packages within it. One is the Data Access Abstraction package and the other is the External Services Abstraction pack-
Chapter 4. Analysis & Design

As seen in Figure 4.28. The Data Access Abstraction follows the repository design pattern for data access. This means that from the point of view of the business logic, to access data in the database, we expose one repository for each entity available. This repository interface allows the business logic to request specific entities. For example, request an employee named George using the Employee Repository. The major benefit here is that the business logic only sees a repository. The actual data can be stored in a database, in files or on any other storage medium and the business logic will never have to know. This separation is what will allow CV to experiment with different database technologies or storage schemes, if they so desire.

Figure 4.28: High level architecture, abstraction layer.

The External Service Abstraction package, contains all the services the business logic needs to carry out its business functions. The reason we put them in the abstraction layer is because we want to create a platform. Mail services, PDF generation services, Exchange rate services and others, as seen in Figure 4.28, might be used by many different business processes and even different applications that the business hasn’t built yet. To facilitate re-usability of those services we abstract them away through an interface and we make them available to any business function that might need them. This idea was taken from [20, ch. 1.3] that refers to such services as “Application Integration Services”.

In Figure 4.29, we explicitly see the dependency between CVS Core and Genny subsystems. The Genny Service of CVS Core is the service that makes available all the functionality Genny implements, to the business logic of CVS Core. Genny itself has a set of platform services that its business logic utilizes to provide its services to CVS Core (or any other clients that might use its provided interface). A great benefit of this approach is that if a service is deemed I/O demanding or CPU intensive, it can always be separated and deployed as a standalone service on a different server (micro-service) allowing us to deal with scalability issues more efficiently. That is exactly how Genny was born.

The Dependency Layer contains similar packages to the Abstraction Layer as seen in Figure 4.30. The Dependency Layer contains concrete implementations of the abstractions defined in the Abstraction Layer. That is the implementation of the repository pattern for data access and the implementation of the APIs (or the
services themselves, if they are not external) for each service defined.

Finally, the Common Layer contains artifacts such as settings and other configurations for the system to run as expected. Usually this layer contains globally available, hence the dependencies from each other layer, information and technology specific configurations.

This concludes the description of our solution architecture for the problem described in the previous chapter. For a complete model of the architecture, as always, see [18].

4.4.2 System Integration Schemes

An important aspect of any enterprise system is integration. How the services in the Dependency Layer communicate with each other in case they are deployed on different servers and how do we communicate with third party services? To facilitate easier integrations for our architecture we define two integration schemes. One that uses an API for integration and one that uses shared storage for integration.

Service Exposure Integration Scheme

Integrating between two services by exposing an API is a quite common approach and we have already seen it with CVS Core External Services. Figure 4.31 shows
two services being integrated using this scheme. One service is exposing an API and the other is consuming it as an external service. The communication here is unidirectional and if we wish to communicate both ways *Instance 2* should also expose an API for *Instance 1* to consume. In case CVS is consuming an external service the API should be provided. In case we are integrating two micro-services within CVS the interface has to be defined. CVS Core and Genny communicate using this scheme.
Shared Storage Integration Scheme

Integrating between two services using shared storage is a less common approach for integration, but we use it here to allow CVS Core to scale easily. Figure 4.32 shows two services being integrated using this scheme. Both services share a common storage location where each can write and read data to or from. Another way to look at it is to compare it to the producer/consumer problem in software engineering [21]. One service is writing data and the other is consuming it and acting accordingly. In our context, we use this scheme to share session data among multiple CVS Core instances. This way we can run a cluster of CVS Core instances to serve the anticipated user traffic. If a server crashes, another will pick up the requests without loss of information since the session is externalized and shared via a shared data store. This allows CVS Core to scale easily, leaving as the only problem the bottleneck of the shared storage itself and possibly Genny.

Figure 4.32: Integration approaches, shared storage scheme.
Chapter 5
Implementation

In the previous chapter we analyzed the problem we are trying to solve and designed a solution for it. In this chapter, we are going to implement the designed solution by selecting a set of frameworks and technologies for it. We move from the PIM to the PSM. First, we select the technologies for each of our subsystems and then we discuss some important implementation details for each. The implemented statistical process (NewBiz product) is also discussed here as well as possible deployment schemes.

5.1 Software Stack
The software technologies used for the implementation of each subsystem of our designed solution were selected with scalability, maintainability and ease of team management in mind. Whenever frameworks were available that could deliver what was needed, they were preferred over a custom solution. Another aspect that was taken under consideration is the available documentation and support of the frameworks used. In general, we tried to use frameworks and technologies that are popular so that documentation and support will be available when needed.

5.1.1 CVS SPA Software Stack
For the implementation of the front end part of the system, we chose the *AngularJS* framework by Google as our core framework, which is a Model - View - Whatever (MVW) framework that we used as an MVVMC framework, as needed by our architecture shown in Figure 4.22. *AngularJS* [22] is a pure JavaScript framework that allows us to build SPA web applications in a component based fashion (AngularJS modules) allowing for higher maintainability and easier development. In addition, *AngularJS* offers built in management for the dependency between View and View Model so that both are kept in sync when the information change on one or the other. The CVS SPA ecosystem is shown in Figure 5.1 depicting the frameworks used for our implementation and the additional tools utilized to assist with the development efforts.

*RequireJS* is “a JavaScript file and module loader”, as specified in [23], that allows us to dynamically load JavaScript files in the browser. We utilize it in our implementation to perform dependency management since we are using a lot of third party libraries and our source code is also split into multiple modules (Figure 4.23)
dependent on one another. Performing dependency management with RequireJS allows us to write clear, modular JavaScript code, focusing on functionality and not worrying about importing dependencies.

CouchPotato is a lazy loader for AngularJS modules [24]. Since a SPA application can have a considerable amount of modules, it is best if we don’t load all modules when the page is first loaded, but load each module when it is needed. This way we reduce the load on our servers and make our application load faster, since it only loads the bare minimum modules to start. Some might argue that splitting our code into so many modules will increase server calls (to load each file) thus making our application slower. CouchPotato helps with this problem since it is loading only the needed files every time. Load time for CVS SPA is in the range of 300 to 500 milliseconds.

Smart Admin is a user interface components toolbox built on top of AngularJS that we used to design the “look’n’feel” of CVS SPA. To avoid dealing with web design issues, Smart Admin provides already made user interface controls like buttons, form fields and many others that can be used to compose the user interfaces defined in Section 4.3.2.

Finally, we use Grunt [25] and Bower [26] as helper tools to assist in our development work. Grunt is a task automation system that we use to automatically package our application for deployment, start our application in test mode and also deploy on a local test server. Bower downloads all third party library dependencies (and their dependencies recursively). As its creators say, it is a package manager for the web. We no longer need to download every third party library we want to use, we simply specify it to Bower along with the required version and Bower will download it for us in our folder structure. Note that Grunt and Bower run on top of NodeJS which should be installed on the development machine. Finally, to allow the CVS SPA development team to work independently, we use a NodeJS module called HTTP-Server that starts a minimal HTTP server so that we can preview
our work. This means that the CVS SPA development team need only have a text editor, CVS SPA source code and NodeJS installed to complete and preview/test any development effort needed for CVS SPA. A clickable demo of CVS SPA can be found in [12] (use any valid email address to log-in).

5.1.2 CVS Core & Genny Software Stack

Both CVS Core and Genny use the same software stack. Both subsystems were built in Java, on top of the Spring framework [27] and use Maven [28] as a build and dependency management system (Figure 5.2). The Spring ecosystem provides an immense collection of enterprise ready libraries that deal with the most common problems of modern enterprises, from security to web-services and data access. The Repository Pattern we designed for our system for data access is readily provided by Spring (specifically Spring Data JPA part of it) requiring minimal development effort to use. In addition, since we are using Spring Data as the implementation of our data access layer, we can use any of the database technologies supported by it with minor changes to our system. This way experimentation is greatly enhanced for CV.

![Figure 5.2: CVS Core and Genny Software Stack.](image)

To use Spring we utilize Spring Boot, a set of Maven configuration files that allow us to set up applications with minimal configuration. Additionally, Spring Boot allows us to create “fat-jars” for deployment. These are usual Java JARs, but they contain everything needed by the application to run, even a servlet container like Apache Tomcat. This way we need not have an application server running on our infrastructure and we can just execute the JAR file on each server instance we wish to deploy it at.

An important aspect of CVS Core and Genny, is data access. In our implementation we used MySQL for our data model persistence and a Redis storage for the externalization of our session data. Access to the relational model is normally performed through the repositories provided by Spring Data JPA as seen in Figure 5.3. Going through all those layers and especially Hibernate (that Spring Data JPA uses), storing and retrieving big amounts of data becomes really slow. For that reason when performance is important (e.g. when storing a million transactions in the database), we take a different approach. We circumvent all the layers and execute native MySQL queries directly to the database. This way we lose database technology independence, but the performance gains are great and in some cases absolutely necessary (for more on performance see Chapter 6).
5.2 Statistical Process Implementation

For testing purposes, we also implemented one product on our platform system so that we can measure its performance. The product we implemented is the NewBiz report the requirements of which can be found in Appendix B.

The manual generation process followed by CV is shown in Figure 4.5 and Figure 5.4 shows the process described and implemented in [8]. In this implementation we approach calculations differently. What was noticed in the implementation of [8] is that the process was ported directly from STATA to Java bringing along some STATA specific operations that could have been omitted. In CVS, Genny is responsible for generating a report and thus performing all statistical analyses. Since transaction data are stored in a MySQL database, Genny performs queries to retrieve intermediate results instead of raw transactions. This way we split the calculation efforts between a server running Genny and a database server. The benefits of this approach are great (as discussed in Chapter 6), although the database server can become a bottleneck since it can be shared among many instances of Genny. Possible ways to overcome these restrictions are discussed in Section 7.2.

The process implemented by Genny is summarized below

1. The sales data file (an example of which can be seen in Appendix D) is parsed for errors.

2. The sales data file is pre-processed to replace currency and country ISO codes (i.e. SE, GR and SEK, EUR) with the corresponding database IDs. Due to the fact that specific currencies and countries will be repeated many times in a sales
data file, we can retrieve an ID once and cache it to reduce database accesses. At the same time transaction amounts are switched to a base currency prior to storing in the database (in our case SEK).

3. Once the file has no currency and country ISO codes and all transaction are in SEK, we can perform a bulk import using native MySQL queries. This way very big (in the range of millions) sales data sets can be stored in the database within seconds.

4. When a generation is requested, Genny performs a series of native MySQL queries to retrieve statistical data from the database.

5. Incomplete statistical calculations are completed in Genny.

6. The retrieved data are converted to the appropriate currency (selected by the user).

7. The graphs for the report are generated.

8. The PDF of the report is generated in the appropriate language (selected by the user).

The above eight steps replaced the current process in CV and where implemented by Genny with the assistance of the MySQL database server. In the next section we are going to see possible deployment strategies for development and production usage that show how the implemented systems can be used.

5.3 System Deployment Schemes

Now that we have described the implementation of our systems, it is time to see how to deploy them for testing and benchmarking. The next two sections (Section 5.3.1 and Section 5.3.2) describe two possible deployments, one for development purposes and one for production usage.

5.3.1 Development Deployment Scheme

The development deployment scheme is the deployment used for the development, testing and evaluation of the system. As seen in Figure 5.5, the whole deployment is running on a single windows machine to ease development for the development team. The list below describes the different components of the deployment.

- **Host Machine.** A single Windows 7 machine is executing the whole deployment.

- **Redis Server.** An instance of a Redis Server is deployed on an Ubuntu Server instance running within Oracle’s Virtual Box virtual machine. This key value storage is used for all the CVS Core instances to store session data in order to avoid session data loss in case a CVS Core instance crashes.
• **Mail Server.** Within the Ubuntu Server instance we deploy a Postfix mail server to handle the emailing of our application. Note here that for proper email delivery a fully qualified domain name might be required for the deployment otherwise receiving mail servers will block emails sent from CVS Core. For that reason, Google’s gmail was also used as a mail server during testing to ensure proper operation.

• **MySQL Database Server.** Within the Windows 7 host a MySQL database server is deployed to hold the data model of our application.

• **Web Server.** Nginx is deployed on the Windows 7 host to act as a web server for static content (CVS SPA), a load balancer to load balance between the different CVS Core instances and also as a SSL termination point.

• **CVS Core Instances.** Within the Windows 7 host, multiple instances of CVS Core can be deployed to handle requests from users. In our development deployment we had four instances running.

• **Genny Instance.** A single Genny instance is deployed within the Windows 7 host to handle report generations for the four CVS Core instances deployed.

Any browser can be used to send requests to the deployed development system and test its proper operation. In our example deployment Chrome was used.

It should be noted here that the AccSys component is a test system implemented to play the role of an accounting system. The only functionality it has is to log payments and refunds of invoices generated by CVS Core. The reason we implemented this system is to showcase how the same architecture can be implemented using different technologies. AccSys follows the same architecture as CVS Core and Genny.
5.3. System Deployment Schemes

5.3.1 Integration Scheme

The integration scheme used to integrate CVS Core and AccSys is the Service Exposure integration scheme described in Section 4.4.2.

5.3.2 Production Deployment Scheme

A production deployment of a distributed system like CVS requires multiple servers in order to receive all the benefits of the distributed architecture. Although the resources were not available to actually deploy and test the system in a distributed environment, we present here a possible distributed deployment for the sake of completeness. In the scheme presented here we utilize a minimum of six servers.

As seen in Figure 5.6, our system’s “frontier” is the web server. We use Nginx as a static content server (serving CVS SPA web client files), an SSL termination point and a load balancer. All requests received are encrypted (HTTPS) and once inside our deployment (data center) they are plain HTTP. Behind the Nginx web server lies a set of CVS Core instances that handle business functions. CVS Core instances use a Postfix mail server to send emails to users and a Redis server to externalize session data. The Redis deployment can also be a Redis cluster for better performance and data redundancy. The CVS Core cluster is using a MySQL database server to store business information. The MySQL server could also be clustered for better performance. Finally, another load balancing layer is used to load balance among the different Genny instances when requests are received from the CVS Core cluster. The Genny cluster also makes use of the MySQL database server to retrieve and manipulate transaction data.

![Deployment Diagram]

Figure 5.6: Deployment schemes, production deployment.
So far we have described our problem, the analysis we performed, the design of the system, its implementation and some possible deployment scheme. The final chapters of this document will discuss the performance aspects of the system and the benefits the software development process followed has.
Chapter 6

Evaluation

In this chapter, we are going to evaluate the performance of the new architecture we implemented in comparison with the system developed in [8]. What this chapter will attempt to do is collect data that will allow us to decide whether the new version of the system shows a better performance than the previous one.

6.1 Experimental Testbed

The experimental testbed used for the performance evaluation of CVS has the specs shown in Table 6.1. Since the server used to evaluate the system in [8] was different than the one used here, we will run the tests for both implementations on our server so that our data come from the exact same infrastructure, using the exact same input data.

<table>
<thead>
<tr>
<th>Component</th>
<th>Test System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>8GB DDR3</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel Core i5 2500 @ 3.30GHz</td>
</tr>
<tr>
<td>OS</td>
<td>Windows 7 Professional, 64bit, SP 1</td>
</tr>
<tr>
<td>HDD</td>
<td>Samsung Evo 256GB SSD</td>
</tr>
</tbody>
</table>

Table 6.1: Test server specifications.

For our tests we used real transaction data provided by CVs customers in the sizes of 1000, 10000, 100000, 200000 and 400000 transactions.

6.2 Performance Profiling

To evaluate the performance of the system, we performed a series of ten (10) NewBiz report generations for each available data set size on both systems and measured the time needed to complete the generation of each report. The generation time was measured at the code level by measuring system time before the generation starts and after the generation is complete. As we can see, as the number of transactions submitted for analysis increases, the time required for analysis also increases. Although this does not seem to be scalable, the mean time required for 400000 transactions has improved quite considerably from almost 3 hours in [8], to 368107ms in CVS. That is a 29-fold increase in performance in the largest data set available.
Figure 6.1 shows how generation time increases for both systems as the data set size increases, but in Genny that happens a lot slower than in [8].

![NewBiz Report Generation Time](image)

**Figure 6.1: NewBiz report generation times.**

Table 6.2 shows the mean time for the generation of each report as well as the standard deviation of the measurements. Note that 6.1 shows no variance bars because variance is too small to be visible on the chart.

<table>
<thead>
<tr>
<th>Data Set Size</th>
<th>CVS Mean</th>
<th>CVS SD</th>
<th>[8] Mean</th>
<th>[8] SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>516.50</td>
<td>118.59</td>
<td>3475.20</td>
<td>294.49</td>
</tr>
<tr>
<td>10000</td>
<td>2579.90</td>
<td>153.85</td>
<td>177760.80</td>
<td>1689.41</td>
</tr>
<tr>
<td>100000</td>
<td>93445.20</td>
<td>425.10</td>
<td>2284850.90</td>
<td>4171.10</td>
</tr>
<tr>
<td>200000</td>
<td>180821.30</td>
<td>440.51</td>
<td>6845338.10</td>
<td>4468.14</td>
</tr>
<tr>
<td>400000</td>
<td>368107.00</td>
<td>492.63</td>
<td>10846322.00</td>
<td>5147.38</td>
</tr>
</tbody>
</table>

**Table 6.2: Mean generation times and standard deviation for each system (in ms).**

Since the majority of the calculations are executed on the MySQL database server as queries on a table that stores transactions, we also want to know how the presence of other data in that table affects generation times. To measure this we enter 10k transactions in the database to be used for the report generation and then we progressively add more transactions (that are not used in the analysis process) just to measure the effect the database table size has on report generation time. Figure 6.2 shows how the analysis time for 10k transactions is increased just by the presence of other transactions in the table. As the additional transactions in the table reach 400k the analysis time almost doubles. The experimental data show a linear increase in analysis time with an increasing table size. That is $O(n)$ time complexity over the size of the database table.

This could be a serious limitation of the system since many clients of CV have large data sets to upload. As we discuss in Section 7.2.4, there are ways to reduce
the effect this has on performance, but this is probably the Achilles heel of the proposed architecture and a certain place where CV should experiment more on.

Before closing this section, we perform some simultaneous generations and see how Genny behaves in such cases. We run one, two, five, ten and fifty simultaneous NewBiz report generations for 10k transactions in the data set being analyzed and 1 million transactions present in the transactions database table in order to take the table size effect into account for a more realistic view. Figure 6.3 shows how generation time increases as more generations are performed at the same time by a single Genny node.
It is obvious that the mean generation time increases quite considerably as simultaneous generations are executed, which makes necessary the existence of a barrier to restrict the incoming requests for a busy Genny server before the database server runs out of memory or the Genny server takes an unacceptable amount of time to complete a generation. A way to achieve this is discussed in Section 7.2.

6.3 Use Case Testing

To test that we indeed fulfilled the requirements we elicited and that the system is visually pleasing and as expected by CV management, we executed all the use cases defined in our use case model. The tests were run on our test deployment and the process was captured on a video that was sent to CV for review. The video can be found in [13] and below follows a part of the response from CV’s senior manager Mr. Ove Svenneke.

“After having watched your demo I am very impressed indeed! Once again you have shown that you are building the system on the foundation of a perfect comprehension of our aims and needs. And this demo just oozes system quality! Very good demo in itself as well.”

6.4 Evaluation Interviews

In addition to the quantitative evaluation of the system, we also performed a series of interviews with senior developers and managers from Tickethour SA, in an attempt to identify the major problems a development team faces during system development and see whether our solution covers them to some extend. The interviews were performed at Tickethour SA offices in Greece and six senior developers, the IT manager and the CTO took part in the process. The notes taken during the interviews are available in Appendix E.

Having interviewed the Tickethour SA development team as well as their CTO and IT Manager, we narrow down the problems discussed in three major categories. Pure managerial problems, technological problems and people problems. The list below summarizes the problems presented by the developers and managers.

- **Managerial Issues**
  - Upper management does not invest in its people (i.e. training).
  - Many meaningless meetings take place that waste time.
  - Upper management makes decisions based on short term monetary benefits, missing the long term benefits an investment in a formal process could bring.
  - Development team hierarchy is not clearly defined and roles are not well assigned, thus leading to bad team management and leadership.
  - The organization depends on people and not on processes giving rise to politics (i.e. influencing for one’s own benefit) within its premises.

- **People Issues**
Developers work without processes so long that get used to this way of working as the proper way.

Communication issues result in phases taking more time than expected (usually the requirements elicitation phase) and business requirements being improperly translated to system requirements.

Developers lack important knowledge which leads to ability overestimation (which can lead to invalid schedule estimations), fear of new technologies and if combined with lack of good design fear of change within the team.

Tasks are not being assigned based on experience and specialization of developers with an approach of “everyone can do everything” being followed were any task can be assigned to any developer.

Inaccurate time estimations result in hard to follow schedules which then lead to long working hours overusing human resources. This easily leads to lower motivation and developers transitioning from being committed and performing at their best, to being just compliant where they do the barely minimal work required by the business. This is a major loss for a company.

High learning curves for new members joining the team due to absent documentation.

Missing connection between IT and business (i.e. a business analyst).

- **Technological Issues**
  - Poor system requirements analysis as a result of bad communication (see previous category).
  - Poor technologies selection due to bad analysis of the system.
  - Lack of formal testing and a dedicated test engineer.
  - Lack of automation in development operations.
  - Lack of self evaluation and improvement processes.
  - No focus on quality.

The proposed process for Helium, does not deal with pure managerial issues although it attempts to reduce the dependency of the organization to people and increase its dependency on processes. Regarding people and technical issues, the Helium process, attempts to reduce communication overheads through the introduction of collaboration tools that facilitate better communication and the use of modeling that provides a form of documentation for the system and its requirements. This way requirements are clear as early as possible reducing recurring communication efforts that cost time and money. In addition, since requirements rarely stay the same during the complete project execution, we assume that change is managed gracefully in Helium through model assisted impact analysis. Having a modular and highly decoupled architecture, we assume that time estimations can be made more accurate and thus project work can be scheduled in a more pragmatic way. Although the Helium process is not a complete process for a for-profit organization, it does provide some basic tools that attempt to solve some of the most basic, yet
important, problems faced by software development teams. As discussed in Section 1.3, the aforementioned benefits of the proposed process cannot be proven in the context of this thesis since repeated application and evaluation over a long period of time would be required to reach any statistically significant result.
Chapter 7

Conclusions

The final chapter of this document presents a discussion regarding the goals we achieved and the system in general (Section 7.1) and how should the company proceed in the future (Section 7.2) to achieve their strategic goals as smoothly as possible.

7.1 Conclusion

The Helium project proposed a customised software development process for CV and then followed that process to develop a distributed system that covers CV’s specific needs. One of the main goals of that system, as discussed in Section 1.3, was to prove the assumption that a distributed architecture utilizing a database for transaction storage could enhance performance compared to the implementation found in [8]. The additional goals set for this project can be found in Table 1.1.

As shown in Figure 6.1 our main assumption has been proven and Genny is actually faster than the system found in [8]. A reduction in generation time by a factor of at least XXX was found in the largest data set available, thus proving that moving away from files and using a database to store and retrieve transactions can indeed increase the performance of the system. In addition, due to the increased performance, some other benefits are expected to arise although some cannot be presently proven scientifically. The capacity of the company has increased since it can now serve more customers in the same amount of time (i.e. as proven in [8]). Using the automated process instead of the manual process, we expect to have less human errors and lower operational costs since there is no need for heavy staffing to cope with high demand. The distributed architecture used (i.e. microservices), allows for easier experimentation, since the system is made up of a set of highly decoupled subsystems that perform its major functions, and more efficient scalability since the different subsystems can scale independently. Regarding CVS’s maintainability and extensibility, the decomposition chosen (Section 4.4.1), the proposed integration patterns (Section 4.4.2) along with the models demanded by the Helium software engineering process (Section 4.1), help in both directions, since the system is highly documented through its models, allowing for easier bug tracking and specific design patterns are used to extend its functionality. Finally, the UX has been improved by using a mobile friendly, well designed set of controls to build our user interfaces as seen in [12].

In total, our main hypothesis has been proven and a set of different aspects of
the system were discussed to spur further research regarding the system architecture and the software development process followed.

7.2 Future Work

In this final section, we discuss the future steps CV could take to further develop CVS into a production ready system. The following sections are in order of importance starting with the most important tasks.

7.2.1 Leftovers

During the development of CVS, the main goal was to bring the system to a stable enough state that we can measure its performance. During development, due to time restrictions, there were two units of work that were left out and need to be implemented in order to bring CVS closer to a production state.

Firstly, a conversion of all incoming transactions to a base currency should be implemented to keep all database transactions in the same currency (in our case SEK). This step should be done prior to importing the transactions into the database (or any other technology used) and it should be implemented as part of the pre-processing step described in Section 5.2. A strategy, as far as currency conversions are concerned, is needed by CV to define which exchange rates should be used every time we convert a transaction amount to a different currency. For example, an amount being converted to another currency could use the exchange rate of the date the transaction occurred or the date the transaction was uploaded to the system. Different strategies will have a different impact on system performance so proper evaluation should take place before deciding. Secondly, in the current MySQL implementation we use MySQL Views to filter transactions for a specific company out of the complete transaction set stored in the database. This means that for each new company being created in the system a new MySQL View should be created to filter the transactions for that specific company. The current implementation has the appropriate views for the test data already created in the database. Our source code has been marked in different places with “TODO” comments in order to help future developers add the described functionality.

7.2.2 Formal Documentation of Statistical Processes

The heart of CVS is Genny. This is where the statistical processes defined by the business are implemented and the valuable statistics for its clients are generated. Studying the manual process that CV follows so far, as well as the direct implementation of it found in [8] we see a major issue that needs to be handled as soon as possible before any further development is carried out. The Java implementation of the statistical process in [8] was directly derived from the STATA implementation used in the manual process executed by CV. This means that language specific peculiarities made it through into the new Java implementation. For example, duplicate column data and lots of intermediate results generated in STATA, probably to ease development in that specific environment, were also found in Java were their presence was completely unnecessary. In addition, during the Helium project execution, porting from pure Java [8] to a combination of Java and MySQL poses the same
7.2. Future Work

risks. For that reason, we first tried to understand the statistical process, to the extend possible, and then implement it for CVS. The removal of many unnecessary actions within the process (mainly needed because of STATA) is one of the reasons why CVS saw such a great improvement in its performance.

The underlying problem is that we are trying to go from a PSM (i.e. STATA) to another PSM (i.e. Java) instead of going from a PIM (i.e. statistical process documentation) to a PSM (i.e. Java or Java and MySQL). The absence of a complete and comprehensive documentation of the statistical process is what causes this problem. In other words STATA should be seen as an implementation of a statistical process and not as its documentation. Documenting the statistical processes defined for the different products sold by CV should be a priority to enable parallel development of those from multiple teams (or thesis projects). This way the comparison of performance for different technologies is enabled and CV’s experimentation is greatly enhanced. The main goals of a document like that should be to clearly define the statistical process mathematically and also convey the business thinking behind the process. In other words, convey the “what” we need to calculate and “why” we need to calculate it.

Finally, a test set should be created containing enough real data to perform a complete system analysis. In the case of CVS, the data set used for evaluation was different than the one used in [8] so the need to re-run the tests for both systems was necessary for our results to have any statistical significance. This was acceptable at this point in time were only two implementations are available, but in the future re-testing more than two systems can easily become a time consuming task. A common data set for all implementations is needed so that we can later compare the results and be able to select the most appropriate implementation based on the strategic criteria set by the company.

7.2.3 Development Infrastructure

Equally important to a common data set for evaluation is a common infrastructure which will be as close to a deployment infrastructure as possible. This way all systems will be tested on the same infrastructure without the need to re-test previous systems.

In general, it is suggested that when developing a distributed system, development efforts are tested on an identical to production deployment [15]. The downside of this is that this will incur a cost for CV to actually lease or purchase this infrastructure and make it available to their development team or the thesis project students. Although it is an extra cost, it is definitely a worthwhile investment to promote a valid comparison between the different technologies tested for CVS.

7.2.4 MySQL Database Sharding

Utilizing a MySQL database to store all transaction data has a weak point as shown in Figure 6.2. The $O(n)$ time complexity over the size of the database table means that as more and more transactions are stored into the database (on a monthly or otherwise basis), the system will become progressively slower. A way to lower the effect this phenomenon has on performance, is to apply a sharding scheme to the database. Sharding is a partitioning technique were one database table is split among
many different database servers according to specific criteria. This way instead of having a single database server that stores all the transactions in the system, we have multiple database servers each holding a part of the total transaction set. A possible sharding scheme could be to assign specific companies to specific database servers. This way, each company’s transactions will be on the same server (to avoid the slow cross-server queries) and each transaction table will grow slower than if it stored all the transactions in the system. A drawback of sharding is the increased cost to maintain multiple database servers for a single table. If sharding increases costs unacceptably compared to the performance gains, then this would be an indicator to try a different database technology for Genny. Other approaches for this problem could be to progressively remove old transaction data, to keep the table size under control or a combination of this approach with sharding.

7.2.5 Evaluate Different Database Technologies

One of the reasons why we applied a microservices architecture to CVS is to allow further experimentation to be easier for CV. CVS Core is stable and covers the requirements as specified. Genny on the other hand, as discussed in a previous section (7.2.4), has some issues that should be dealt with. The combination of Java and MySQL to perform the statistical analysis proved to be quite beneficial, but the problem shown in Figure 6.2 posses a threat to CV’s scalability. Apart from sharding (discussed in Section 7.2.4) using a different database technology might prove to be a solution to this problem while maintaining high performance. Because we used Spring Data for our data access layer, switching to other database technologies supported by Spring Data will be fairly easy.

7.2.6 File Management

Uploading files through a form (as we do now in CVS SPA) is a good enough solution for a small number of files being uploaded monthly, but as CV’s customer base grows in size, file uploads will become an issue especially if many of them happen at the same time. If the system is stress tested and the number of uploads that it can handle proves to be inadequate, a third party service could be used instead. Amazon S3 is a possible choice. CVS SPA could upload the files directly to Amazon S3 and then provide CVS Core with an ID so that the file can be retrieved when necessary. This way we can avoid overloading the servers, but the files will still need to be downloaded to the data center and into the database. Another possible third-party provider of such services is TransferBigFiles. A rather funny, yet quite informative forum conversation on this matter can be found in [29].

7.2.7 Queuing for Generation Services

Figure 6.3 shows how Genny responds when multiple report generations are requested at the same time. In the current implementation, Genny will accept any number of report generations requested. Since the mean time needed for each generation increases as more and more generations are requested, we risk an unacceptable increase in generation time as well as Genny running out of memory. The same memory problem is true for the MySQL database server, especially if we consider the fact
that multiple Genny instances might be sharing a single MySQL server instance. A way around this problem would be to use a queuing system for our Genny instances. This way, all requests for report generations coming from the CVS Core instance (or cluster of instances) will get in a queue and each Genny instance will simultaneously serve only the number of reports that its memory allows and that it can generate in an acceptable amount of time. Since CVS was built on top of the *Spring* framework, *Spring AMQP* could be used to implement the aforementioned queuing system.
Appendix A

Full System Impact Analysis - Account Registration

The following diagram is an impact analysis diagram that spans the whole system, from the requirements model to the data model. Here you can see all the components of the system that interact to perform the account registration business function.
Appendix B

NewBiz Report FRD
Customer Value Sweden AB

Customer Value Production system

Functional Requirements Document

Author Customer Value /Ove Svenneke

Current Date 27 August 15
Last Edited Date 2014-01-24

Version Number 1.0
Document Status Final
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Version history</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>About this Document</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Purpose</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>Prerequisites</td>
<td>3</td>
</tr>
<tr>
<td>2.3</td>
<td>Target audience</td>
<td>4</td>
</tr>
<tr>
<td>2.4</td>
<td>Related Documents</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Functional Areas</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Scalability</td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Market (country) localization</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>Products</td>
<td></td>
</tr>
<tr>
<td>3.2.1</td>
<td>Customer Value New Biz</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Back Office (BO) data</td>
<td></td>
</tr>
<tr>
<td>3.3.1</td>
<td>Customers</td>
<td></td>
</tr>
<tr>
<td>3.3.2</td>
<td>Customer Files</td>
<td></td>
</tr>
<tr>
<td>3.3.3</td>
<td>Report Files</td>
<td></td>
</tr>
<tr>
<td>3.3.3.4</td>
<td>Return Reports</td>
<td></td>
</tr>
<tr>
<td>3.3.3.8</td>
<td>Usage</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>Back Office interface (BO) CV internal</td>
<td></td>
</tr>
<tr>
<td>3.4.1</td>
<td>Customer Portal - Admin</td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Customer Portal</td>
<td></td>
</tr>
<tr>
<td>3.5.1</td>
<td>Customer Portal interface</td>
<td></td>
</tr>
<tr>
<td>3.5.2</td>
<td>Customer Portal - admin page</td>
<td></td>
</tr>
<tr>
<td>3.5.3</td>
<td>Basic payment page</td>
<td></td>
</tr>
<tr>
<td>3.5.4</td>
<td>Up-/download page</td>
<td></td>
</tr>
<tr>
<td>3.5.5</td>
<td>File handling</td>
<td></td>
</tr>
<tr>
<td>3.5.6</td>
<td>File manual process</td>
<td></td>
</tr>
<tr>
<td>3.5.7</td>
<td>File columns mapping</td>
<td></td>
</tr>
<tr>
<td>3.5.8</td>
<td>Marketing costs</td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>File upload automatic process</td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>CV New Biz Report Production</td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>Security</td>
<td></td>
</tr>
<tr>
<td>3.8.1</td>
<td>Certificates</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Success Criteria</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Sign Off</td>
<td></td>
</tr>
<tr>
<td>Appendix A – BO (internal) Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appendix B.1 - File Specification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appendix B.1 - Customer file example</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appendix C - Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appendix E - Product language versions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Appendix F - pdf example of the New Biz product</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1 Version history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>By</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>2013-03-19</td>
<td>OS</td>
<td>Draft</td>
</tr>
<tr>
<td>0.4</td>
<td>2013-12-31</td>
<td>OS</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>2014-01-02</td>
<td>OS</td>
<td>Draft to be reviewed by CV staff</td>
</tr>
<tr>
<td>0.6.1</td>
<td>2014-01-11</td>
<td>OS</td>
<td>Final to be reviewed by developer</td>
</tr>
<tr>
<td>1.0</td>
<td>2014-02-18</td>
<td>OS</td>
<td>Final, but will be improved upon request. Explanatory files added separately, explained in 3.7. User rights further described in app A. Success criteria added in section 5. Security section added but to be completed; section 3.8.</td>
</tr>
</tbody>
</table>

2 About this Document

In this document

- "CV" is short for Customer Value (our brand) or Customer Value Sweden AB (the company),
- "customer" refers to customers/clients to CV,
- "consumer" refers to customers to the CV customer.
- "BO" refers to Back Office which is where CV staff and/or the system will store and administrate customer information and customer files.

Blue text indicates functions we do not expect to be included in this Ex-job project. They are included as indication of how we expect the system to be developed further in the future. The reason for this inclusion is that we appreciate if the system is build in a way not making such expansion difficult. If any of these functions can be included we absolutely don't mind! But focus should be on developing a stable system that can become a scalable basis for further enhancements.

2.1 Purpose

This Functional Requirement Document is designed to outline each functional area of the project, detailing its logical requirements. It is however not a technical specification requiring methods how the system should be built.

The aim is to build a system to produce report products (now one product only) in an automatic way for e-trade companies. These are analyzes of consumer purchase patterns/behavior supporting the marketing organisations of our customers evaluating market investments and improving customer loyalty. The reports also are used to evaluate the business performance of our customers in conjunction eg with new capitalization processes.

2.2 Prerequisites

Choosing between including nice-to-have-functions and developing a stable and future proof system focus on the latter.
We need a working system giving basic functions with these prioritized

1. Basic database to handle
   1. CV Customer Input Files
   2. CV Customer Information
   3. CV New Biz reports produced

2. Producing, using the input file and basic customer info settings, the Customer Value New Biz Report (included as an Appendix with this FRD).

3. Making the report available for download by CV staff or to be sent via e-mail to the customer.

Dependent of the time resources that can be given to this project a customer online interface "Customer Portal" such a web interface can be included as described with texts in blue below. But, still, priority is on the above. It is however a requirement to build the system in a way making a web customer interface easy to add.

The CV Customer Info should be possible, in the future, to expand with basic customer relations functions where CV customer interactions (as purchases, inquiries etc) can be handled via the web online interface both by CV staff and customer users. The latter is not described in this document.

Note that language version means that the entire customer interface and - in the future - the products produced shall appear in a local language. The Back Office / administrative routines shall always be in English. This means that the system now discussed shall have an English Back Office and a customer interface (if developed) in Swedish.

Dependent of tools preferred we suggest using well documented open source, as Linux, when applicable.

Note this is a functional description not instructions for exactly how to build the data tables etc.

How the system should be documented we assume is specified by Ex-Job instructions and thus not covered in this document.

2.3 Target audience

The target audience for this FRD document is programmer(-s) that have chosen this project for their KTH Ex-job and any KTH staff involved in the project.

This document and all additional information given by CV is confidential.

2.4 Related Documents

Customer Value New Biz report - example/demo (in Swedish).
Customer Value Report texts (txt document).
Customer Interface texts (to be added if this will be developed).
3 Functional Areas
The aim of the system is to
- Administering CV Product settings (Appendix A Products)
- Administering CV Customer
  - Basic information (Appendix A Customers; Customer Settings; Users)
  - Products subscribed (Appendix A Orders)
  - Files uploaded (to be specified)
  - Files ready (to be specified)
- Via a Back Office system (BO)
  a. make it possible for CV staff or e-trade (customer) users to upload extracts with consumer purchase information retrieved from accounting system of a specific customer.
     "Customer File"
  b. produce an actual report including graphs and tables with standard texts (these are identical for all reports of a type produced) included.
     "New Biz Report". Note: this is not distributed to customers.
  c. make it possible for CV staff to download the produced report pdf:s for manual distribution to customers - and if developer time allows for the customers themselves to download their reports.
     "New Biz Report pdf". This is distributed to customers.
  d. [make it possible to download a Return File including produced consumer details reported back the customer. This is not a feature of the New Biz report however and thus only mentioned here as the system in the future needs to have this functionality.
     "Result File".]

User and internal actions shall be logged. This can be done in a simple way with a text string record containing the new, changed or deleted information that has been entered. Shall contain date/time and be sorted by date/time. The log shall be restarted in a new text file every [. ...]. It should be saved to be accessible to CV staff via a simple Admin BO interface.

3.1 Scalability

3.1.1 Market (country) localization
In the present solution the system shall be used for the one market only (Sweden). (But we plan to within reasonable time to expand operations to other markets.)

Note however that cross-border selling is normal within e-trade. This means that transactions may occur in different currencies although the e-trade service is physically in (for example) Sweden.

We need a system that can fairly easy be expanded to different markets. This means we will need to integrate functions for

1. Countries.
   Note that a CV customer located in one country (or more) may have it’s customers = consumers in several countries.
   - Identifying geographical Markets. This is as we expect customers to want "markets" constructed for them including more than one country. Eg BeNeLux or Greece and Cyprus. This is however nothing that needs to be considered in this
version. If it is possible to opt for a solution that a further development of the system in this direction is not hindered this is appreciated.

2. Languages

- The CV internal system should work in English only.
  - Customer Portal interface should however allow us to set up to 3 languages to be used in a specific Market.
- Products are produced in Swedish. We shall add an English version. And - future - make it possible to expand to several Product languages. (Compare Appendix D).

3. Currencies

Single report

Normally a cross-border selling customer will want a report to cover all markets in one single report where any non Main Currency (as set in BO) used should be converted to the main currency according to Sub-Market settings in BO (Appendix A). This conversion is made using the Currency data in the input file. We shall require customers to use ISO 4217 codes. The BO "Main Language" language product version should be used.

The report language should be "Main Language". If a report versions in "Main Language" is not available use English.

However we most probably will see customers using non standard currency codes. This is indicated in BO with "Input currency code system" = "bespoke". If "bespoke" use the customers BO "Bespoke Currency System Conversion table" (Appendix A) to retrieve the ISO codes to be used by system.

Separate editions

Companies shall also have the option to chose the report to be produced in separate editions for separate markets as indicated by BO "Country Separation" = "on". In these cases a complete report shall be produced covering all cross-border buying consumers in non "Home Market" country (setting in BO) using "Main Currency". The countries chosen by BO settings shall also be reported in separate reports using the "Main Currency" and in report language = English¹.

Report languages for sub-markets

By standard English.

Perhaps we allow users to choose preferred report language for the by market where our products will be sold. For example: In Belgium the user should be able to choose between English, Dutch, French or German.

We need to include information and texts in different languages to be used in market adaptations in the online interface and the products. Note that the same language can be used in several countries. Eg french can be used for the French and Belgian markets.)

3.2 Products

It should be feasible to add products and product versions² the customer may choose (basic, standard and advanced for example).

---

¹ The idea is that customers working in different markets will probably use English as their company language. Companies working in a "Main" market only may however prefer the report in their local language ("Main Language").

² At present we have two products: Customer Value New Biz and Customer Value Report. As the latter requires more complicated interaction with statistical analysis tools we have chosen the New Biz report for this project. It should be possible however to expand the system to handle also the CV Report product and other report products added to our product portfolio in the future.
Each product should be possible to produce in different languages. Compare Appendix E.

In this system version we will at the beginning handle one product - Customer Value New Biz Swedish version - only. But the aim is to add other products and in different language versions.

3.2.1 Customer Value New Biz
This product is best understood examining the

- the code now used to produce the input for graphs and tables to be included in the final product. Appendix C.
- the logic used now in MS Excel (enclosed / Appendix D)
- An example of a final report (pdf) Appendix E

Please request any explanations and further details needed!

3.3 Back Office (BO) data
We need to keep data about customers, files and products produced saved in an CV internal Back Office system.

We need to

1. store Customer Files. This is from where the system retrieve data input.
2. store New Biz Reports and New Biz Report pdfs. This is from where customers downloads products. (They shall be able to reload files if they need to.)
3. store Return Files

(1) will store updated information only, that is no historical information shall be saved.

(2) - (3) will store all files received and produced until cleared. We will thus need a function to clear old data (manual and/or automatic). This can however be developed later. Data purge will be made by Admin only.

3.3.1 Customers
BO TABLE identifying customers and users + settings. See Appendix A.

3.3.2 Customer Files
This is where the files uploaded by customers with data used by the productions part of the system is saved. We will save historic files. Manual or rule for file deletion can be added later.

3.3.3 Report Files
This is where produces files are saved. Best would be if the raw file\(^3\) can be saved.

We need anyway to save pdfs making it possible for a customer to retrieve the reports.

Manual or rule for file deletion can be added later.

3.2.4 Return Reports
These are customer files with data produced by the analysis included in the production routine for Customer Value Report (and possible other reports). This is not a part of this system version as the New Biz Report does not include Return Files. These files are in text format.

---

\(^3\) If such a file will be needed by the system.
3.3.3 Usage
The system shall keep track of customer usage. See USAGE TABLE in Appendix A. In the long run this part has two purposes

- Inform CV staff and customer via Customer Portal about purchases and usage
  - Transfer information needed for invoicing to a CV accounting system [not needed now].

3.4 Back Office interface (BO) CV internal
This is available to CV staff only.

The purpose to keep BO data updated. And to administrate uploaded files and reports produced by the system. There should be two ways to update the number of reports bought - by Customer Portal or internal BO.

Note. Products used should be compared to Products bought and Products free.

When adding "new" products to be made the value bought (or free) will be added. The value will thus show the total number of products the customer has bought (got free) from start.

(Deduct should be from free first hand; that should not need to be considered (?) as the system will prior to producing check that used is > bought + free.

We need a function to add, activate and close account. Accounts should not be erased. Ceased accounts should only be deactivated. They can be reactivated later. Default for a new account is active.

3.4 Customer Portal - Admin
Not included in this system version. Shall make it possible for customers to themselves administrate certain data items and files.

3.5 Customer Portal
We need a web based portal in which customers can open a new account and buy products. A Back Office interface, available to CV staff, is needed from the start. This needs not be developed as a web based system. But the CV internal interface can very well also be in a web https format. This can be a decision of the developer.

The Customer Portal has two sections.

- Register
  - an account
  - order products
- Administration (where customer may change setting, add users etc.) Not developed in this version.

Note. Free reports (as mentioned in this document text and illustrations) will not be possible to register by customers for the New Biz Report. The function will be fully implemented in a coming system version in which we add support for Customer Value Report.

CV staff can change the free reports value in this system version.

Note. To be activated if we expect this to be a problem a maximum number of new accounts should be possible to register via the Customer Portal per hour. This number shall from start be 999 but should be possible to adjust. If a company tries to register when the max total has been reached just give a comment "Systemet tar inte emot registreringar av tekniska skäl, vänta...".
kontakta support@creditsafe.se så hjälper vi dig." / "The system does not accept new accounts for technical reasons. Please contact support@creditsafe.se for assistance.

3.5.1 Customer Portal interface
The Customer Portal is reached from a web page [not included in this project]. The portal needs to be available in a web interface via https.

New account:
Open new account
To open an account the system creates an account ID.

- Account ID shall not be the same as the organisationsnummer\(^4\) as the customer may want to transfer the account to another organisationsnummer in the future.
- The system shall allow one account per organisationsnummer only.

Fields, see Appendix A. Note: this is specified according to Swedish standards.

For new markets we will need to modify the screen layout and content. And use other languages.

\(^4\) The name of the official Company Number in Sweden.
Note. This is just a rough outline. Page name has not been decided.

The user name and pw are selected by the client. This data is saved encrypted. The individual entering this information becomes default a Main contact.

The entered information is displayed on a web page for review and approval.

Confirm mail. A successful entry should be confirmed by email to the Main Contact person including all entered information except pw. A copy of this email should also be sent to administration@customervalue.se.

The confirmation mail shall have a link the main contact is required to use - within 2 hours. When this has been done the account is activated. This is confirmed by a confirmation page with a button "Proceed" / "Fortsätt" stating

"Tack för att du registrerat ditt konto hos Customer Value. Tryck på knappen nedan så kommer du vidare." / "Thanks for registering your account with Customer Value. Press the button below to proceed."

We will need to add text explanations as it becomes clear how the system will function and how far the development in this stage will go.

In this version additional users can be entered by CV staff only.

3.5.2 Customer Portal - admin page

The admin function will add functions to administer users etc. The following is just a rough outline.

- enter new users
- delete users
- change account information
- access (c) charge account
3.5.3 Basic payment page

Add a payment page to be using Paypal or Klarna. Note. It will most probably be necessary to use different payment exchanges in specific countries.

A payment needs to be registered and accepted before the account and order is registered.

Payment checks

First hand for an internal invoicing routine. If we chose to add invoicing to the customer portal this should also be used as customers want to pay by invoice.

The system shall optionally, upon payment procedures, make a credit check via Web Services to a external credit information company giving a yes/no response. If customer is rejected display a message\(^5\). A typical xml response may follow the following schema:

\(^5\) "We are sorry. The account cannot be opened. For further information an assistance please contact Customer Support who will assist you at [market e-mail address] or [market phone number]."
d. Customer Portal

A user can enter the Customer Portal of the client the user is registered at by entering username and pw. Three trials to enter should be allowed. A fourth erroneous entry will close the user account asking the user to get the Main contact to register a new pw or contact Customer Value support/customer service. After three failed attempts to log in a warning to the user should be printed on page.

*Note. Texts will be provided.*

The Customer Portal of a client can be open to several Client users simultaneously.

Any Customer Portal should be accessible to CV Admin.

Language: English. *Localized language version will be needed in the future.*

Any access or tried but failed access to Portals should be logged with:
- date/time
- account accessed
- ip-address and domain if available
- success/failure and reason (incorrect account ID, incorrect pw, system error (down etc), unknown)

### 3.5.4 Up-/download page

From the Customer Portal landing page the Client user should be able to select:
- access to instructions
- information about automatic uploads from accounting systems or e-trade systems
- upload of customer information (see next section)
Customer Value Production system 1.0

- download Reports
- download Result Files

Customer ID SE1234156

Upload your files manually

Download Result Files

Choose File

- SE1234156_201210-201212
- SE1234156_201207-201209
- SE1234156_201204-201206
- SE1234156_201201-201203

The file name should have this format:

Change the file name:

Upload to Customer Value

Cancel
Manual file upload

Customer Registration from home (index) page customervalueonline.se

New analysis
Manual

Retrieve from Accounting System v 2.0

Https upload file page

Browse for file / Upload

File format check

Format test reject message to user

Ok

File saved to customer account in db

Job queue

v 1.0

email new file message to customerservice@customervalue.se

SRS report generating system

Return file Report pdf

send report/file ready mail to customer v 1.0

add link to report

add link to file

manual, from back office v 1.0

customer portal page v1.0
- your reports
- your files

v 2.0

(Note. "SRS" is a title not used any more. 2.0 = blue.)

3.5.5 File handling

We need to set up a file server and/or SFTP server.

Each client shall have a specific area or portfolio to which files are being sent.

The files uploaded by the customer contains transactions from the customers e-trade business, that is purchases / return detail values (date, currency, amount etc). See Appendix B. These are - by the system - to be accumulated to the data needed to produce the report.
There will be two alternatives for clients to send files manual and automatic. This is checked in BO with the File upload setting "manual" or "auto". This is typically an "or" choice. We probably, however, will probably need to allow manual file upload also for "auto" customers.

i) Manual upload.

A web page accessible after customer login shall contain a standard upload routine in which the user can select the file from his local computer and the send it (under https).

We will need a specific file name. To ensure this name to be correct the user will have to fill in a couple of questions. Together with system information this should be used to create the file name. Thus: regardless of the name the user has the file shall be given a new name.

The file name shall
- identify the customer (customer ID)
- the period=month for the data alternatively if this is a basic (complete) file upload (customer entered information on this page; YYMM)
- date & time when received (system)

Files data ("Customer"; "Period"; Date) shall be kept and displayed in the BO under the customer ID together with
- status (see below) "Status" (created by system, see below)
- user ID (the user uploading the file) "User" (as logged in)
- original file name (the name the customer had given the file when uploading) "Original name" (taken from original file name)
- created file name "Created name" (created by system)

Customer files shall be saved in the database [space may become a problem?]. When the file has successfully been saved the file in (any) SFTP-area can be erased.

ii) Automatic upload

Automatic process means that the customer is using an accounting system that has a routine that retrieves data from it’s ledger database, prepares the data format to comply with our system file requirements. This can be completely automatic by the system or initiated semi automatically by the customer by activating an additional function that performs the necessary steps.

3.5.6 File manual process

Customers using manual file processing will be indicated in the BO with the File upload setting "manual". This is used to tell the system what is normal. Customer with the "auto" setting on shall however also be able to upload manually (as their automatic upload function possible can fail).

This page shall be accessible to all customers for both up- and download purposes.

The page is thought to be a part of the Customer Portal.

The Client user can choose services. In this system we in this phase will have only one Product, the Customer Value New Biz Report. (No result files or free reports are available.).

On this page the files uploads shall be displayed:

<table>
<thead>
<tr>
<th>File</th>
<th>Period</th>
<th>Date</th>
<th>Time</th>
<th>Uploaded by</th>
<th>Status</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>[file name]</td>
<td>YYMM</td>
<td>YYMMDD</td>
<td>HH:MM</td>
<td>[user ID]</td>
<td>[system status]</td>
<td>[button]</td>
</tr>
</tbody>
</table>

The user shall be able to download his own file on request (button) for control purposes.

The Download button shall be active as soon as the upload and file save is complete regardless of file status.
It shall only be possible to upload a file if the setting "Products used" < "Products bought" + "Products free". If not the Download button should remain inactive and a text should be displayed on page. "The products available in your account has been used. Please contact Customer Value Support. They will help to add products to your account. [contact info to support added into text]"

Reports generated shall also be displayed:

<table>
<thead>
<tr>
<th>Period</th>
<th>Based on</th>
<th>Produced</th>
<th>Status</th>
<th>Downloaded</th>
<th>Download</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYMM</td>
<td>[file name]</td>
<td>[system date]</td>
<td>[Report status]</td>
<td>[Yes/No]</td>
<td>[button]</td>
</tr>
</tbody>
</table>

The download button should initially be inactive. See comments in "Indicating" column below. It should be possible to download a file several times.

File statuses should be:

<table>
<thead>
<tr>
<th>EN</th>
<th>SW</th>
<th>Indicating</th>
<th>Explanation to user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Received</td>
<td>Mottagen</td>
<td>File has been uploaded. It has been queued for quality check.</td>
<td>The file will be quality checked.</td>
</tr>
<tr>
<td>Checked</td>
<td>Kontrollerad</td>
<td>Ready for production, that is it has been received and checked to be ok. See File quality check below.</td>
<td>The file is ready and queued for report production.</td>
</tr>
<tr>
<td>Error</td>
<td>Felaktig</td>
<td>The file quality check was not passed.</td>
<td>The file seems to have a format error. Please check file. Contact Support for assistance.</td>
</tr>
<tr>
<td>Processing</td>
<td>Bearbetas</td>
<td>The file has been transfered to the production routine.</td>
<td>The file is being used to produce a report.</td>
</tr>
<tr>
<td>Replaced</td>
<td>Ersatt</td>
<td>The user has upload a new file for the same period.</td>
<td>A new file has replaced the former file.</td>
</tr>
</tbody>
</table>

Report statuses should be:

<table>
<thead>
<tr>
<th>EN</th>
<th>SW</th>
<th>Indicating</th>
<th>Explanation to user</th>
</tr>
</thead>
<tbody>
<tr>
<td>Producing</td>
<td>Framställer</td>
<td>The report processing has commenced but is not ready. Download button is inactive.</td>
<td>The report is being produced.</td>
</tr>
<tr>
<td>Ready</td>
<td>Klar</td>
<td>The report is ready. The Download button is made active.</td>
<td>Ready for download. Klar för att hämtas.</td>
</tr>
</tbody>
</table>
(If needed status = "Unknown error" can of course be included.)
(We see no need to automatically clear old files by an algorithm at this stage.)

Services shall be:

**New Biz Report**
- "Upload a file" / "Ladda upp en fil"
- "Fetch report" / "Hämta rapport"

  When we will add the service Customer Value Report a Result file can be downloaded by the customer. Result Files will only be delivered to Charge Clients if they have a file name complying with the Reports accessible setting in BO or One-off clients with the "Number of Reports left" value > 0 or blank.

  The number of free Reports will be controlled by the "Number of Free Reports left" field in BO. As a Free Report has been uploaded to the database the "Number of Free Reports left" should be reduced by one. If "Number of Free Reports left" < 1 then a message should be printed to the user.

    "Dina fria rapporter är slut. Kontakta Customer Value på customerservice@customervalue.se så hjälper vi dig att utöka antalet rapporter".

**File quality check:**

The file should be checked before further actions. A simple check would be nice to have already in this phase:
- filed separator should be tab or pipe [ | ]
- the number of fields (columns) should be [...]  
- in case customer has a file column headers = standard setting the first record (row) should contain headers: x;x;x;x; [to be discussed]
- fields (columns) x, x and x has numeric content
- text does not contain characters that can cause errors in production [to be discussed].

The File quality check should produce File status as described above.

If a new file is uploaded with the same file name by the Client this file shall get the same name as described above but with a new date/time stamp. The Client user should however be warned before this new file version is accepted. If a report has not been produced yet:

"A file with this name has already been uploaded. Do you want to replace the file you already have uploaded?"

"En fil med samma namn finns redan upplagd. Vill du att ersätta den tidigare uppladdade filen?"

Button "Yes replace the former file" / "Ja. Ersätt den tidigare filen"

---

6 It will not be meaningful to try to upload files with an old period in the name (to get extra reports/files not paid for) as the result will be corrupted. The period is a parameter used in the statistical analysis. New downloads with an old name will be rejected as the Report has been generated.
Button "No Cancel." / "Nej. Avbryt"

If yes overwrite the old file. If no, cancel and do nothing.
In case a replacement file is uploaded this should not cause the usage counter to register is as a new download.

If a report has been produced (checked with the Report part of the database for Reports with the same period in its identification):
"A file with the same name has already been uploaded. A report was also produced. Do you want to replace the former file and have a new report produced."
"En fil med samma namn har tidigare lagts upp. En rapport har skapats från den filen. Vill du ersätta den gamla filen och skapa en ny rapport?."

Button "Yes replace the former file" / "Ja. Ersätt den tidigare filen, skapa ny rapport."
Button "No Cancel." / "Nej. Avbryt"

In case a replacement file is uploaded this should cause the usage counter to register is as a new upload. (Customers will have to contact CV support to have this undone.)

3.5.7 File columns mapping

As customers in some cases are not able to use our standard column names. This should be indicated in the BO by a customer "File column headers" setting = own. When "own" we need to map the fields to our standard fields. A table should be added per customer (to be used only when the setting = "own") translating the uploesd field names to our standard. This table is updated manually or by a mapping interface in the customer portal.

Se Appendix A Customer settings "File column headers".

3.5.8 Marketing costs

The New Biz report will in next version include a function to analyze the marketing investments compared to results as shown by the New Biz report.

We need to make it possible to upload such data or register it within the Customer Portal.

For this purpose we need a separate Portal page available. In this the customer shall be able to choose period (month) and add a cost associated with activities aiming at customer purchases within this period. These can be marketing campaigns made, discounts given etc).

In the first version simple data will be entered

<table>
<thead>
<tr>
<th>Result month</th>
<th>Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMYY</td>
<td>ZZZ ZZZ</td>
</tr>
</tbody>
</table>

Costs are registered in Main currency. (It is up to the customer to use the correct currency.)
The entries shall be stored in a TABLE accessible to the production system. This will be specified as an addition to this FRD.

3.6 File upload automatic process

Customers using automatic file processing will be indicated in the BO with the File upload setting "auto".

Our system does not have to consider whether the data extraction and adaptation has been made automatically or semi-automatically. The files will be delivered in a standard format we already know is correct. This means that the quality check is not needed.

Needed however is a routine that is checking the upload area (SFTP or whatever chosen) for new files.
When a new file has been detected the system should start the process.

### 3.7 CV New Biz Report Production

The Report and the corresponding pdf is the core of the system. It contains an analysis of the consumer purchase behavior recent 36 months of the customer. It

- performs calculations of the data to produce the graphs and tables of the according to a model specified by Customer Value (Appendix C - Code).
- constructs the tables (this is now made by the Excel document enclosed)
  a) containing data needed to produce graphs
  b) to be included in the New Biz Report

(a and b are not the same)
- constructs the graphs to be included in the New Biz Report (this is now made by the Excel document enclosed)
- generates the New Biz Report (source for pdf)
- generates the New Biz Report pdf
  - [generates a return file with results data for each Report to be retrieved by the customer.]
- updates the New Biz Report and the New Biz Report pdf [+ Return File when applicable] to the customer database to be distributed to the customer and saved.

*The rules to be used to build the reports we think is best understood by*

- reading the code (Appendix C). *This is a code direct from the statistic tool Strata now used*
- investigating the MS Excel document now used to produce tables and graphs and build the document that (now) is printed to pdf.

Enclosed explanatory files are:

customer input:  temp_20140219a.txt
                 temp_20140219a_exempel.xlsx
intermediate file: temp_20140219b.txt
                  temp_20140219b_exempel.xlsx
report production file: newbiz13002d_rapport_v401_nyversion.xlsx
This is an explanation of the presently used procedure to summarize detailed consumer data items needed to produce the tables and graphs in the report product in Excel. The reason for this procedure is that we report by month. The consumer may well make more than one purchase within a single month. We also need to adjust the amounts for any returned orders by consumers.

Customer data

The temp_20140219a.txt file contains a customer data set. The temp_20140219a_exempel.xlsx file contains the first 100 rows. This is provided to provide a better view of the data structure.

From the customer data the adjusted dataset that you will find in temp_20140219b.txt is created. The first 100 records in temp_20140219b_exempe1.xlsx.

The main purpose is to create the column N that shall include the order amounts adjusted with the returned goods amounts. The columns H - N will be explained before long.

From the above sheet the summarized data basis you will find in the sheet "Rådata från Stata" (raw data from Stata) in newbiz13002d_rapport_v401_nyversion.xlsx is created.

A first and last page (report cover) should to be added. Cover page 1 shall have a text added with

- Customer name
- Month year - the period the report covers eg "September 2014"
- Production date eg "October 8 2014".

See Appendix F.

The report shall be produced in English. We may in the future make it possible to adapt to different languages.

### 3.8 Security

<table>
<thead>
<tr>
<th>edgeid</th>
<th>country</th>
<th>ord_date</th>
<th>ord_year</th>
<th>currency</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 SE</td>
<td>4/29/09</td>
<td>2009</td>
<td>SEK</td>
<td>241,92</td>
<td></td>
</tr>
<tr>
<td>10 SE</td>
<td>12/22/09</td>
<td>2009</td>
<td>SEK</td>
<td>185,92</td>
<td></td>
</tr>
</tbody>
</table>

### 3.8.1 Certificates

<table>
<thead>
<tr>
<th>edgeid</th>
<th>custno</th>
<th>ord_date</th>
<th>ord_year</th>
<th>country</th>
<th>currency</th>
<th>bex_oms</th>
<th>nyku</th>
<th>yrkun</th>
<th>retur_bex</th>
<th>retur_oms</th>
<th>oms_fix</th>
<th>retur_fix</th>
<th>ord_oms</th>
<th>ctry</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 0000000010</td>
<td>4/29/09</td>
<td>9</td>
<td>2009</td>
<td>SE</td>
<td>SEK</td>
<td>241,9</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>241,9</td>
<td>2</td>
</tr>
<tr>
<td>10 0000000010</td>
<td>12/22/09</td>
<td>09</td>
<td>2009</td>
<td>SE</td>
<td>SEK</td>
<td>185,9</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>185,9</td>
<td>2</td>
</tr>
</tbody>
</table>
5 Success Criteria
In this order:

1 High priority
A database with functions in black text included and fully functional
Back Office routines in black text included and fully functional.
A routine producing the New Biz report as documented in FRD.
Implemented and tested.
System documentation including code.

2 Medium priority
Customer portal as described in black text.
Suggestions for how to expand functions of the system to include blue text functions and a general description of the system scalability.

3 Lower priority
Blue text functions.

6 Sign Off

<table>
<thead>
<tr>
<th>Sign off</th>
<th>Name</th>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Appendix A – BO (internal) Data

### Customers

<table>
<thead>
<tr>
<th>Field</th>
<th>Field name</th>
<th>Rights, all</th>
<th>Rights, read</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Char</td>
<td>All rights</td>
<td>Read rights</td>
</tr>
<tr>
<td>Customer ID</td>
<td></td>
<td>N</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Company number</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Supplementary id no.</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Active/inactive</td>
<td></td>
<td></td>
<td>CV Admin</td>
<td>y/n</td>
</tr>
<tr>
<td>Customer category</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>Charge; Free</td>
</tr>
<tr>
<td>Main market</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>CV owned by, main group</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>CV owned by, sub group</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Company Name</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Address 1</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Address 2</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Address, postal code</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Address, town</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Address, country</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>e-mail (company)</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Phone (company)</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Date recorded</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
<tr>
<td>Date changed</td>
<td></td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
</tr>
</tbody>
</table>
Customer settings [rights all CV staff]

<table>
<thead>
<tr>
<th>FIELD</th>
<th>COMMENT</th>
<th>COMPULSORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer ID</td>
<td>Customer ID identifying the respective customer.</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>See also note below.</td>
<td></td>
</tr>
<tr>
<td>Country Code</td>
<td>The country in which the transaction was generated.</td>
<td>Yes, if transactions covering more than one market are included.</td>
</tr>
<tr>
<td></td>
<td>Use any of the ISO code standards (2, 3 alpha or numeric 3).</td>
<td></td>
</tr>
<tr>
<td>Order ID</td>
<td>Order number identifying the unique purchases.</td>
<td>Yes, if market(s)</td>
</tr>
<tr>
<td></td>
<td>Order ID will facilitate a better match quality identifying the original</td>
<td></td>
</tr>
<tr>
<td></td>
<td>order of any returned items.</td>
<td></td>
</tr>
<tr>
<td>Transaction date/time</td>
<td>Transaction date in the format YYYY-MM-DD HH:MM:SS.</td>
<td>Yes. Time can be excluded.</td>
</tr>
<tr>
<td>Currency</td>
<td>Currency of the transaction value. Use ISO 4217 or include a translation</td>
<td>Yes, if transactions covering more than one market are included.</td>
</tr>
<tr>
<td></td>
<td>table if a proprietary code is used.</td>
<td></td>
</tr>
<tr>
<td>Belopp</td>
<td>Transaction value excluding VAT.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field column headers</th>
<th>&quot;standard&quot; / &quot;own&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report currency</td>
<td>ISO 4217 three char alpha (where the two first characters = ISO 3166-1 alpha-2)</td>
</tr>
<tr>
<td>Sub-Market 1 [etc]</td>
<td>ISO codes (several); each connected to one ISO currency</td>
</tr>
<tr>
<td>Report currency&quot; [ISO 4217]</td>
<td></td>
</tr>
</tbody>
</table>
### Product market separation

"Off" country transactions should be accumulated in the analysis as one single country report using Main currency. Currencies other than Main, according to code in customer input file, shall be converted to Mail. Default off.

"On" = report shall be made in one main version for all markets (see Markets above) + separate reports for each Sub-market (according to Market settings). This is not relevant in this system version.

### Report localization

"Main Currency" = [ISO 4217 three char alphabetic code can be used (if no argument against it)]

"Main Language" = [ISO 639-1]

### Currency conversion

"Input Currency Code System" [ISO 4217; bespoke]

"Year" [use table as Jan 1st]

"Updated" [use daily currency values via xml service like Oanda]

Note. We do not plan to subscribe to a xml service until later. But please do prepare if time is no problem.

Default “Year”

### File transfer method

Manual/Auto

Auto-system [name of accounting system - for info only]

### Users

<table>
<thead>
<tr>
<th>Field</th>
<th>Field name</th>
<th>Rights, all</th>
<th>Rights, read</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer ID</td>
<td>AN</td>
<td>CV User</td>
<td>CV User</td>
<td>link Customer ID in Customer table</td>
</tr>
<tr>
<td>User ID</td>
<td>AN</td>
<td>CV Admin</td>
<td>CV User</td>
<td>Set by system</td>
</tr>
<tr>
<td>User Name</td>
<td>AN</td>
<td>CV User</td>
<td>CV User</td>
<td></td>
</tr>
<tr>
<td>Main contact</td>
<td></td>
<td></td>
<td></td>
<td>y/n</td>
</tr>
<tr>
<td>User access to products</td>
<td>CV User</td>
<td>CV User</td>
<td></td>
<td>Eg NB-SE. Can be more than one. This decides to whom a specific product shall be distributed / available.</td>
</tr>
<tr>
<td>e-mail</td>
<td>AN</td>
<td>CV User</td>
<td>CV User</td>
<td></td>
</tr>
<tr>
<td>Phone</td>
<td>AN</td>
<td>CV User</td>
<td>CV User</td>
<td></td>
</tr>
<tr>
<td>Date recorded</td>
<td>AN</td>
<td>CV User</td>
<td>CV User</td>
<td>[syst date]</td>
</tr>
</tbody>
</table>
Orders

<table>
<thead>
<tr>
<th>Product</th>
<th>Period</th>
<th>Start month</th>
<th>Products bought</th>
<th>Products free</th>
<th>Products used</th>
</tr>
</thead>
<tbody>
<tr>
<td>[product id]</td>
<td>[monthly/quarterly]</td>
<td>[YYMM]</td>
<td>[numeric]</td>
<td>[numeric]</td>
<td>[numeric]</td>
</tr>
</tbody>
</table>

Usage table (details) many occurrences per product

|---------|---------|----------------------|----------------------|---------------------------------|

Transferred to accounting system can be a check box (on/off; default off). Manual; column for information to CV staff only.

Products

<table>
<thead>
<tr>
<th>Product code</th>
<th>Market</th>
<th>Owned by</th>
<th>Currency</th>
<th>Price of product</th>
<th>Price 2 of product</th>
<th>Price 3 of product</th>
<th>Comment</th>
</tr>
</thead>
</table>

FIELD | COMMENT | COMPULSORY |
---|---|---|
Customer ID | Customer ID identifying the respective customer. See also note below. | Yes |
Country Code | The country in which the transaction was generated. Use any of the ISO code standards (2, 3 alpha or numeric 3). | Yes, if transactions covering more than one market are included. |
Order ID | Order number identifying the unique purchases. Order ID will facilitate a better match quality identifying the original order of any returned items. | Yes, if market s |
Transaction date/time | Transaction date in the format YYYY-MM-DD HH:MM:SS. | Yes. Time can be excluded. |
Currency | Currency of the transaction value. Use ISO 4217 or include a translation table if a proprietary code is used. | Yes, if transactions covering more than one market are included. |
Belopp | Transaction value excluding VAT. | Yes |
Owned by is retrieved automatically by the system and cannot be changed manually. Compare USERS.

<table>
<thead>
<tr>
<th>NB</th>
<th>SE</th>
<th>[CV User ID]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NB</td>
<td>GR</td>
<td>[User ID]</td>
<td></td>
</tr>
</tbody>
</table>

**File format (input)**

<table>
<thead>
<tr>
<th>Customer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country Code (ISO 3166)</td>
</tr>
<tr>
<td>Transaction date/time</td>
</tr>
<tr>
<td>Currency (ISO 4217 code)</td>
</tr>
<tr>
<td>Amount (purchase) excl VAT</td>
</tr>
</tbody>
</table>

**File format Result File**

<table>
<thead>
<tr>
<th>Customer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>[to be completed]</td>
</tr>
</tbody>
</table>

**Marketing costs** (compare 3.5.7)

<table>
<thead>
<tr>
<th>Result month</th>
<th>Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>[YYMM]</td>
<td>[integrals]</td>
</tr>
</tbody>
</table>
Bespoke currency system conversion table

This table will be used to convert the specific customer’s non standard currency codes to ISO if the "Input currency code system" setting (compare above) is "bespoke".

<table>
<thead>
<tr>
<th>Bespoke code</th>
<th>ISO 4217</th>
</tr>
</thead>
<tbody>
<tr>
<td>[any format]</td>
<td></td>
</tr>
<tr>
<td>[example:]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>country</th>
<th>ord_dat</th>
<th>ord_year</th>
<th>currenc_y</th>
<th>bex_om</th>
<th>custno</th>
<th>nykund</th>
<th>yrkund</th>
<th>retur_bex</th>
<th>retur_om</th>
<th>oms_fix</th>
<th>retur_fi</th>
<th>ord_om</th>
<th>ord_tb</th>
<th>ctry</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE</td>
<td>4/29/09</td>
<td></td>
<td></td>
<td>241.92</td>
<td>100000001</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>241.92</td>
<td>120,96</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>12/22/09</td>
<td></td>
<td></td>
<td>185.92</td>
<td>100000001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185.92</td>
<td>92,96</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>6/7/10</td>
<td></td>
<td></td>
<td>74.88</td>
<td>100000001</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>74.88</td>
<td>37,44</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>1/28/11</td>
<td></td>
<td></td>
<td>31.36</td>
<td>100000001</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>31.36</td>
<td>15,68</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>6/11/11</td>
<td></td>
<td></td>
<td>606.72</td>
<td>100000001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>606.72</td>
<td>303,36</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>1/7/12</td>
<td></td>
<td></td>
<td>533.12</td>
<td>100000001</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>533.12</td>
<td>266,56</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>5/25/12</td>
<td></td>
<td></td>
<td>892.8</td>
<td>100000001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>892.8</td>
<td>446,4</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>12/14/12</td>
<td></td>
<td></td>
<td>287.36</td>
<td>100000001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>287.36</td>
<td>143,68</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>8/27/13</td>
<td></td>
<td></td>
<td>140.16</td>
<td>100000001</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>140.16</td>
<td>70,08</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>9/18/10</td>
<td></td>
<td></td>
<td>133.08</td>
<td>100000002</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>133.08</td>
<td>66,54</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>9/29/11</td>
<td></td>
<td></td>
<td>240.32</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>240.32</td>
<td>120,16</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>10/13/1</td>
<td></td>
<td></td>
<td>383.36</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>383.36</td>
<td>191,68</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>10/14/1</td>
<td></td>
<td></td>
<td>383.36</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>752</td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>10/14/1</td>
<td></td>
<td></td>
<td>-383.36</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-383.36</td>
<td>10</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>11/4/11</td>
<td></td>
<td></td>
<td>496.64</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>496.64</td>
<td>248,32</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>11/19/1</td>
<td></td>
<td></td>
<td>138.24</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>138.24</td>
<td>69,12</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>12/26/1</td>
<td></td>
<td></td>
<td>70.4</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>70.4</td>
<td>35,2</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>12/28/1</td>
<td></td>
<td></td>
<td>262.91</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>262.91</td>
<td>131,46</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>2/15/12</td>
<td></td>
<td></td>
<td>239.36</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>239.36</td>
<td>119,68</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>4/3/12</td>
<td></td>
<td></td>
<td>76.8</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>76.8</td>
<td>38,4</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>4/12/12</td>
<td></td>
<td></td>
<td>126.72</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>126.72</td>
<td>63,36</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>4/18/12</td>
<td></td>
<td></td>
<td>374.4</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>374.4</td>
<td>187,2</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>5/30/12</td>
<td></td>
<td></td>
<td>0</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>6/6/12</td>
<td></td>
<td></td>
<td>56.96</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>56.96</td>
<td>28,48</td>
<td>752</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>7/11/12</td>
<td></td>
<td></td>
<td>63.36</td>
<td>100000002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>63.36</td>
<td>31,68</td>
<td>752</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### File column headers

Compare Appendix B.1.

This is just a to indicate that a mapping table is needed; the design is thus not prescribed.

Note that the field order in the uploaded file may be (probably will be) different than the field order we will have.

The intermediate field names for transactions have not been decided by CV. We suggest any fairly descriptive field names (see App B.1).

<table>
<thead>
<tr>
<th></th>
<th>CV field name 1</th>
<th>etc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer A</td>
<td>customer field name 3</td>
<td>customer field name 1</td>
</tr>
<tr>
<td>Customer B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>etc</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Rights

<table>
<thead>
<tr>
<th>Group</th>
<th>Change/delete - account - charge details</th>
<th>Add, change and delete other accounts parameters</th>
<th>Read</th>
</tr>
</thead>
</table>
| CV Management   | Y  - all customer details all markets  
- any CV details | Y all                                           | Y all   |
| CV Admin        | Y  - all customer details within own market administrated | Y - all customer details within own markets administrated | Y - all customer details within own markets administrated |
| CV BO User      | N                                         | Y                                             | Y        |
| CV Sales        | N                                         | N                                             | Y        |
| Customer Admin  | N                                         | Y - limited customer details within subscribed market(-s). Compare … | Y - all customer details within subscribed market(-s) |
| Customer User   | N                                         | N                                             | Y - all customer details within subscribed market(-s) |
Appendix B.1 - File Specification

The Customer Value Report input data requirements are as follows.

To perform the analysis transaction data retrieved from the accounting / ERP system is required. The transaction data shall cover the business area the analysis is supposed to cover only. If several business areas (eg different brands, countries etc) shall be included we need the data separated in different files.

Your IT department or IT consultant should be able to help you extract the data needed.

All transactions must be included in the file(-s). This means that

- The transactions must include all purchases but also all returned items.
- For returned items we need the transactions to include negative amounts corresponding with the items returned.
- To be able to identify new customers correctly the customer ID needs to be identical for all transactions for a customer. Take contact with Customer Value Support if you believe this is not the case.

First upload (basic file)

A complete file is needed covering, if possible, all transactions from the start of the web shop.

If a shorter period than this is delivered please consider that customers that have bought before this specific month will be considered to be new customers when they buy again within the time scope the delivered file covers.

Include transactions for former customers (churn). That is, it is not sufficient to include present customers only.

Monthly uploads

Include new transactions only that have been registered in the month the file covers.

File format

File format should be tab separated text file. Pipe can also be used as a column separator.

Note

The customer ID can be the actual customer ID used in the accounting system ledger. Customer Value will alternatively use any anonymized or encrypted ID that is based on the identities in the accounting system. If the latter will be provided it is crucial that the same conversion method is used for every file delivered to Customer Value enabling us to make a unambiguous connection to the respective actual customer ID in the customer ledger.

Customer file example in Appendix B.2
Appendix B.1 - Customer file example

The customer file contains transactions - purchases or returned goods (negative bex_oms = order price). This file is produced by an accounting system named Bex. We will receive files with different column titles as customers are using different accounting systems. These should be mapped to the system parameters to be used by our system. Compare 3.5.8 and Appendix A File column headers.

The file is enclosed clv1001_all_exempel_v301.xlsx
Appendix C - Code

This is the code now used by the statistical analysis program Strata to build the input for graph and table building. This is a static code, it needs not to be changed (unless the product is further developed or corrected). We need this to be recoded into the system.

The code calculates - from the transactions input - the data tables needed to produce the report graphs and tables.

Code to handle marketing costs as described in 3.5.7 will be added.

The last row writes a text file (in this case baby13002d_v310). This file is (in the present manual solution) copied into a Excel sheet (in this case baby13002d_rapport_v310).

* BABY13002 Andra program för månadsstatistik
* Från CLV12204 som skattar modeller för olika tidsperioder

```
version 11

clear
set more off
set memory 128m

local katalog "Kund0004"
local ver "v310"  /* Programversion */
use "\katalog'/clv1001_all_'ver''

drop if retur_bex==1  /* Tar ej med negativa poster från BEX^^*/

generate nykund_noll=1 if nykund==1 & ord_oms==0
replace nykund_noll==0 if nykund_noll==.
generate nykund_real=1 if nykund==1 & nykund_noll!=1
replace nykund_real==0 if nykund_real==.

generate first_oms=ord_oms if nykund==1  /* Variabel som mäter första köp */
generate ord_mon=mofd(ord_date)  /* Variabel som mäter första köp */
format ord_mon %tmCCYY-NN

tempfile babydata1001
save "`babydata1001'", replace  /* Sparar hela materialet */

collapse (sum) ord_oms first_oms nykund nykund_real nykund_noll ///
(mean) mean_ord_oms=ord_oms ///
(count) anttrans=ord_oms firsttrans=first_oms antretur=oms_fix ///
(min) min_date=ord_date ///
(max) max_date=ord_date oms_fix, by (custno ord_mon)

generate oldkund=1 if nykund==0
replace oldkund==0 if oldkund==.
generate oldkund_noll=1 if oldkund==1 & ord_oms==0
replace oldkund_noll==0 if oldkund_noll==.
generate oldkund_real=1 if oldkund==1 & oldkund_noll!=1
replace oldkund_real==0 if oldkund_real==.
generate oldkund_oms=ord_oms if nykund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1
```

*BABY13002* Andra program för månadsstatistik
*Från CLV12204 som skattar modeller för olika tidsperioder*

version 11

clear
set more off
set memory 128m

local katalog "Kund0004"
local ver "v310"  /* Programversion */
use "\katalog'/clv1001_all_'ver''

drop if retur_bex==1  /* Tar ej med negativa poster från BEX^^*/

generate nykund_noll=1 if nykund==1 & ord_oms==0
replace nykund_noll==0 if nykund_noll==.
generate nykund_real=1 if nykund==1 & nykund_noll!=1
replace nykund_real==0 if nykund_real==.

generate first_oms=ord_oms if nykund==1  /* Variabel som mäter första köp */
generate ord_mon=mofd(ord_date)  /* Variabel som mäter första köp */
format ord_mon %tmCCYY-NN

tempfile babydata1001
save "`babydata1001'", replace  /* Sparar hela materialet */

collapse (sum) ord_oms first_oms nykund nykund_real nykund_noll ///
(mean) mean_ord_oms=ord_oms ///
(count) anttrans=ord_oms firsttrans=first_oms antretur=oms_fix ///
(min) min_date=ord_date ///
(max) max_date=ord_date oms_fix, by (custno ord_mon)

generate oldkund=1 if nykund==0
replace oldkund==0 if oldkund==.
generate oldkund_noll=1 if oldkund==1 & ord_oms==0
replace oldkund_noll==0 if oldkund_noll==.
generate oldkund_real=1 if oldkund==1 & oldkund_noll!=1
replace oldkund_real==0 if oldkund_real==.
generate oldkund_oms=ord_oms if nykund==0

generate nykund_oms=ord_oms if nykund==1

generate oldkund_oms=ord_oms if oldkund==0

generate nykund_oms=ord_oms if nykund==1

tempfile babydata1002
save "`babydata1002'", replace  /* Sparar hela materialet */
```
Customer Value Production system 1.0

2013-01-18

collapse (sum) ord_oms first_oms nykund_oms oldkund_oms ///
    nykund nykund_real nykund_noll ///
    oldkund oldkund_real oldkund_noll anttrans firsttrans

antretur ///
    (count) antkund=custno ///
    (min) min_date ///
    (max) max_date, by (ord_mon)

tempfile babydata1002mon
save "`babydata1002mon'", replace /* Sparar månadsdata */

* format %12.0g ord_oms oldkund_oms nykund_oms
* list
* outsheet using "`katalog'/baby13002a_`ver'.txt", replace

summarize min_date, meanonly
scalar min_kopdatum = r(min)
summarize max_date, meanonly
scalar max_kopdatum = r(max)
display min_kopdatum max_kopdatum

**********************************************************************
* local startdat td(01022010) /* Välj startdatum (dagen efter startmånad) */
* local steps 40 /* Välj antal månader */
local startdat=dofm(mofd(max_kopdatum)-`steps'+2) /* Antal månader som ska
summeras */
local utvperiod1 3 /* Antal månader som ska summeras */
local utvperiod2 12 /* Antal månader som ska summenas */
local utvperiod3 24 /* Antal månader som ska summeras */

* local steps=mofd(max_kopdatum)-mofd(`startdat')+2
* display `steps'

local i=0
local dat1=`startdat'

* use "`babydata1002'", clear
tempfile babydata1003 /* Nykundscohortens kundnummer */
tempfile babydata1004 /* Initial försäljning till nya kunder */
tempfile babydata1005 /* Återköp inom 3 månader */
tempfile babydata1006 /* Återköp inom 12 månader */
tempfile babydata1007 /* Återköp inom 24 månader */

while `i'<=`steps'-1 {
    local dat0=dofm(mofd(`dat1')-1)
    local dat21=dofm(mofd(`dat1')+`utvperiod1'-2)
    local dat22=dofm(mofd(`dat1')+`utvperiod2'-2)
    local dat23=dofm(mofd(`dat1')+`utvperiod3'-2)
    use "`babydata1002'", clear /* Data per kund och månad från ovan */
    keep if ord_mon==mofd(`dat0') & nykund==1 /* Väljer
nykundscohort */
collapse (min) min_date, by(custno)
generate nyk_cohort=mofd(min_date) /*
Nykundscohort */
drop min_date
save "`babydata1003''", replace /* Sparar
cohortens kundnummer */
merge 1:m custno using "`babydata1001''", keep(match)
format %tmCCYY-NN nyk_cohort ord_mon
keep if nykund==1
collapse (sum) ord_oms nykund nykund_real nykund_noll ///
       (count) anttrans=ord_oms
antretur=oms_fix ///
       (min) nyk_cohort min_date=ord_date
 ///
       (max) max_date=ord_date oms_fix, by
(custno)
generate oldkund=1 if nykund==0
replace oldkund=0 if oldkund==.
generate oldkund_noll=1 if oldkund==1 & ord_oms<0
replace oldkund_noll=0 if oldkund_noll==.
generate oldkund_real=1 if oldkund==1 & oldkund_noll!=1
replace oldkund_real=0 if oldkund_real==.
generate oldkund_oms=ord_oms if nykund==0
generate nykund_oms=ord_oms if nykund==1
collapse (sum) ord_oms=ord_oms ///
nykund_oms oldkund_oms nykund ///
nykund_real0=nykund_real ///
nykund_noll0=nykund_noll ///
oldkund oldkund_real oldkund_noll
 ///
    anttrans0=anttrans ///
    antretur0=antretur ///
       (count) antkund0=custno ///
       (min) min_date ///
       (max) max_date, by (nyk_cohort)
if `i'==0 {
    save "`babydata1004''", replace
} else if `i'>0 {
    append using "`babydata1004''",
    replace
}
if mofd(`dat21')<=mofd(max_kopdatum) {
    display `dat21'
    use "`babydata1003''", clear
    merge 1:m custno using "`babydata1001''",
    keep(match)
    format %tmCCYY-NN nyk_cohort ord_mon
    keep if ord_mon>=mofd(`dat0') &
    ord_mon<=mofd(`dat21') &
    nykund!=1
    if _N>0 {
        collapse (sum) ord_oms nykund nykund_real
    }
    antretur=oms_fix ///
    (min) nyk_cohort
    min_date=ord_date ///
oms_fix, by (custno)

generate oldkund=1 if nykund==0
replace oldkund=0 if oldkund==.
generate oldkund_noll=1 if oldkund==1 &
ord_oms<=0
replace oldkund_noll=0 if oldkund_noll==.
generate oldkund_real=1 if oldkund==1 &
oldkund_noll!=1
replace oldkund_real=0 if oldkund_real==.
generate oldkund_oms=ord_oms if nykund==0
generate nykund_oms=ord_oms if nykund==1
collapse (sum) ord_oms`utvperiod1'=ord_oms
    ///
    nykund_oms oldkund_oms
nykund ///
oldkund ///

oldkund_real`utvperiod1'=oldkund_real ///
oldkund_noll`utvperiod1'=oldkund_noll ///
anttrans`utvperiod1'=anttrans ///
antretur`utvperiod1'=antretur ///

(antkund`utvperiod1'=custno ///

(nyk_cohort)
if `i'==0 {
    save "`babydata1005'",
}
else if `i'>0 {
    append using
    "`babydata1005'"
    save
    "`babydata1005'", replace
}
)
if mofd(`dat22')<=mofd(max_kopdatum) {
    display `dat22'
    use "`babydata1003'", clear
    merge 1:m custno using "`babydata1001'",
    keep(match)
    format %tmCCYY-NN nyk_cohort ord_mon
    keep if ord_mon>=$mo0 &
    ord_mon<=$mo22 & ///
    nykund=!1
    if _N>0 {
        collapse (sum) ord_oms nykund nykund_real
    }
ynkund_noll ///
antretur=oms_fix ///
min_date=ord_date ///
oms_fix, by (custno)

(anttrans=ord_oms
(count)

(nyk_cohort)
(min) min_date ///
(max) max_date, by

(nyk_cohort)
(min) min_date ///
(max) max_date, by
generate oldkund=1 if nykund==0
replace oldkund=0 if oldkund == .
generate oldkund_noll=1 if oldkund==1 &
ord_oms<=0
replace oldkund_noll=0 if oldkund_noll== .
generate oldkund_real=1 if oldkund==1 &
oldkund_noll!=1
replace oldkund_real=0 if oldkund_real== .
generate oldkund_oms=ord_oms if nykund==0
generate nykund_oms=ord_oms if nykund==1
### Field title

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ord_mon</td>
<td></td>
</tr>
<tr>
<td>ack_oms</td>
<td></td>
</tr>
<tr>
<td>ack_nykund</td>
<td></td>
</tr>
<tr>
<td>ack_anttrans</td>
<td></td>
</tr>
<tr>
<td>aktivkund</td>
<td></td>
</tr>
<tr>
<td>ack_kundnum</td>
<td></td>
</tr>
<tr>
<td>ack_min_date</td>
<td></td>
</tr>
<tr>
<td>ack_max_date</td>
<td></td>
</tr>
<tr>
<td>ord_oms</td>
<td></td>
</tr>
<tr>
<td>first_oms</td>
<td></td>
</tr>
<tr>
<td>nykund_oms</td>
<td></td>
</tr>
<tr>
<td>oldkund_oms</td>
<td></td>
</tr>
<tr>
<td>nykund</td>
<td></td>
</tr>
<tr>
<td>nykund_real</td>
<td></td>
</tr>
<tr>
<td>nykund_noll</td>
<td></td>
</tr>
<tr>
<td>oldkund</td>
<td></td>
</tr>
<tr>
<td>oldkund_real</td>
<td></td>
</tr>
<tr>
<td>oldkund_noll</td>
<td></td>
</tr>
<tr>
<td>anttrans</td>
<td></td>
</tr>
<tr>
<td>firsttrans</td>
<td></td>
</tr>
<tr>
<td>antretur</td>
<td></td>
</tr>
<tr>
<td>antkund</td>
<td></td>
</tr>
<tr>
<td>min_date</td>
<td></td>
</tr>
<tr>
<td>max_date</td>
<td></td>
</tr>
<tr>
<td>nyk_cohort</td>
<td></td>
</tr>
<tr>
<td>ord_oms0</td>
<td></td>
</tr>
<tr>
<td>nykund_real0</td>
<td></td>
</tr>
<tr>
<td>nykund_noll0</td>
<td></td>
</tr>
<tr>
<td>anttrans0</td>
<td></td>
</tr>
<tr>
<td>antretur0</td>
<td></td>
</tr>
<tr>
<td>antkund0</td>
<td></td>
</tr>
<tr>
<td>ord_oms3</td>
<td></td>
</tr>
<tr>
<td>oldkund_real3</td>
<td></td>
</tr>
<tr>
<td>anttrans3</td>
<td></td>
</tr>
<tr>
<td>antretur3</td>
<td></td>
</tr>
<tr>
<td>antkund3</td>
<td></td>
</tr>
<tr>
<td>ord_oms12</td>
<td></td>
</tr>
<tr>
<td>oldkund_real12</td>
<td></td>
</tr>
<tr>
<td>anttrans12</td>
<td></td>
</tr>
<tr>
<td>antretur12</td>
<td></td>
</tr>
<tr>
<td>antkund12</td>
<td></td>
</tr>
<tr>
<td>ord_oms24</td>
<td></td>
</tr>
<tr>
<td>oldkund_real24</td>
<td></td>
</tr>
<tr>
<td>anttrans24</td>
<td></td>
</tr>
<tr>
<td>antretur24</td>
<td></td>
</tr>
<tr>
<td>antkund24</td>
<td></td>
</tr>
</tbody>
</table>

### Data

- Collapse (sum) `ord_oms` for period 2:
  - `ord_oms` = `ord_oms`
  - `nykund_oms` for period 2:
    - `nykund_oms` = `nykund_real`
  - `oldkund_oms` for period 2:
    - `oldkund_oms` = `oldkund_real`

- Oldkund for period 2:
  - `oldkund_real` = `oldkund_real`

- Antretur for period 2:
  - `antretur` = `antretur`

- Antkund for period 2:
  - `antkund` = `antkund`
antretur`utvperiod2'=antretur ///
  (count)
antkund`utvperiod2'=custno ///
  (min) min_date ///
  (max) max_date, by
(nyk_cohort)
  if `i'==0 {
    replace
    }
  else if `i'>0 {
    append using
      "`babydata1006'"
    save
      "`babydata1006'"
    replace
      "`babydata1006'", replace
  }
}
if mofd(`dat23')<=mofd(max_kopdatum) {
  display `dat23'
  use "`babydata1003'", clear
  merge 1:m custno using "`babydata1001'",
  keep(match)
  format %tMCCYY-NN nyk_cohort ord_mon
  keep if ord_mon>mofd(`dat0') &
  ord_mon<=mofd(`dat23') & ///
  nykund!=1
  if _N>0 {
    collapse (sum) ord_oms nykund nykund_real
    nykund_noll ///
    antretur=oms_fix ///
    min_date=ord_date ///
    ord_mon<=mofd(`dat23') & ///
    keep(match)
    format %tMCCYY-NN nyk_cohort ord_mon
    keep if ord_mon>=mofd(`dat0') &
    ord_mon<=mofd(`dat23') & ///
    ord_oms<=0
    replace oldkund=0 if oldkund==.
  }
  oldkund_noll!=1
  replace oldkund_noll=0 if oldkund_noll==.
  generate oldkund=1 if nykund==0
  generate oldkund=0 if oldkund==.
  generate oldkund_noll=1 if oldkund==1 &
  oldkund_noll!=1
  replace oldkund_real=0 if oldkund_real==.
  generate oldkund_real=1 if oldkund==1 &
  nykund !==
  collapse (sum) ord_oms`utvperiod3'=ord_oms ///
    nykund_oms oldkund_oms
  oldkund ///
  generate oldkund_oms=ord_oms if nykund==0
  generate nykund_oms=ord_oms if nykund==1
  collapse (sum) ord_oms`utvperiod3'=ord_oms ///
    nykund_oms oldkund_oms
  nykund ///
  oldkund ///
  oldkund_real`utvperiod3'=oldkund_real ///
  oldkund_noll`utvperiod3'=oldkund_noll ///
  anttrans`utvperiod3'=anttrans ///
  antretur`utvperiod3'=antretur ///
(count) 
antkund`utvperiod3'=custno /// 
(min) min_date /// 
(max) max_date, by 

(nyk_cohort)

    if `i'==0 {
        save "`babydata1007'",
        replace
    }
    else if `i'>0 {
        append using
        "`babydata1007'",
        save
        "`babydata1007'",
        replace
    }
    local dat1=dofm(mofd(`dat1')+1)
    local i=`i'+1

tempfile babydata1002ack
while `i'<=`steps'-1 {

use "`babydata1004'",
clear

keep nyk_cohort ord_oms0 antkund0 anttrans0 antretur0 //
    nykund_real0 nykund_noll0
merge 1:1 nyk_cohort using "`babydata1005'", keep(match master) //
    keepusing(ord_oms3 antkund3 anttrans3 antretur3 //
        oldkund_real3 oldkund_noll3)
drop _merge
merge 1:1 nyk_cohort using "`babydata1006'", keep(match master) //
    keepusing(ord_oms12 antkund12 anttrans12 antretur12 //
        oldkund_real12 oldkund_noll12)
drop _merge
merge 1:1 nyk_cohort using "`babydata1007'", keep(match master) //
    keepusing(ord_oms24 antkund24 anttrans24 antretur24 //
        oldkund_real24 oldkund_noll24)
drop _merge
outsheet using "`katalog'/baby13002b_`ver'.txt", replace

generate ord_mon = nyk_cohort

tempfile babydata1008
save "`babydata1008'", replace /* Sparar utvärderingen av cohorter */

*******************************************************
* Här beräknas ackumulerat antal kundnummer *
* Samma antal perioder som anges ovan av STEPS *
*******************************************************

local i=0
local churn=12 /* Antal månader av icke-köp som definierar churn */
tempfile babydata1002ack
while `i'<=`steps'-1 {
local dat0=dofm(mofd(min_kopdatum)+`i'+1) /* Första dagen
nästa månad */
local dat0=dofm(mofd(`startdat')+`i') /* Första dagen
nästa månad */
use "`babydata1001'", clear /* Data per kund från ovan */
keep if ord_date<`dat0'
drop ord_mon
generate ord_mon=mofd(`dat0')-1
format ord_mon %tmCCYY-NN
collapse (sum) ord_oms nykund ///
   (count) anttrans=ord_oms ///
   (min) min_date=ord_date ///
   (max) max_date=ord_date ord_mon, by
   (custno)
generate aktivkund=1 if mofd(max_date)>(mofd(`dat0')-1-
`churn')
replace aktivkund=0 if aktivkund==.
collapse (sum) ack_oms=ord_oms ack_nykund=nykund ///
   ack_anttrans=anttrans aktivkund ///
   (count) ack_kundnum=custno ///
   (min) ack_min_date=min_date ///
   (max) ack_max_date=max_date, by
   (ord_mon)
if `i'==0 {
   save "`babydata1002ack'", replace
}
else if `i'>0 {
   append using "`babydata1002ack'",
   replace
}
local i=`i'+1
}
format %15.0g ack_oms
sort ord_mon
keep ord_mon ack_oms ack_nykund ack_anttrans aktivkund ack_kundnum ///
   ack_min_date ack_max_date
merge 1:1 ord_mon using "`babydata1002mon'", keep(match)
drop _merge
* keep if ord_mon>(mofd(max_kopdatum)-`steps') /* Samma antal månader
som ovan */
outsheet using "`katalog'/baby13002c_`ver'.txt", replace
merge 1:1 ord_mon using "`babydata1008'", keep(match)
drop _merge
outsheet using "`katalog'/baby13002d_`ver'.txt", replace
Appendix D Rules creating report tables and graphs

Compare enclosed kundX13002d_rapport_v301 sheet 1. This sheet handles the input imported from the Strata code. Marketing costs (3.5.7) will be added.

Will translate fields if you want them in English.
### Appendix E - Product language versions

Note. This table is just an idea. It can be build in any way preferred to achieve an effective system operation.

<table>
<thead>
<tr>
<th>Product</th>
<th>Product version</th>
<th>Language (ISO)</th>
<th>Text section identifier</th>
<th>Localized text</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Biz</td>
<td>Basic</td>
<td>SWE</td>
<td>[code being the same for all languages]</td>
<td>Customer Value New Biz Report hjälper dig att undersöka nya kunder - deras försäljningsutveckling och köpbeteende över tiden. Det finns goda … … …</td>
</tr>
</tbody>
</table>
Appendix F - pdf example of the New Biz product
Enclosed separately.
Appendix C

Predictive Report FRD
Customer Value Sweden AB

Customer Value Prediction

Functional Requirements Document (FRD)

Author
Customer Value Sweden AB
Ove Svenneke

Last Edited Date
2015-02-23

Version Number
0.2

Document Status
Incomplete Draft
1 Version history 3
2 About this document 4
2.1 Purpose 4
3 Functional Areas 6
3.1 General aspects 6
3.1.1 Scalability 6
3.1.2 System expandability 6
3.2 System enhancements 7
3.3 (New) Functions 7
3.4 Data 9
3.5 Predictive analysis 10
3.5.1 Input to system 10
3.5.2 Calculation process 11
3.5.3 Statistical software interaction overview 12
3.6 Duplicate elimination [optional] 16
3.7 Delivery to customer 16
3.7.1 Delivery File 16
3.7.2 Delivery Note 17
3.8 Analysis 17
3.8.1 Online 17
3.8.2 Status Report [not yet defined by CV] 18
3.8.2 Performance Improvement (change) Report 18
# Version history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>By</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2015-01-16</td>
<td>OS</td>
<td>Incomplete, draft</td>
</tr>
</tbody>
</table>
2 About this document

In this document

“CV” is short for Customer Value (our brand) or Customer Value Sweden AB (the company),

“customer” refers to customers/clients to CV,

“consumers” refers to customers of a CV customer\(^1\).

“RPF”, Repurchasing Forecast, is the objective of the CV business. It predicts future purchases by individual consumers as a probability (percent).

“RPF model” refers to a mathematical formula that with parameter inputs to this formula and calculated purchases data used to compute RPF.

“Pidon” is a classification of consumers according to loyalty and changing purchasing behaviour.

Customer Lifetime Value (CLV) is a value expressing the future purchasing value for a certain consumer.

2.1 Purpose

This Functional Requirement Document is designed to outline each functional area of the project, detailing its requirements. It is however not a formal / technical specification requiring methods and how the system should be built.

We appreciate suggestions for improvements not covered in this FRD that can further enhance the system functionality or performance.

The project this FRD describes should be developed on / integrating an existing system developed in 2014\(^2\) including data upload, data validity control, database and product (“report”) download functions.

\(^1\) “Consumer” normally is a more general description. In this document it is used for individual customers of a specific CV customer to avoid confusion.

\(^2\) FRD Customer Value Production system; IT Product Realizations Process towards Marketing Consultancy Automation, KTH Master Thesis by Georgios Paralykidis.
The current project aims to fulfill the need for a full scale system that can deliver functions for

- upload / download functions and db amendments to handle prediction parameters produced manually by a statistical software,
- produce consumer purchasing behaviour predictions, RPF, for different customers based on prediction parameters,
- produce downloadable result files, delivery reports,
- produce an online graphics report system to be used by customers.

RPF is the core offering by CV. We therefore need a stable and scalable system which means that the present production routine needs to be understood, but also - and perhaps chiefly - the future business expansion objectives of CV.

As the information handled by CV is sensitive - from a CV, consumer and customer business aspect we need security to be considered. Security issues are not in themselves a part of this project; the system should however be designed and developed with security needs in mind.
3 Functional Areas

3.1 General aspects

3.1.1 Scalability
The data volumes will grow in conjunction with CV growth:

- CV will need to have large number of customers building a substantial company and defending it’s market shares.

- Today CV handles rather small customer bases (<150,000 consumers). Attracting larger companies CV needs to handle bases with millions of consumer accounts.

- CV if successful will need to add functions to handle regional or local offices for marketing and administration.

3.1.2 System expandability

3.1.2.1 Localization, consumers [not required in this version]
As CV customers are selling to several markets and are accepting other currencies than SEK a basic currency conversion already has been built into the system. The exchange rates are coded within the system using standard ISO codes.

We will need to be able to retrieve exchange rates

- from an external service to be updated on demand by CV staff (Admin)
- from an external service to be updated automatically according to a schema or
- bespoke added to the system as defined by each customer.

Customers will need to set up their own rules for how currencies shall be handled. For example decide the main (accounting) currency, define groups of countries that should be possible to analyze together.

3.1.2.2 Localization, customers [not required in this version]
We expect customers to want their own “markets” constructed for them including more than one country to be able administer and analyze several country performances combined. Or set limits to the information local staff can access.

Localization affects currency values, output text language etc. We expect however that English will be sufficient as the only language available for the foreseeable future.

This is needs not to be considered in this version but aspects the system design should in coming versions to be able to support.
3.1.2.3 Data
If possible the system should be designed so that modules are accessed using standard field names facilitating future system developments. (Eliminating the use of different names for one logic piece of data; using internal APIs or similar.)

3.1.2.4 Delivery
CV needs a system that can deliver refined data back to customers (push or pull) via APIs (web services or similar) also to be able to retrieve data from customers via APIs. We will need to be able to deliver (push or pull) results via secure FTP.

3.2 System enhancements
3.2.1 Performance
3.2.1.1 Aggregation of transaction data
Today processing a transaction file in the Data Aggregator is quite time consuming. We need to investigate this and find solutions to optimize this process.

3.2.1.2 Real time [not required in this version]
We expect major e-commerce companies to require real time transaction processing and delivery of updated data (one record at a time). This area also needs to be researched and planned for. There is no requirement to solve and develop real time at present, but it should not be hindered by the system solution.

Ideally, in a real-time environment we will be able to deliver updated Pidon class (see below) before the consumer leaves the e-shop site. (Making it possible to adapt the shop’s behaviour towards the customer online.) As Pidon cannot be updated until a purchase is final this implies a few seconds time margin or shorter.

Customers not requiring real time will expect to receive the result file with Pidon classes within hours if not minutes.

3.3 (New) Functions

---

3 The intention is to construct an external API service (web services or similar) to be used by customers to upload data for individual consumers or sets of consumers directly into their CRM or business information systems (accounting systems).

4 Assuming real time customers themselves will store Pidon codes for their consumers we think that real time retrieval of data from CV will not be needed.
The system should be integrated with the present CV system (thus extended and improved). The present solution - CV Production System - was built with the modules described in fig 1 below.

Fig 1. CV Report Generation System (source: CV Report Generation System documentation)

What we need to add primarily is a function to produce consumer purchasing behaviour predictions (RPF) for CV customers.

We also need to add a step one online function to be used by customers analyzing their consumer data to understand the quality of their customer (consumer) base\(^5\).

---

\(^5\) Please note that the present system delivers pdfs to be uploaded by customers. In a future step these reports will be accessible as sets of diagrams and tables that can be modified regarding design and time scope (day/week/month/quarter and the total scope eg jan 2013 - dec 2015).
3.4 Data
In this section upload, database modifications and download are covered.
prediction model parameters
3.4.1 Input file formats
[alternative formats to be used by customer should be allowed]
3.4.2 Back Office amendments
...
3.4.3 Parameters and values

<table>
<thead>
<tr>
<th>Parameter/value</th>
<th>Format</th>
<th>Values occurring</th>
<th>Used by/for</th>
<th>Save</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.5 Predictive analysis
3.5.1 Input to system

*Overview*

CV produces manually, using a statistical application software (Stata), predictive models forecasting it's customers’ consumer future purchasing behaviour (RPF). This is being done for each CV customer as a new customer account is being created.

The RPF *models* are updated infrequently (at present on a yearly basis) to reflect general market changes (as competition, general trends/fashion etc.).

The statistical models are parameterized by the statistical software meaning that it results in numeric values “RPF model parameters” - that are to be used, by the Predictive System, to calculate RPF for individual consumers. The process is described below and in detail in Appendix 1.

Note that there is one set of RPF model parameters for each customer only. There are one RPF value for each consumer and customer. (The consumer can of course belong to several customers, but - as patterns likely varies - the sets describing this consumer will in such cases be different for each customer6.) As consumer purchasing behavior changes over time RPF is regularly updated.

RPF is used to produce output distributed to customers as described below and in appendixes.

*Predictive System*

The output from the statistical software is input to the Predictive System to be used with aggregated purchases transaction date available in CV db. The statistical software output is RPF model parameter sets for individual customers.

The RPF model parameters should change all present consumer RPF values for this specific customer from this point on (recent RPF values only are used and saved).

In a first stage input should be added manually to the system using a new function in the present CV customer administration tool (part of the Production System).

RPF model parameters can be updated at various points in time for different customers.

---

6 Also the IDs for a certain consumer are not the same for different customers.
If possible an automatic transfer of the parameters from the statistical software could be investigated and perhaps implemented. Such an interaction will however not now be necessary as customers at present are rather few. This is not a requirement.

The end result, consumer classification (Pidon), is calculated using RPF. It expresses the future loyalty/stability/trend of the consumer.

The system also should use the information to calculate Customer Lifetime Value (CLV).

It should only be possible to add/change and view RPF parameters by CV staff roles only: CV Admin and CV User7.

Neither parameters, intermediate calculated values or RPF shall be possible for the customer to see or use directly as is within the CV system.

Output to customers is Pidon classification and CLV.

3.5.2 Calculation process
The system shall step by step produce consumer Pidon classes and CLV described in short here but in more detail in appendixes [yet no or preliminary content only].

Values represents one consumer “belonging” to a specific CV customer.

a) RPF
   The probability of repurchases by a certain consumer in percent. Appendix 1.

b) CV group
   Four classes occur: CV1, CV2, CV3, CV4 where CV1 contains the quartile consumers with highest RPF. Appendix 2.

c) Pidon8 classification
   The classes characterizes the predicted purchase pattern of a consumer - either these are stable or changing. Appendix 3.

---

7 Compare Customer Value FRD 0.6.1 Appendix p 27. CV User is not yet defined in present Production System.
8 Acronym for Premium, Improvers, Degraders, Others, New Biz
d) Customer Lifetime Value (CLV)

CLV is an estimate of the value of a consumer for a certain e-business (customer). It represents the sum, in currency, of a consumers probable future purchases\(^9\).

Appendix 4.

3.5.3 Statistical software interaction overview

To understand the interaction with the statistical software see Fig 2 below.

---

\(^9\) Traditionally a CLV formula does not include a prediction value. This is a CV enhancement by CLV. As the CLV calculation is amended by a probability factor and based on a average purchase sum of consumers belonging to a CV-group it is not used to indicate the value for a specific consumer - but rather the typical individual value for the consumers classified in a certain group.
Note that data for each CV customer is always handled separately. Data from other customers shall not be used in any way except for the customer that owns the data.

i) Transaction data and RPF model parameters are fetched from the db.

ii) Data is corrected for returned items by consumer. Corrections are updated to the db (if a purchase was registered in an earlier transactions update). [Included already in system]

iii) Test and correction in data for duplicate consumer occurrences. Also here historic data in db is corrected when needed. [Not yet included in the Production system.] Appendix 5.

iv) After new transaction data for a customer have been uploaded and updated the db RPF model parameters and cut-off levels for a certain customer are retrieved from the db and used to calculate RPF for each consumer belonging to this customer. The result is a zero to one value. This is made as described in Appendix 1.

The RPF model parameters produced by the statistical analysis are reused until a new statistical analysis has been made and it’s RPF model parameters have been imported into the db.

v) The consumers are classified using the RPF value into four equally (approximately) big classes CV1 - CV4, where CV1 contains the quartile of consumers with the highest RPF value etc.

vi) CLV is calculated using the average value of purchases by consumers of the customer within CV1 to CV4 respectively. The latter is recalculated on every instant of updated transactions. Appendix 3.

vii) Pidon classifications (in the illustration named “Customer segments”) are computed according to rules using the CV1 - CV4 groups to reflect customer purchasing behavior patterns. The rules appears from the colour pattern in the upper part of Fig

---

10 This occurs according to a plan/schema or when the customer decides to do it. The Predictive System thus needs to “know” when an upload and successful update to the db has been made. The same goes for the RPF model parameters, there is no specific time for them to be updated into the db.
The following classifications are used (descriptions here are simplified):

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Premium</td>
<td>Customers with high RPF (CV1) for more than one period (month). A customer using the optional Premium Plus category described below may want to exclude “Premium Plus” consumers from the “Premium” group in deliveries etc.</td>
</tr>
<tr>
<td>Premium Plus</td>
<td>Consumers with very high RPF value. This is calculated using a percentage (of the Premium group) decided by the individual customer.</td>
</tr>
<tr>
<td>Improvers</td>
<td>Consumers before with a degrading RPF performance, but now improving</td>
</tr>
<tr>
<td>Degraders</td>
<td>Consumers before Premium but now with degrading RPF performance</td>
</tr>
<tr>
<td>Others</td>
<td>Consumers with low RPF. A customer using any of the optional Other categories described below may want to exclude them from “Premium” in deliveries etc.</td>
</tr>
<tr>
<td>Others Minus</td>
<td>Consumers belonging to the Others group but with very low RPF. This is calculated using a percentage (of the Others group) decided by the individual customer.</td>
</tr>
<tr>
<td>Others Once</td>
<td>[this is a suggested category not definitely decided]. Consumers belong to the Others group having bought once only (one time shoppers).</td>
</tr>
<tr>
<td>New Biz</td>
<td>Customers not before appearing in data.</td>
</tr>
</tbody>
</table>

Individuals in a customer’s consumer file with zero RPF are not Pidon classified and considered not to be customers to the CV customer. They shall be kept in the db though.

Thus we have two main categories containing consumers with a relatively stable RPF - Premium and Others (including the Plus/Minus sub categories).

Improvers and Degraders are “change” categories indicating customers that tend to move towards Premium or Others.

viii) The db is updated with recent RPF values (belonging only to the customer for which they have been calculated) and also recent Pidon codes, CLV and CV-group. [finns anledning att behålla CV?; finns anledning att behålla historiska värden?]

ix) Values to be included in a Delivery Note are calculated: Sums, shares and revenue per Pidon and Retention. The latter is a calculation describing the share of New Biz having returned with at least one additional purchase within 12 months. Appendix 7.
x) An output file with all consumers (including non customers) belonging to the customer is produced including Pidon classifications and more. This shall be made available to the customer. See 3.7.1 and specification in appendix 5.

xi) A delivery note document describing the status of the updated data set shall be produced to be included with the output file made available to the customer. Details in 3.7.2. See appendix 6 for details and appendix 7 for an example of the lay-out.

xii) A set of diagrams are produced to be available to the customer online. See details in 3.8.
3.6 Duplicate elimination [optional]
Customers having forgotten their account credentials sometimes create new accounts when logging in to a e-shop they have favored before. This in a negative way affects the loyalty assessments, that is the Pidon classifications and analytic descriptions of loyalty development and behavior changes in the customer base.

We need to correct the data before Data Aggregation (compare diagram in 3.3). This is done using consumer contact data - if available - as name and e-mail address. However this is not possible to do 100 % due to lack of data or changes the consumers have made by moving or changing their e-mail accounts, common names etc.

What needs to be done is matching contact information in new records with the historic information.

When matching occurs a new constructed customer ID will be used for all matching transaction records. The affected historical records shall therefore get an alternative ID as well as the new transaction record making the time series as correct as possible in this respect.

If data does not contain contact information both for new and historic transactions a matching process can of course not been made. But when historic transactions to a degree contains contact information (although not necessarily complete) should be made.

Rules: see Appendix 5.

3.7 Delivery to customer
3.7.1 Delivery File
The delivery file contains a title header and all consumer records (including non customers to the CV customer ??) belonging to a customer with Pidon codes.

[kommentar: raderat CLV] This should be produced as the calculations have been completed and the database updated.

Delivery files should be kept in the database and possible to retrieve by CV Admin and CV User, that is also historic files should be saved but not be available online to customers. CV Admin and CV User should be able to erase files.

Only the

File format
The file shall be a text file and - not required - xls format. It will include alpha numeric characters. It need to handle non ASCII 7 characters and comply with text formats used for other purposes within the CV system.

The customer should be able to choose between tab and semicolon separated files (and if included xls).

*File transfer*

The recent file shall be available to the customer via web browser interface download in a similar fashion as New Biz Report.

Details in Appendix 6.

3.7.2 Delivery Note

The job of the Delivery Note is to inform the customer of basic facts of the file. These are

- the number of records included distributed per Pidon
- calculated month revenue per Pidon group
- share of sales pie diagram
- retention bar graph

Details in Appendix 7.

3.8 Analysis

3.8.1 Online

The purpose of online analysis is to give customers the opportunity to by graphs and tables review customer loyalty facts and how customer relations develops over time.

Customers should be able to choose different subjects (graphs/tables) as standard reports to be reviewed repeatedly online, export to slide show software or in downloadable pdf documents.

If possible customers should be able to adapt graphs/tables by choosing time frames (length of period visualized), select parameters shown (including, excluding parameters from a standard set; e.g. display Premium excluding Premium Plus and the latter separately).
We should [not required] - in a coming phase - make it possible for customers to add information online to the analysis - for example marketing costs for new customer acquirement costs, discounts costs.

3.8.2 Status Report [not yet defined by CV]
The Status Report is intended to give a first view of customer (consumer) base quality for new CV customers. But it will also be a “where-we-are-now” document that can be used by customers internally working with and discussing customer relations aspects.

It answers questions as:

- How many of our customers are loyal; how many are not?
  – Answered by Pidon distribution facts.

- From where comes the money?
  – Answered by Pidon relative to revenues.

- What is it worth (what may it cost) keeping/defending a customer relation?
  – Answered by CLV per Pidon group.

- How good are we in strengthen relations with slow buying customers?
  – Answered with information regarding Improvers vs Others.

- Do we win or lose business when customer change their loyalty (for the better or worse)?
  – Answered by sales per Improver vs. Degrader

- How many new customers become loyal?
  – Answered by New customer cohort and Pidon analysis.

- How do we perform over time?
  – Answered comparing current performance with historic.

3.8.2 Performance Improvement (change) Report
The Change Report gives customers an overview showing how good they are in using Pidon to improve customer relations. Sort of meta statistics.

- How good are we in keeping customers loyal over time?
- How good are we in turning Improvers into Premium over time?
- What is the share of Others having improved over time?
• Describe customer losses over time! Compare with new customer acquirement; do we win or lose? What is the cost of losing a customer having to replace her with a new?
Appendix 1

RPF consumer value calculation

**Definitions**

Report\_month = the last month where we have transaction data to the last day

Dat0 = the first day (day number) in the following month (the month after report\_month)

**From transaction data, calculate variables for each customer ID:**

First, calculate the basic RFM-variables:

- **Recency** = the number of days between Dat0 and the last purchase date
- **Frequency** = the accumulated number of purchase transactions
- **Money** = the accumulated order sum of the purchase transactions (after adjusting for returned items)

Then, calculate the following variables, which are the variables used in the prediction model:

- **Lrecency** = LN(recency)
- **Lfrequency** = LN(frequency)
- **Lmoney** = LN(money)
- **Lrecency2** = Lrecency\(^2\)
- **Nykund1** = 1 if the customers first purchase transaction is during Report\_month
- **Nykund2** = 1 if the customers first purchase transaction is during Report\_month-1
- **Nykund3** = 1 if the customers first purchase transaction is during Report\_month-2
- **Retur1** = 1 if purchase transaction during the Report\_month have been returned
- **Retur2** = 1 if purchase transaction during the Report\_month-1 have been returned

**General form of the prediction model:**

The probability of repeat (future) purchase is calculated by a logistic model. In general, the model is defined in the following way.

First, let

\[ X\beta = \beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n \]

Where:

\( \beta_0, \beta_1, \ldots, \beta_n \) = the parameters of the model, estimated by logistic regression in STATA
x_1, \ldots, x_n = \text{the variables of the model, defined/calculated according to the previous section}

Then, the probability of future purchase is defined as:

\[ p_{\text{hat}} = \frac{e^{X\beta}}{1 + e^{X\beta}} \]

**For a new client, receive the model parameters estimated by STATA:**
The general form of the prediction model is given by the previous section. For each client, the parameters of the prediction model are estimated by logistic regression in STATA. The estimated parameters specific to the client are given by a report from STATA according to appendix 1.

Thus, the following table (se appendix 1) contains the estimated parameters.

<table>
<thead>
<tr>
<th>Variable</th>
<th>active</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrecency</td>
<td>.17145407</td>
</tr>
<tr>
<td>lfrequency</td>
<td>.97813961</td>
</tr>
<tr>
<td>lmoney</td>
<td>.14136128</td>
</tr>
<tr>
<td>lrecency2</td>
<td>-.06421893</td>
</tr>
<tr>
<td>nykund1</td>
<td>-.26467591</td>
</tr>
<tr>
<td>nykund2</td>
<td>-.16708073</td>
</tr>
<tr>
<td>nykund3</td>
<td>-.10030814</td>
</tr>
<tr>
<td>retur1</td>
<td>.09455065</td>
</tr>
<tr>
<td>retur2</td>
<td>.07824615</td>
</tr>
<tr>
<td>_cons</td>
<td>-.20073245</td>
</tr>
</tbody>
</table>

The parameter \( \beta_0 \) is given by “_cons” in the table. \( \beta_1, \ldots, \beta_n \) are given by the other rows in the table (where the name of each parameter corresponds to the defined/calculated variables according to the previous section).

Given the estimated parameters for each specific client, the probability of future purchase (\( p_{\text{hat}} \)) could be calculated for each customer, using the general logistic equation and the defined/calculated variables for each customer ID.

**Probability levels for creating four CV-groups**
When the probability for future purchase (\( p_{\text{hat}} \)) is calculated for each customer ID, the customers could be grouped in four CV-groups. The probability limits are also given by the report from STATA, se appendix 1.

\[ \text{plim18} = .21875124 \]
\[ \text{plim12} = .12639561 \]
\[ \text{plim6} = .07339017 \]

Thus, the CV-groups are defined as:

CV-group=1 if \( p_{\text{hat}} > \text{plim18} \)
CV-group=2 if \( \text{plim12} < p_{\text{hat}} \leq \text{plim18} \)
CV-group=3 if \( \text{plim6} < p_{\text{hat}} \leq \text{plim12} \)
CV-group=4 if phat<= plim6

Appendix 1.1 : Parameters from STATA

<table>
<thead>
<tr>
<th>Variable</th>
<th>active</th>
</tr>
</thead>
<tbody>
<tr>
<td>lrecency</td>
<td>.17145407</td>
</tr>
<tr>
<td>lfrequency</td>
<td>.97813961</td>
</tr>
<tr>
<td>lmoney</td>
<td>.14136128</td>
</tr>
<tr>
<td>lrecency2</td>
<td>-.06421893</td>
</tr>
<tr>
<td>nykund1</td>
<td>-.26467591</td>
</tr>
<tr>
<td>nykund2</td>
<td>-.16708073</td>
</tr>
<tr>
<td>nykund3</td>
<td>-.10030814</td>
</tr>
<tr>
<td>retur1</td>
<td>.09455065</td>
</tr>
<tr>
<td>retur2</td>
<td>.07824615</td>
</tr>
<tr>
<td>_cons</td>
<td>-2.0073245</td>
</tr>
</tbody>
</table>

. scalar list
plim18 = .21875124
plim12 = .12639561
plim6 = .07339017

. log close
name: <unnamed>
Customer Value Predictive FRD 0.1

log: C:\Users\Dataone\Documents\Kund0004\temp15001a.smcl
log type: smcl
closed on: 18 Feb 2015, 21:36:39

-------------------------------------------------------------------

log: C:\Users\Dataone\Documents\Kund0004\temp15001a.smcl
log type: smcl
closed on: 18 Feb 2015, 21:36:39

-------------------------------------------------------------------
Appendix 2
CV grouping rules

Appendix 3
Pidon rules

Appendix 4
CLV calculation

Appendix 5
Duplicate reduction method

Appendix 6
Output to customer, file specification
Appendix 7

Delivery note lay-out

Company AB
Period analyzed Oct 2014

Distribution of Pidon groups

<table>
<thead>
<tr>
<th></th>
<th>PREMIUM</th>
<th>IMPROVERS</th>
<th>DEGRADERS</th>
<th>OTHERS</th>
<th>NEW BIZ</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>P+</td>
<td>12 382</td>
<td>34 %</td>
<td></td>
<td></td>
<td>43 151</td>
<td>66 %</td>
</tr>
<tr>
<td>Other Plus</td>
<td>1810</td>
<td>3 %</td>
<td>1837</td>
<td>851</td>
<td>37 115</td>
<td>6036</td>
</tr>
<tr>
<td>G-</td>
<td>20572</td>
<td>25 %</td>
<td>3 %</td>
<td>1 %</td>
<td></td>
<td>2 205</td>
</tr>
<tr>
<td>Other Minus</td>
<td>1 837</td>
<td>3 %</td>
<td></td>
<td></td>
<td>56 %</td>
<td>6 %</td>
</tr>
<tr>
<td>TOTAL</td>
<td>2 1106</td>
<td>34 %</td>
<td>2 074</td>
<td>978</td>
<td>49 266</td>
<td>65 533</td>
</tr>
</tbody>
</table>

Month revenue

<table>
<thead>
<tr>
<th>SEGMENT</th>
<th>REVENUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Premium+</td>
<td>3 101 870</td>
</tr>
<tr>
<td>Premium ej plus</td>
<td>4 107 003</td>
</tr>
<tr>
<td>Premium total</td>
<td>7 208 873</td>
</tr>
<tr>
<td>Improvers</td>
<td>3 415 093</td>
</tr>
<tr>
<td>Degraders</td>
<td>0</td>
</tr>
<tr>
<td>Others</td>
<td>51 150</td>
</tr>
<tr>
<td>New Biz</td>
<td>4 703 514</td>
</tr>
<tr>
<td>Summa</td>
<td>15 378 630</td>
</tr>
</tbody>
</table>

Retention

New customers: average share recurring customers, accumulated from month 1 to 12
Appendix 8

Retention.
Appendix D
Sample Transaction Set

<table>
<thead>
<tr>
<th>edgeid</th>
<th>country</th>
<th>ord_date</th>
<th>ord_year</th>
<th>currency</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>SE</td>
<td>2009-04-29 00:00:00</td>
<td>2009</td>
<td>SEK</td>
<td>241.92</td>
</tr>
<tr>
<td>10</td>
<td>SE</td>
<td>2009-12-22 00:00:00</td>
<td>2009</td>
<td>SEK</td>
<td>185.92</td>
</tr>
<tr>
<td>20</td>
<td>SE</td>
<td>2010-09-18 00:00:00</td>
<td>2010</td>
<td>SEK</td>
<td>133.08</td>
</tr>
<tr>
<td>20</td>
<td>SE</td>
<td>2011-09-29 00:00:00</td>
<td>2011</td>
<td>SEK</td>
<td>240.32</td>
</tr>
<tr>
<td>20</td>
<td>SE</td>
<td>2011-10-13 00:00:00</td>
<td>2011</td>
<td>SEK</td>
<td>383.36</td>
</tr>
<tr>
<td>50</td>
<td>SE</td>
<td>2008-03-01 00:00:00</td>
<td>2008</td>
<td>SEK</td>
<td>300.8</td>
</tr>
<tr>
<td>50</td>
<td>SE</td>
<td>2008-10-16 00:00:00</td>
<td>2008</td>
<td>SEK</td>
<td>383.36</td>
</tr>
<tr>
<td>50</td>
<td>SE</td>
<td>2013-05-25 00:00:00</td>
<td>2013</td>
<td>SEK</td>
<td>44.16</td>
</tr>
<tr>
<td>60</td>
<td>SE</td>
<td>2007-06-27 00:00:00</td>
<td>2007</td>
<td>SEK</td>
<td>515.62</td>
</tr>
<tr>
<td>90</td>
<td>SE</td>
<td>2009-10-30 00:00:00</td>
<td>2009</td>
<td>SEK</td>
<td>149.6</td>
</tr>
<tr>
<td>90</td>
<td>SE</td>
<td>2009-10-30 00:00:00</td>
<td>2009</td>
<td>SEK</td>
<td>149.6</td>
</tr>
<tr>
<td>90</td>
<td>SE</td>
<td>2010-04-09 00:00:00</td>
<td>2010</td>
<td>SEK</td>
<td>63.36</td>
</tr>
<tr>
<td>90</td>
<td>SE</td>
<td>2010-09-07 00:00:00</td>
<td>2010</td>
<td>SEK</td>
<td>398.67</td>
</tr>
<tr>
<td>90</td>
<td>SE</td>
<td>2010-09-23 00:00:00</td>
<td>2010</td>
<td>SEK</td>
<td>91.41</td>
</tr>
<tr>
<td>120</td>
<td>SE</td>
<td>2008-09-30 00:00:00</td>
<td>2008</td>
<td>SEK</td>
<td>484.2</td>
</tr>
<tr>
<td>120</td>
<td>SE</td>
<td>2008-10-23 00:00:00</td>
<td>2008</td>
<td>SEK</td>
<td>766.72</td>
</tr>
<tr>
<td>120</td>
<td>SE</td>
<td>2008-12-24 00:00:00</td>
<td>2008</td>
<td>SEK</td>
<td>0</td>
</tr>
<tr>
<td>250</td>
<td>SE</td>
<td>2013-07-20 00:00:00</td>
<td>2013</td>
<td>SEK</td>
<td>0</td>
</tr>
<tr>
<td>280</td>
<td>SE</td>
<td>2013-05-15 00:00:00</td>
<td>2013</td>
<td>SEK</td>
<td>222.72</td>
</tr>
<tr>
<td>280</td>
<td>SE</td>
<td>2013-05-16 00:00:00</td>
<td>2013</td>
<td>SEK</td>
<td>-222.72</td>
</tr>
<tr>
<td>280</td>
<td>SE</td>
<td>2013-05-16 00:00:00</td>
<td>2013</td>
<td>SEK</td>
<td>222.72</td>
</tr>
<tr>
<td>280</td>
<td>SE</td>
<td>2013-06-12 00:00:00</td>
<td>2013</td>
<td>SEK</td>
<td>150.4</td>
</tr>
<tr>
<td>280</td>
<td>SE</td>
<td>2013-07-02 00:00:00</td>
<td>2013</td>
<td>SEK</td>
<td>312.32</td>
</tr>
<tr>
<td>1800</td>
<td>SE</td>
<td>2005-05-14 00:00:00</td>
<td>2005</td>
<td>SEK</td>
<td>0</td>
</tr>
<tr>
<td>100300</td>
<td>SE</td>
<td>2006-10-21 00:00:00</td>
<td>2006</td>
<td>SEK</td>
<td>2122.16</td>
</tr>
<tr>
<td>100310</td>
<td>SE</td>
<td>2004-11-23 00:00:00</td>
<td>2004</td>
<td>SEK</td>
<td>646.4</td>
</tr>
<tr>
<td>100310</td>
<td>SE</td>
<td>2004-12-21 00:00:00</td>
<td>2004</td>
<td>SEK</td>
<td>617.2</td>
</tr>
<tr>
<td>100310</td>
<td>SE</td>
<td>2005-03-25 00:00:00</td>
<td>2005</td>
<td>SEK</td>
<td>663.2</td>
</tr>
<tr>
<td>100310</td>
<td>SE</td>
<td>2005-06-08 00:00:00</td>
<td>2005</td>
<td>SEK</td>
<td>608.8</td>
</tr>
<tr>
<td>100310</td>
<td>SE</td>
<td>2005-06-09 00:00:00</td>
<td>2005</td>
<td>SEK</td>
<td>517.2</td>
</tr>
<tr>
<td>100310</td>
<td>SE</td>
<td>2005-08-19 00:00:00</td>
<td>2005</td>
<td>SEK</td>
<td>668.2</td>
</tr>
<tr>
<td>100310</td>
<td>SE</td>
<td>2006-08-22 00:00:00</td>
<td>2006</td>
<td>SEK</td>
<td>794.2</td>
</tr>
</tbody>
</table>
Appendix E

Interview Notes
Process

- Cooperation problems based on each team member's character (other colleagues, etc.)
- Lack of specs.
- Other resources are minor (i.e., PC server...)
- Testing processes and team specializations found in military projects lead to poorly tested systems and lower risk.
- Proactive ness is important especially in infrastructure.
- Supervisor 3 should assign tasks in a more optimized way (based on time restrictions and task dependencies).
- Time schedule pressure = leads to lower quality.
- Simple things take more time than they should due to bad design.
- Bad management based on false perceived safety = as long as people work long hours we are OK.
- Team overload does not allow for improvements.
- Customer liking vs high-quality system?
- Improving on a bad code base will make improvements lost if a rearchitecture happens.

Index

- Split computations among multiple servers (issues: dum....)
- OpenGL not good for complex math models.
Integration with systems that are out of our control.

Lack of processes caused trouble — leads to devs getting used to wrong practices.

Lack of formal doc. and communication schemes (GIT).

Communication should be the first step for architecture (meetings etc.).

Code review should be a mandatory step to reduce risk.

Design should be the job of an architecture team. Others are good devs and others are good cracks. Architect, architect, architect.

Time and determination. Slow things down. Slow moving processes are not favored.

Meetings to discuss dev ops + TAC Action.

Training is important for the members of the team. Diffuse knowledge within the team, or use external courses etc. Cooperation create expanded benefits in terms of team.

Composition of Java as a service.

DB abstraction layer for Java.

MySQL and abstraction of DB.

JDBC Spring.
Process:
- Connections lead to many assignments (lack of experience and knowledge).
- Wrong selection method of technologies (missing analysis).
- Hierarchy missing and roles are given in a wrong way.
- The lack of knowledge leads to not understanding existing (serious) problems.
- Testing is necessary especially for larger systems.
- CS is needed = formal dev ops.
- Lack of processes increase time for simple tasks and thus causes.
- Existence of processes allow for evaluation of changes and thus allow for continuous improvements.
- Estimation paper is a good way to reach consensus.
- Meaningless meetings between Dev and Biz.
- Biz gives high importance to cost criteria when evaluating new ideas, which makes non-immediate paying ideas not being implemented.
- Business analysis is missing to bond the Biz and IT worlds.
- Do not micromanage!!! "everyone does everything"
- Specialization is a plus. Do not do "generalization". Do what you're best at to be more productive at it.
- Fear of new techs, usually because of bad design that requires a lot of work to migrate. Thus, a lot of things can go wrong.
- Case study: the IT team lack of knowledge leads to wrong (inefficient) decisions.
- Bad design leads to increased risk. Refactoring is a solution to this.
- Documentation should be more organized.

Indie:
- Independence +
- Traceability +
- Reduces fear of change
- Project specific things
- CS and automate devops

Architecture should allow parallel dev and testing without having to wait for other tasks to finish.
- Specs to TDD is slow although reduces risk.
- Good project management due to better load distribution.
- Process:
  - Business begs to systems translation efficiency
  - Communication with clients = too much time spent.
  - Team members not up to date, overestimation of ones abilities.
  - Toolset missing for proper management and defining time needed.
  - Process in time needed for delivery.
  - Large scale projects with no processes = BOOM!
  - Small teams might benefit into processes.
  - Process reduces communication overheads.
  - ~50% just from communication overheads.
  - Process go hand in hand with architecture.
  - Change project: adaptation to process takes time. Business does not invest.
  - Added bureaucracy.
  - Decreased productivity at first. Denial from team members.
  - Picking in MySQL.
  - Automation tables.
  - Config, Manage DB.
  - JDBC batch size.
- Lack of proper management techniques
- Non-realistic estimations (deadlines)
- Non clear specs
- Leader problems (not capable to lead properly)
- Customer communication problems which lead to business analyst not available to elicit reqs from customers
- Lack of processes (formal) either project management or dev
- Lack of knowledge and expertise leads to avoidance of using proper processes factors that affect: cost, company mentality (pure biz companies think differently than tech companies), experience, size.

Exceptions Exist!

- Inter SW&E project to waste!!
- Politics promote personal preferences, Hierarchy position increases influence
- Team leader has to want change, the team must be on board, time must be available, business must agree based on costs and other factors.

Arch
- Review process with team (Adopt xS Company Wolfs over time)
- Experiment with DBs in Genny (AlongBM)
- Dev team profiles are better distinguished (e.g. Genny dev need not have any business knowledge)
- Better trace management w/feedback from Genny to CWS Core
- Dynamic instance creation by CWS Core, when load increases (Cost-Effect)
- Dependence on people
- Inconsistent processes
- Deadlines short.
- Not enough testing.
- Not enough analysis.
- Staging environment necessary (e.g., UAT).
- Communication process missing, phones are not logged.
- No process to foster but rather (no logging, ad-hoc).
- Absence of architecture leads to individuals acting on personal preference (e.g., selecting low frameworks).
- Uniformity is a plus (in terms of dev env): dockerize, git & docs etc.
- Size and $ matter.
- Good processes vs. more independence from people.
- Formalize, formalize, formalize, formalize!!!
- Proper design, to mandate coding style and.
- Automation (CI...)

- Formal process with models found in underscoring.
- Low dep from technologies.
Process

Team coordination

- Need better assignments based on XP.
- No processes followed
- One man show is dead.

Why no process?

- Lack of knowledge.
- Solutions exist for problems but not used.
- System quality will get the hit in the end.
- SW debt = ?, not understood.

- No focus on quality. Those who get the impact invest in higher quality.

- Invest in people development.
- Team motivation increases. 
- Transition is a problem, people resistance. 
- Politics matter. People don't want to loose important positions in the company.

- It is important to be person independent.

- Team to remove fear.
- Previous SW debt is carried on the next team and things get worse.
- DevOps.

- Arch + Expression + lots of benefits
- Some problems apply as before.
- Start experimenting before you "have to",

- TDD as a way to identify high coupling points in the architecture.
- Information = less errors.
- Parallelism (threads) when processing.

- Peer to peer parallelism?

- Patterns = common language between devs.
- Learning curve = better
- Testing is necessary and important

- CARS

- Pedagog support/ maintenance efforts.
Processes are important

- Lack of process: individuals act on personal expense and bring programs/projects process.
- Lack of budget: no experimentation.
  - Narrow options.
  - Decision making is affected; alternatives are not properly evaluated.
- Business fault: lack of knowledge, no money → Be proactive, and
- IT's fault: do not inform the owner, wrong projects discussed openly.

- Productivity:
  - Most orgs use power position influencing (hard tactics)
  - Ambition (lack)
  - Short time frame, no proactive scheduling
  - Receive more (necessary)

- Politics:
  - "Red!
  - Slow things down for personal reasons (3/4 teams!)"
  - ""

- Rearrangement, learning curve case.
- Process independence of #. Different scales use different processes
  but all should use processes.

Just fine!
Appendix F

NewBiz Report - Generated Sample
Customer Value Newbiz focuses on new customers, sales trends and customer behavior over time.

Group Newbiz includes new customers. It is particularly interesting for the reason that the cost of acquiring new customers can be relatively high.

The report Newbiz is a complement to Customer Value Report, which reports the active customers Köpprognoserna's prediction about how customers will buy in the future. This is made clear by the customers grouped into Premium, improvers, Degraders, Newbiz and Others.

The group Newbiz is a potential for future growth. This of course requires that new customers continue to buy. It is therefore not enough to examine how the new customers purchased on their first visit. An evaluation of the costs of acquiring new customers from search engines or other channels should be considered in the long-term köpputvecklingen. Customer Value Newbiz provide the evidence necessary to make the important evaluation.

The report pages 3 and 4 show the new customer to its business, first reported in the number of customers and then in sales.

The following statements follow the purchasing loyalty of the customers who started buying a particular month. This report presents how they continued to buy for 3, 12 and 24 months after the initial purchase.

The aim is to follow the trend of customers generated at different times. By division is as accurate as monthly are given good opportunities to relate to nykundsgrupp media campaigns, promotions or changes in the competitive situation.

To report should be transparent, it is designed with graphs. The Annex also the precise numerical information used to produce the graphs.
Nykundsackvisition
Displays development-acquisition, and new customers importance for the month sale. The graphs show how many new customers-acquisition generated for the respective month. See Annex Table 1.

The definition of the new customers are the ones who never before bought according to the available data. The diagram (Fig. 1) presents the monthly number of new customers (blue color) which bought First time of the month compared to the number of repeat (red color) that bought the same month. To clarify, show the lower graph (Fig. 2), the monthly rate of new customers as a percentage of the total number of buying customers. See Table 2.
Sales new vs returning customers per month
To show the importance that the new customers have total sales reported here the overall sales performance (amount) Breakdown of new and repeat customers.

The upper diagram (Fig. 3) reported monthly sales of DKK to new customers compared with recurring (red color). The blue color represents the new customer’s first purchase during the month. The yellow color reports these customers additional purchase in the same month. So it is optional to consider the additional (yellow) purchases as sales to new customers or repeat sales. See Annex Table 1.

Again, to clarify shows the lower graph (Fig. 4) the proportion of sales to new customers as a percentage of the total monthly sales. See Annex table 2.
Here is a text about how great it is to check on the average sales ...

**Figure 5**

The five diagrams above gives you safely thoughts and ideas for how you can improve your business. Here are some tips and suggestions on questions to ask in order to get valuable information by examining them.

The influx of new customers - and sales to them - as you expected? If it is less certain periods - what did you then or before then? What happened to the market?

Is the balance between new and repeat customers reasonable? During the early phases means new customers, usually more. If you manage to get customers to come back and buy for a long time, they may repeat customers increasingly importance. They are then simply for an increasing share of sales.

You probably need to bring in new customers to your business will evolve as you wish. But it may be possible to develop the business also with existing customers. First, by selling more to each a. But also by ensuring that they remain customers. It gives a stronger effect of nykunds-raising. Your company develops simply faster. And will probably be more profitable.

It is important to compare what it costs to buy a customer. Also, compare the cost of retaining an existing customer. An important question to ask is how long a customer needs purchase to be profitable.
Sales trends and buying behavior

How the new customers continued to buy are shown in the following diagram where the report describes how the customers who bought the first time a month, nykundscohort, have returned with new purchase within 3, 12 and 24 months respectively. The report thus shows how the customers started to buy a particular month will continue to buy by reporting their purchases after the first month they bought.

The maximum repurchase period of 24 months, it is obviously only to show to the customers buying for the first time during the first twelve months of the combined reported period.

The following diagram (Fig. 6) shows the percentage of customers who come back with purchases as a percentage. All the bars in a particular month, reports the following sales developments regarding the customers started buying it month. The red bar accounts for the proportion who returned with purchase within months. The green shows the repeat purchase within 12 months and the purple within 24 months. See Annex Tables 3a and 3b.

![Figure 6](image)

All new customers in a nykundscohort has obviously done a first purchase during his first month, that's why they qualify as new customers. But as the figure above shows there are relatively few who returned with further purchases during the first three months. This also means that the additional revenue (due to additional purchase) during the first three months is relatively low compared to the initial income from the respective nykundscohort.

By evaluating each nykundscohort example, after three months, we get interesting information that shows the importance of new customers in different periods.

- Overall revenues (after three months) consists of the new customers’ first purchase and the additional revenue generated by the customers that already during the first few months returned with additional purchases.

- Once the evaluation is done already after three months is the new customer first purchase generally for most of the total revenue.

- The new customer repurchase - the amount of the additional revenues - is therefore usually a small part of the total revenue as relatively few customers will come back with further purchase in just three months.
The following diagram (Fig. 7) shows the blue curve buyback share of total revenue in each nykundscohort, when the evaluation is done after three months.

Similarly, the red and green curve the buyback share of total revenue gradually increases in nykundscohorterna when the evaluation is done after 12 and 24 months. Compare table 3c.

Here you can follow each nykundscohort (the customers started to buy a given month) from two aspects. How many purchases they have made and how much (amount) they bought after First month the customer has purchased. Again, the development of 3, 12 and 24 months respectively.

Chart below (Fig. 8) shows the number of purchases. Each point on the horizontal line corresponds to the customers who started buying the month as the label indicates. The lines above the label shows how many purchases precisely those customers have made in each time interval. See Annex Table 4.

The diagram on the next page (Fig. 9) shows the development of sales in DKK. See Annex Table 5. Overall, the diagrams purchasing patterns of customers who started buying at different times. It may be valuable to address the differences with regard to the season and the campaigns that occurred at different times.
The diagrams under the title "Sales trends and buying behavior" aims to be able to evaluate how loyal your new clients will be. The diagram presents what we call "nykundscohorter". This may be a little awkward in the beginning to read them. But more than well worth it to sit in.

Actually, it's simple. Each period (month) on the horizontal axis represents new customers purchasing for the first time that month. For each such period, you will see up to three tasks (charts or points on a line).

In the explanations for the colors used in each diagram, you see what they represent. They differ slightly between the figures.

Wisely is to examine the differences between the different periods.

• Is proportions similar month to month so behave your new customers in the same way regardless of when they started trading. It may suggest that you use the same way to acquire new customers. And that the season will
not affect purchase behavior so much. A conclusion can therefore be - in this case - that casual customers buying such as Christmas shopping is not as important to your business. All this might suggest that all is well. But is it enough good?

- Are there differences between the months so it is worth thinking about why. Is it seasonal variations? Or is it because your activities on various occasions resulted in new clients with differing properties? One trick is to simply draw in what activities you did and when. Perhaps also other events of importance. New competitors. Competitors changing prices and other offers. For example.

Wise also examine how high the bars are compared with one another in Figure 6 (which shows the number of purchase). And the differences in income (Figure 7).

- You see quickly and easily how likely customers started to buy a given month were to buy again. Tend repurchases to be smaller or larger over time?

- Charts 8 and 9 differs by showing purchases per customer in the number of respective DKK.

- Whether you choose to start from the first or second pair of graphs are the important questions you should ask:

  Are you getting enough high repurchase rate to your customers becomes profitable enough .
  - Are there differences between the periods that you can learn from for the future nykundsackvisitionen to become more efficient and profitable?

The questions are important to ask. For it is given - profitable customers also means more profitable business.
Appendix G

Enlarged Model Diagrams

This appendix contains larger versions of all model diagrams that were too big to fit into the regular pages in the report.
Figure G.1: CVS Data model.
Figure G.2: CVA SPA components impact diagram.
Appendix H

Technical Background

This appendix contains an overview of some necessary technical knowledge to understand the concepts behind the ideas discussed in chapters 4 & 5.

H.1 Software Development Processes

As specified in [3], “a process (or process model) is a collection of activities, actions and tasks that are performed when some work product is to be created”. It could be argued that a process model is a recipe that describes how to execute the work of a project, but in the context of software engineering it is more of an archetype pattern than a recipe. An archetype is defined in [30, sec. 1.3] as “a primordial thing or circumstance that recurs consistently and is thought to be a universal concept or situation”. An archetype pattern is defined, also in [30, ch. 1.3], as “a collaboration between archetypes that occurs consistently and universally”. The important aspect of an archetype pattern (not to be confused with the archetype pattern in OO software design) is that it exhibits variability. To clarify, this means that an archetype pattern can be applied partially, instead of in its entirety, if the context to which it is applied makes this necessary [15].

When defining software development processes, we usually see recurring activities with similar, if not identical, goals. For example, some of the software development processes for business software have an activity to communicate with stakeholders. Based on our definition in the previous paragraph, this activity is an archetype. If more than one archetypes appear in each process of a set of processes, we could say that this set follows a specific archetype pattern.

A process framework, discussed in [3, ch. 1.4], is a form of an archetype pattern that defines a set of high level activities (framework activities) that are essential to successfully complete a software engineering project regardless of the application domain it belongs to. In addition to that, there is a set of umbrella activities that apply to the process as a whole and deal with other aspects of it like tracking, execution control, risk management, quality management, performance evaluation, configuration and re-usability. The five framework activities defined are the following:

- **Communication.** This is where we attempt to understand the problem that needs to be solved. The main stakeholders are identified and contacted in order to gather requirements and document them to create a solid basis for the next phases to commence.
• **Planning.** This phase defines the required steps that get us from the requirements definition down to actual work towards the projects goals. Here is the place where the work to be done is identified, estimated (in terms of time and money) and assigned to team members for execution.

• **Modeling.** This is where the blueprints for the software project are created. It can be as formal as a UML model or as informal as a text document. Regardless of the formality used, the goal of this phase is to understand the big picture and get a clearer architectural point of view of the system (or subsystem) that we are dealing with. Whether it deals with requirements analysis or system design, modeling is a tool that helps engineers facilitate better communication and document project artifacts in a more concise way.

• **Construction.** This is where the actual code is written to bring the design to life and testing is performed to detect and correct any defects as early as possible.

• **Deployment.** This is where a completed software part is delivered to the client for evaluation in order to get feedback and make sure the initially set requirements where all satisfied.

Using a *process framework*, like the one described here, allows us to create a customized software development process to satisfy the specific needs of a project or an enterprise as a whole. The risks of using an ad-hoc approach (development without a defined process) are many (see [14]) and make using a defined development process a much more attractive alternative.

### H.2 Modeling & UML

A model is “a semantically closed abstraction of a system” [31]. Modeling is the representation of a system using symbols that have a specific meaning. All the available symbols for modeling (a.k.a. the notation), along with their semantics constitute a modeling language [32]. An example of such a language is UML. The benefit of a modeling language over custom drawings to convey ideas is that a modelling language defines its notations along with their semantics. For example, an oval shape in UML is used to depict system use cases and, as long as we are using UML, it will always mean exactly this. On a custom drawing by a developer, we cannot know what each shape means and even the developer himself might not be able to tell when enough time has passed. A defined modelling language like UML offers consistency and standardized semantics most developers are familiar with [31]. Benefits of modeling include, but are not limited to, improved productivity, easier and earlier evaluation of the system, documentation, experimentation (exploring alternatives), easier maintenance and re-usability [33].

In Helium, we use UML for our modeling needs and more specifically the diagrams and elements shown in Table H.1 alongside their semantic definitions as specified in [31]. In addition, we use some custom elements (custom stereotypes) like the user interface elements, used in our user interface diagrams and the requirements elements, which we use extensively in our requirements management phase. A requirements element is used to model a requirement along with its status and
the user interface element is used to describe a single or a set of user interface components.

<table>
<thead>
<tr>
<th>Element/Diagram</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stereotype</td>
<td>A stereotype is the classification of an element. A stereotype has semantic impact. Certain stereotypes are predefined in the UML; others may be user defined.</td>
</tr>
<tr>
<td>Actor</td>
<td>Actor is a stereotyped type representing an abstraction that lies just outside the system being modeled.</td>
</tr>
<tr>
<td>Use Case</td>
<td>A use case is a set of sequences of actions a system performs that yields an observable result of value to a particular actor.</td>
</tr>
<tr>
<td>Component</td>
<td>A component is a reusable part that provides the physical packaging of model elements.</td>
</tr>
<tr>
<td>Node</td>
<td>A node is a run-time physical object that represents a computational resource, generally having at least a memory and often processing capability as well, and upon which components may be deployed.</td>
</tr>
<tr>
<td>Diagram</td>
<td>A diagram is a graphical projection of a collection of model elements, most often rendered as a connected graph of arcs (relationships) and vertexes (other model elements).</td>
</tr>
<tr>
<td>Use Case Diagram</td>
<td>A use case diagram encompasses actors, use cases, and their relationships.</td>
</tr>
<tr>
<td>Activity Diagram</td>
<td>An activity diagram encompasses states and their relationships, organized by actions, and is used to specify the behavior of an operation.</td>
</tr>
<tr>
<td>Deployment Diagram</td>
<td>A deployment diagram encompasses components, nodes, and their relationships.</td>
</tr>
<tr>
<td>Class Diagram</td>
<td>A class diagram encompasses types, classes, and their relationships.</td>
</tr>
<tr>
<td>Component Diagram</td>
<td>A component diagram encompasses components and their relationships.</td>
</tr>
</tbody>
</table>

Table H.1: Semantics of UML elements and diagrams used in Helium.

Finally, to represent relationships between the different elements in our model we use the relationships shown in Table H.2 as defined in [31]. Additionally, we use custom (stereotyped) relationships such as the trace to allow for model traceability and implements to define relationships between a PIM and a PSM.

**H.3 OOAD & Microservices Architecture**

Object Oriented Analysis & Design (OOAD) is an analysis and design approach that utilizes the object oriented paradigm of programming for implementation, aiming for better communication among stakeholders and higher quality results through modeling. The Helium process, described in Section 4.1, is a derivative of OOAD. Within an OOAD software development process (as described in [11]), an analysis
Appendix H. Technical Background

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship</td>
<td>A relationship is a semantic connection among elements.</td>
</tr>
<tr>
<td>Generalization</td>
<td>A definition of an element subtype.</td>
</tr>
<tr>
<td>Association</td>
<td>An association is a bidirectional semantic connection among instances.</td>
</tr>
<tr>
<td>Dependency</td>
<td>Dependency is a unidirectional using relationship from a source (or sources) to a target (or targets).</td>
</tr>
</tbody>
</table>

Table H.2: Semantics of UML relationships used in Helium.

of the system takes place initially, in an attempt to understand the problem and produce models that define what the system should do (requirements model) as well as what are the different entities that make up the system (domain model). The collaboration of the domain model entities (objects) is what results in the system having the required behaviour.

The way domain objects are organized and interact with each other to produce the desired outcomes is subject to a few principles in OOAD [11]. The most common principle in OOAD is “encapsulate what varies” and along with “code to an interface rather than an implementation” they form the basis of the OOAD development approach. Both of these principles allow for the creation of maintainable, flexible and extensible software [11]. Encapsulation dictates the separation of things that vary into different entities so that changes are contained rather than being distributed throughout the whole application. Coding to an interface makes for a clear separation between implementations and specified interfaces. This way switching between different implementations is easier, making the system more flexible and open to experimentation, further allowing for controlled access to the different data handled by each object ([3] defines this as Information Hiding).

In addition to the basic principles described above, OOAD also follows the Open-Closed Principle (OCP), the Don’t Repeat Yourself (DRY) principle and the Single Responsibility Principle (SRP) as described in [11]. The OCP principle dictates that domain objects should be open for extension and closed to any modifications. That is, once an object is designed and implemented, additional changes can be made by extending its functionality using a different, related object (i.e. subclass or delegation) and not by changing the original object itself. The DRY principle proposes that similar pieces of functionality should be gathered together and reused in all places needed. This way when changes must be made, they will be made in a single place rather than in multiple locations. The SRP principle is quite extensively used in CVS and is the principle that promotes the notion of objects (or components in general) that handle a single responsibility for the system. Usually, this principle is stated as “an object should have a single reason to change”, which effectively means the same thing. Of course, these are not all the principles utilized in OOAD. A more extensive discussion on different kinds of principles for object design as well as the more generic architectural design can be found in [3].

The CVS system is making use of the microservices architectural pattern on a high level to break the system down into subsystems that communicate through a defined API. In a sense, the SRP principle seen above is applied on a subsystem level instead of a domain object level. As described in [34], a microservices architecture is an architecture where the system is decomposed into a set of collaborating ser-
vices that communicate over defined APIs. Essentially, it could be seen as building multiple web applications and integrating them to accomplish the system’s goals as a mush-up of the underlying (micro)services. Taken from [34], Table H.3 shows the pros and cons of such an approach.

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ Microservices are small compared to a monolithically architectured system.</td>
<td>→ Deployment becomes harder since the infrastructure must be managed to run all the different services.</td>
</tr>
<tr>
<td>→ Each service can be developed and deployed independently from the others as well as scale independently.</td>
<td>→ Communication overheads between the instances are increased (especially if a cloud provider like Amazon Web Services (AWS) is being used).</td>
</tr>
<tr>
<td>→ Development team management is greatly enhanced.</td>
<td>-</td>
</tr>
<tr>
<td>→ Improved maintenance and bug detection.</td>
<td>-</td>
</tr>
<tr>
<td>→ Technology stack can change easier than in a monolithic application.</td>
<td>-</td>
</tr>
</tbody>
</table>

Table H.3: Semantics of UML elements and diagrams used in Helium.
Bibliography


Acronyms

**API** Application Programming Interface. 34, 36, 37, 39, 40, 42–44, 170, 171

**AWS** Amazon Web Services. 171

**B2B** Business to Business. 2

**CLI** Command Line Interface. 39

**CSS** Cascade Style Sheets. 37

**CV** Customer Value. i, v, ix, 2–4, 7–9, 13–17, 20, 21, 25, 28, 35, 42, 49–51, 56–58, 61–64, 173

**CVS** Customer Value System. v, ix, x, 7–9, 13, 28, 29, 34–36, 44, 50, 53, 55, 61–65, 164, 170, 174

**DI** Dependency Inversion. 39

**DRY** Don’t Repeat Yourself. 170

**FRD** Functional Requirements Document. 16, 21

**HBAC** Hierarchy Based Access Control. 7, 8, 35

**HTML** Hypertext Markup Language. 37

**HTTP** Hypertext Transfer Protocol. 34, 48, 53

**IT** Information Technology. i, 1, 4, 59

**MVVMC** Model - View - View Model - Controller. 36, 47

**MVW** Model - View - Whatever. 47

**OCP** Open-Closed Principle. 170

**OO** Object Oriented. 4, 34, 167

**OOAD** Object Oriented Analysis & Design. 169, 170

**PIM** Platform Independent Model. 15–17, 47, 63, 169

**POS** Point of Sale. 28
**PSM** Platform Specific Model. 15, 47, 63, 169

**QoS** Quality of Service. i

**RBAC** Role Based Access Control. 7–9, 35, 41

**SPA** Single Page Application. 34, 36, 47

**SRP** Single Responsibility Principle. 170

**UML** Unified Modeling Language. xi, 4, 15, 16, 18, 168–170, 174

**UX** User Experience. 4, 61
Glossary

MySQL View A View in MySQL is practically a stored query being executed every time another query referenced the View name as a table to read data from. 62

project road-map A project road map is a plan of the projects an organization aims at implementing alongside a schedule that defines when each project is to be implemented. Other information such as high level goals, risks and milestones can also be found in a project road-map. i

software development process See software development process model. 1–4, 13, 167

software development process model A development process model is a set of well defined activities, with defined relationships within the process and defined artifacts that each produces. 2, 3, 5, 13, 179

stakeholder A primary stakeholder is an entity (i.e. a person, a business etc.) who has immediate interest on the successful outcome of the a project. 167

STATA As seen on the providers web site (http://www.stata.com): “Stata is a complete, integrated statistical software package that provides everything you need for data analysis, data management, and graphics”. 2, 50, 62, 63

work product Work product is an artifact generated at the end (or during) an activity within a process by the employee that executes it. 167