Working for the customer

Acting as software consultant in a migration project

Master Thesis performed in Computer Science at The Swedish Guide and Scout Association
Mälardalen University
School of Innovation, Design and Engineering
by

Dan Hallin

Västerås 2012

Supervisor:	**Rikard Land**
	Frank Lüders
	Hélena Läck
Examiner:	**Ivica Crnkovic**

Västerås Mar 3, 2012.

Abstract

In this report is described the work of acting as a technical consultant for a large non-profit organization replacing their membership and invoice system through an adaptation of an existing system. The developer of the new system worked with the Scrum method, allowing the customer to do a lot of interaction in controlling the progress of the project.

The technical consultancy was done on the customer’s side and focused on two areas. One was to be an advisor, making translations between customer and developer jargon to make both sides understand each other. The second part was in taking responsibility for the database migration between the two systems. Theory about database migration is applied on the process of mapping out the old database. The quality of the migration was important to the customer and decisions to balance quality and cost are described.

The report finds: Morris Golden Rule #1 of migration: “Data migration is a business not a technical issue” holds. Adding a technical consultant on the customer side seemed to increase understanding between the two parties. The Scrum method put a high workload on the customer, but was suitable for this type of short deadline and imprecise definition project.

Keywords: Migration, Scrum, Software Development, Customer

Sammanfattning

Den här rapporten beskriver arbetet som teknisk konsult för en stor ideell organisation i färd med att ersätta sitt medlemsregister och fakturasystem genom en anpassning av ett existerande system. Projektets deadline hade förkortats och projektet bedrevs under stor tidspress. Utvecklaren av det nya systemet arbetade med Scrum-metoden, vilket gjorde att kunden i stor utsträckning kunde styra utvecklingen.

Konsultarbetet gjordes på kundsidan och fokuserade på två områden. Ett område var att agera rådgivare, genom att göra översättningar av kundens respektive utvecklarens interna språk så att båda sidor bättre skulle förstå varandra. Det andra området var att ta ansvar för datamigreringen mellan de två systemen. Teori kring databasmigrering användes för att kartlägga den gamla databasen. Migreringskvalité var i fokus hos kunden och beslut kring hur kvalite jämte kostnad balanserades beskrivs.

I rapporten dras slutsatsen att Morris Gyllene Regler Nr 1 om migrering: “Data migrering är ett affärssystem-problem, inte ett tekniskt problem” håller. Att lägga till en teknisk konsult på kundsidan verkade öka förståelsen mellan de två parterna. Scrummetoden la en hög arbetsbelastning på kunden, men passsade bra i den här typen av projekt med kort deadline och en oprecis definition av projektet.

Nyckelord: migrering, Scrum, mjukvaruutveckling, kund

Acknowledgements

Many people have helped in different ways during the creation of this work.
There is a few however who deserve special mention.

Hélena Läck at the The Swedish Guide and Scout Association for accepting me as a part of the project lead team and trusting me with the responsibility of the data migration.

Christian Westgaard at Redpill-Linpro whom I worked closely with for migrating the database; his patient expertise taught me most of what I now know about practical database migration.

Rikard Land at Mälardalen University who gave me half a ton (or more) of comments on which I could base this report's layout and structure on.

And there is of course also a special Someone whom without none of this would have been likely to happen.

Contents

1 Introduction

1.1 Background

The Swedish Guide and Scout Association (Svenska scoutförbundet / SSF) had outgrown its current system for managing membership data and invoicing of members. Additional functionality had been patched on to an existing system leading to a haphazard user interface. Administrational problems had arisen as the system did not handle the added functionality in a way such that its users could efficiently use the system.

These main issues had been identified by SSF based on user feedback:

1. It is difficult to find where to enter information and it is difficult to understand what information to enter
2. Information is not saved when entered
3. Creation of invoices is troublesome and time consuming
4. The generation of invoices is prone to manual errors and there is no way of checking for these errors before sending the invoices to the printers
5. Users find it to be very difficult to enter the information for the annual reports required for governments grants
6. It is difficult to create lists of members, for example a printable member roster or address lists
7. It is impossible to generate statistics regarding members
   SSF had employed a specific project leader whose sole responsibility was on the project. The project leader did have a general education on computer systems, but did not have any detailed technical know-how of computer systems.

1.2 Goals of thesis

The purpose is to describe the demands put on the customer when ordering a new system requiring the migration of a live member and invoicing system. It looks at what can be gained from adding a technical advisor who can fill in as a communicator between the customer and the supplier.

The goal of the thesis is to participate as a technical consultant being employed by the customer in the migration project, taking responsibility for the data migration, acting as advisor to the project leader and evaluating the Scrum working method from a customer perspective.

1.2.1 Database migration

The old system was supplied as a web service. SSF did not have access to the internals of that system. The technical advisors goal is to ensure, by assisting the customer and the supplier, that the data can be migrated to the new system with an acceptable level of quality versus cost.

1.2.2 Technical advise

For a customer with little insight into software development, it can be difficult to understand and assess the complexity of software development problems. This can lead to communication problems between supplier and customer.

The advisor can give advice on the level of complexity of certain problems and solutions. A solution that a customer think may be complex may actually be trivial, and vice versa. The advisors role is to give rapid responses on what level of complexity a problem or specific solution falls into.

The goal of the technical advisor would be to fill the gap in technical competence between the customer and the supplier. The question is whether this can help in planning and meetings.

1.2.3 Scrum evaluation

The Scrum software development process involves the customer on a regular basis. This study evaluates the demands which are put on the customer during the development.

1.3 Theoretical concepts

1.3.1 Scrum

“Scrum is an iterative and incremental methodology for software projects and product- or application-development.” [1]. It provides “a framework within which complex products can be developed.” [2].

Scrum is grounded in empirical process control theory upheld by three pillars [2]:

Transparency – *“Transparency ensures that aspects of the process that affect the outcome must be visible to those managing the outcomes.”*

Inspection – *“The various aspects of the process must be inspected frequently enough so that unacceptable variances in the process can be detected.”*

Adaptation – *“If the inspector determines […] that one or more aspects of the process are outside acceptable limits […] the adjustment must be made as quickly as possible to minimize further deviation.”*

Scrum runs in short sprints, typically 2-4 weeks in length. The goal of each sprint is deliver an improved version of the working product which is then demoed to the customer. The Scrum management requires the customer to partake in some specific activities.

1.3.1.1 Roles in Scrum

- Product Owner. This is a single person responsible for managing the Product Backlog, prioritizing what work is to be done to maximize the value.
- Scrum Master – Responsible for ensuring the Scrum]process is understood and followed
- Scrum Team – The developers, controlled by the Scrum Master.

1.3.1.2 Framework items

These are the Scrum items that are more or less visible to the customer:

- Product Backlog – “a prioritized list of everything that might be needed in the product”
- Sprint – “a iterative period of one month or less which delivers an increment of the final product that is potentially releasable.”
- Sprint Backlog – “a list of tasks to turn the Product Backlog for one Sprint into an increment of potentially shippable product”

- Sprint review - At the end of every sprint a meeting is held where the Scrum Team present the functionality that has been done for the stakeholders, this
- Sprint planning meeting – In the beginning of each sprint is held a meeting where it is decided which stories are moved from the backlog into the sprint backlog.

1.3.2 Data migration

As a theoretical guide on data migration is used Morris' book "Practical Data Migration" [3]. It contains practical guidelines, examples and anecdotes on the parts involved in a data migration project. In the project his methods are used, we use his golden rules and compare to the stages he divides data migration projects into.

1.3.2.1 Golden Rules

1. ***"Data migration is a business not a technical issue"***
   "The enterprise understands the meaning and relative value of its data. It is this value that must be preserved and enhanced in the transformation activity that is data migration."
2. ***"The business knows best"***
   "We must acknowledge that we on the project cannot know more about the enterprise rules than the enterprise itself."
3. ***"No organization needs, wants or will pay for perfect quality data"***
   "[…] compromises need to be made once the issues are fully understood. Accepting this in advance means that sensible considered priorities can be drawn up."
4. ***"If you can't count it, it doesn't count"***
   "Well managed projects have measurable deliveries"

1.3.2.2 Project stages

Morris prescribes that you must:

- *"Stick to the golden rules"*
- *"Identify Data store owners and business domain Experts"*
- *"Use Data Quality Rules"*

Morris suggests a data migration project to be split into the following phases. This procedure is used to be able to start the project as early as possible while leaving the dependency on the new system design to as late as possible.

Stage 1: Initiation
Organizational and political activities that increase the chances of success
Stage 2: Legacy Data preparation
"Analyze and fix the problems in the Legacy Data Stores"
Stage 3: Data preparation
"Design and build Transient Data Stores."
"Create the extract, transform and load definitions."
Stage 4: Build, Test and Go Live
Building and testing of load scripts and creation of Data Transitional Rules.
He further describes how a migration project will naturally become an iterative process. It will interact with the development of the target system, leading to design changes as the project moves along.

1.3.2.3 Data mapping

“A Data Mapping is the rule by which one or more data items in a Legacy Data Store will have their values moved to one or more items in the new system database.” He expresses the data mapping relationships in the form “number of source fields : number of destination fields”

- 1:1 is the most common mapping, one source field to one destination field
- 1:M means one source field is split over several destination fields
- M:1 means several source fields have to be merged into a single destination field
- M:M means several source fields are parsed and then split over several destination fields

He describes how the rules of the mapping being made are documented into tables which are signed off by the relevant data stakeholders.

1.3.2.4 Data stakeholders

“A Data Stakeholder is any person within or outside the organization who has a legitimate interest in the data migration outcomes.” Morris identifies a number of stakeholders that need to be allied into a data migration project:

- Data store owner - “has formal responsibility for the quality of the data in the system and for its use”
- Business domain expert - “is up to speed with the way the data source is used now. […] can adequately represent the business domain’s interests.”
- Technical data experts - Knows about “aspects of the system that only the technically expert can know about” such as “file formats, access permissions, interfaces etc.”
- Programme experts - Knows about the target system and its needs
- Corporate data architect – “information management function where corporate data models and data architectural models will be maintained”
- Audit and regulatory experts - Represent the “external auditory and regulatory requirements”
- Data customers - “internal recipients of data”

1.4 Definition of special terms

1.4.1 SSNO

The Swedish SSNO (social security number) [4] consists of 10 digits. It is formatted such that the first six digits are made from the birth date of the person it belongs as: yymmdd. Separated by a hyphen (-) are the four last digits which are arbitrary, making the total SSNO unique for that person. The total format is: yymmdd-xxxx. The final digit is a calculated checksum of the nine first digits. Officially the hyphen is a century decider, changing to a plus (+) sign the day a person turns a hundred years old. In practice two digits are instead added in front of the SSNO denoting the whole year (see Figure 1 – Format of Swedish SSNO).

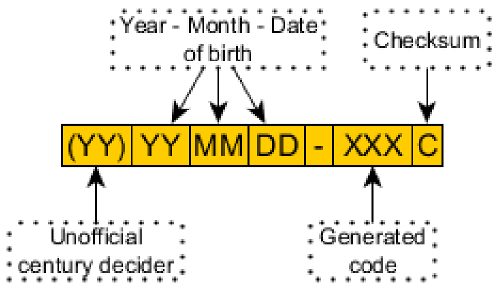
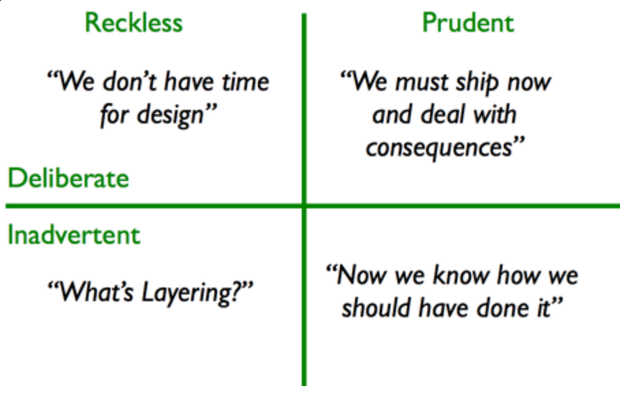


Figure 1 – Format of Swedish SSNO

1.4.2 Technical Debt

Fowler defines technical debt as the difference between the current state of a system and of a system well-designed and with well-documented code. He makes a distinction between different kinds of technical debt dependent on their origin in a quadrant of reckless / prudent and deliberate / inadvertent (see Figure 2).

If not 'repaid' Technical debt generates interest by hampering further development of the system. Fowler suggests that the term Technical Debt is "handy for communicating to non-technical people" [5].



Reckless
Prudent
"We don't have time for design"
"We must ship now and deal with consequences"
Deliberate
Inadvertent
"What's Layering?"
"Now we know how we should have done it"

Figure 2 – Technical Debt Quadrant,

(C) Martin Fowler, used with permission

1.5 Report layout

Chapter 2 contains a description of the old systems functionality and how different parts relate in the migration project. It details problem areas and corresponding goals and solutions for the new system.
Chapter 3 details the plans for ordering a new system and how the purchasing process was done. It lists the project plan and details the components to be developed.
Chapter 4 describes the project activities planned to reach the goals of this thesis.
Chapter 5 describes the studies made inside the project
Chapter 6 describes the activities performed in migrating the database of the old system to the database of the new system.
Chapter 7 describes the activities performed as an advisor in the project
Chapter 8 describes the project organization and the methods and tools that were employed for communication between the customer and the software supplier.
Chapter 9 - 11 contains evaluations of the activities with achievements, lessons learned and possible 'to do' areas.

Chapter 12 summarizes the project, draws conclusions and makes suggestions for improvements in some areas.

2 The old system

The old system consisted of a web service supplied by an external company. It had a tab based navigation system and a basic layout using forms (see Figure 3).

2.1 History

SSF had a long history of using computerized membership management; the first system was deployed in the 1970’s. The current system had been running since 2000 where the data had been imported from the previous system. The current system had undergone a number of major changes during the recent years; the changes were the result of adapting to changing requirements for the right to receive government grants.

2005 - Addition of member confirmation, a module that forced the group administrators to annually enter a validation date for each member’s membership approval.

2006 - Annual group reports for government grants, a module that allowed the group administrator to split the approved members into subgroups.

2007 - Generation of invoices, a module that allowed SSF to invoice the members directly, instead of groups doing it locally and reporting the results.

2.2 Overview

2.3 The system users

There are about 600 groups registered in the system. Several hundred more are expected to join as a result of mergers with other organizations. Each group consists of 5 to 600 members, with an average of 58 members per group.

2.3.1 Members

Members did not normally use, or even know about the system. All members did however have a limited access to the system. If their email was registered in the system, they could retrieve their ‘reference code’, an 8-letter password-like string which could not be changed by the member. Another option was to send a request to the group administrator asking to send the reference code manually. With the reference code the user could log in and look at and update its own data, such as address, telephone number and email address.

The first time a member would usually meet the system would be when they wanted to register to certain events. Some events required the user to enter their ‘reference code’, which they then had to retrieve.

2.3.2 Group leaders

In every group there are a number of main leaders, typically 4-8 persons, with responsibility for a sub-group typically consisting of 5-30 members. They are responsible for keeping track of the members by handling the paperwork and communication with members. They need to register and update the personal information about members, such as addresses, birth dates and contact information. There was no standardized way of handling this input from the members, every group would create its own procedure. The input was typically forwarded to the group administrator to be entered into the system.

The group leaders may be given some administrational privileges, in which case they could directly take on some of the tasks of the group administrators.

2.3.3 Group administrators

A few hundred group administrators which handle the system on a group level, being responsible for the handling of members. Their tasks are to:

- Register and update the information about the group and its members.
- Give system privileges to group leaders.
- Insert data for government grant applications.
- Keep track of the payment status of membership fees.

2.3.4 Organization administrators

A few super administrators which handle the system centrally. They handle invoicing, the creation of new scout groups and act as a call center for members and other administrators.

2.4 User interface weaknesses

The old systems user-interface was a tab based navigation system with a basic layout using forms (see Figure 3). The multiple levels of tabs, with all not being visible at the same time (note the 'return'-button in the top left of the tabs, it returns to another tab row), made for a confusing interface where it was very difficult for the user to find the right the information.

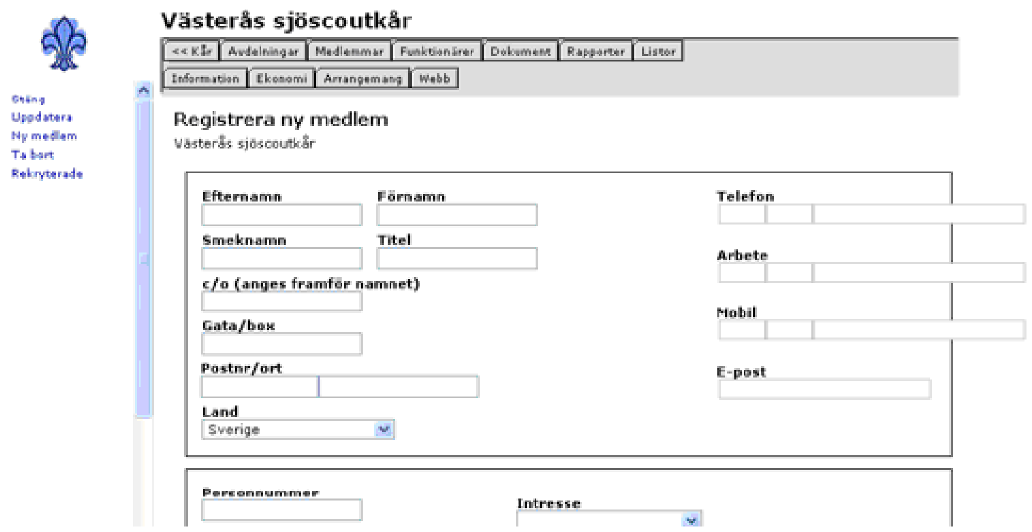


Figure 3 – Layout example of old system

2.4.1 Organization user-interface

2.4.1.1 Problem:

Group administrators have difficulties finding where to update the group's information, resulting in support calls to the organization administrators.

2.4.1.2 Goal:

Create a user-interface which is logical in the placing and naming of functions.

2.4.2 Membership fee user-interface

2.4.2.1 Problems:

The user-interface for setting the members term fees were separated from the user-interface for handling members. Newly created members could thus not have a term fee set. This resulted in invoices with incorrect sums being sent to the member.

The term fee consisted of three parts, the organization fee, the district fee and the group fee. The organization and district fees were only available to the local administrator in an external document. The local administrator thus had to calculate the correct fees based on data in that document and could not see in the system whether the calculation were correct. This could result in invoices with incorrect sums being sent to the members in that group, or worse, negative fees, resulting in an invoice not being sent at all.

2.4.2.2 Goals:

Fees would be mandatory for the administrator to set when approving a membership and the total sum as well as the parts of the calculation would be directly visible when creating fees. Negative sums would be impossible.

2.4.3 Invoicing of membership fees user-interface

2.4.3.1 Problem

The organization administrator was responsible for creating invoices being sent to all members of the system. The creation of invoices was troublesome and time consuming.

If the parameters of the creation of invoices were incorrect this could lead to major errors in the invoices created. The parameters were entered manually and there was no way of checking for errors before sending the invoices to the printers. In the fall of 2009 a manual error led to the creation and sending of invoices to all members which had already paid the membership fee.

When large numbers of invoices were created, the system could fail in creation and the system suppliers had to be employed for the creation of the invoices.

2.4.3.2 Goal

Create a system which could handle large batches of invoices where it was not possible to resend an invoice to a member who had already paid the fee. For extra safeguarding and control it should be possible to view the invoices, both in batches and the individual items before they are sent to the printers.

2.5 Data weaknesses

2.5.1 Missing SSNO

2.5.1.1 Problem:

Three years prior to export a rule had been added to the system to force administrators to enter a members’ social security number (SSNO) on the creation of a new member. Users had found ways to get around this enforcement and about 7% of the members did not have a SSNO defined, half of which had been created after the rule was added.

2.5.1.2 Goal:

The data that is required for the system to handle members must be mandatory to enter when creating a new member.

2.5.1.3 Solution:

The new system got requirements for SSNO being unique for each user, and being a mandatory item to enter in a new user. It further controlled the correctness of the checksum built into the Swedish SSNO.

2.5.2 Incorrect member data

2.5.2.1 Problem:

As members were not aware of the possibility of updating their own information in the system the data would become incorrect as the members for example moved to new addresses or attained new telephone numbers.

2.5.2.2 Goal:

The members should be informed of easy ways of accessing the system to see and change their own information.

2.5.2.3 Solution

The new system got requirements for the users to be able to retrieve their password through a typical ‘request password’ if their email was registered in the system.

2.5.3 Incorrect member statistics

2.5.3.1 Problem:

The statistics system produced illogical results, commonly producing slight offsets. As government grants were based on these statistics these inaccuracies could result in problems in the auditing of the organization.

2.5.3.2 Goal:

The new system would properly keep track of memberships such that logical inconsistencies were not possible.

2.5.3.3 Solution

The reason for the old system producing illogical results was that a change in data would result in overwriting previous data. The new system used versioning tables, any change in the database was stored as a new version of that item and the previous version remained. This made it technically possible to restore and retrieve data such that it was at any previous date or time.

2.5.4 Users accessing the system

2.5.4.1 Problem

If the same email address was registered twice in the system, it was impossible to retrieve the reference code, thus a group admin had to supply it manually. This was often the case when the same person with two memberships, a parent and its child (usually registered with its parents address) or two siblings (with their parents address) were registered. As the reference code was very complex in format, and was unchangeable, it was forgotten by the member and generally had to be resupplied every time it was to be used.

2.5.4.2 Goal

The user should be able to access the system through different methods and should always be able to retrieve its login information through the system.

2.5.4.3 Solution

The new system was designed such that either email or member number could be used to retrieve a password, if the user had an email in the system. In the case where one email was registered on two or more users, an error prompted the user to enter its member number.

2.5.5 Government grant applications

SSF had recently adapted to a new way of applying for government grants. All members had to be specifically assigned into subgroups; this was done by the group administrators, who received input from the group leaders. This had required a new technical solution. It was implemented in such a way that it was separated from the rest of the system, even though it was handling the same data.

The system was difficult to handle, the user interface was divided over several pages. The pages were also prone to raising errors, making attempts at handling the departments futile even for cunning users.

3 Plans for the new system

The old supplier was no longer capable of delivering the services required by SSF. The running costs of the system were also larger than expected. A project leader was employed in June 2009 to research alternative options and gather quotations from suppliers based on requirements gathered from suggestions of the system users. The cost was included in the next budget and was approved by the annual assembly in November 2009. There was a general consensus about the need for a renewed system and no one expressed negative opinions about the cost of a new system.

3.1 Creating specifications for a new system

The creation of specifications for a new system was done by arranging workshops at times and places where a lot of members gathered. Input from these workshops and other sources, such as complaints or suggestions gathered by email were then summarized into a specification. The specification [6] [7] [8] was written in a ‘loose’ form, detailing in general terms the needs of the system.

3.2 Goals of new system

The overall goal [9] of the new system is to decrease the workload for administrators on all levels of the system.

3.3 Choice of supplier

A quotation was sent to different software system developers. SSF came to choose a supplier which was already working on a system for the national scout organization of Norway.
The offer included a strategic alliance with that organization, allowing for shared costs on future development after an initial investment into the already ongoing project.

The supplier was the result of a Norwegian-Swedish merger, thus having personnel localized in both countries. The supplier employed a Scrum working method and was a member of the 'Free Software foundation'.

3.4 Project initiation

3.4.1 Deadlines

A failure of prolonging the contract with the previous system owner shortened the deadline for development considerably (see Figure 4). The system was initially planned to be finished by January 2011. The failure of prolonging the contract meant that the old system would stop its services by 1 July 2010. As a consequence SSF had to prioritize functionality, such that vital requirements were in place when old system was shut down, see the changes required in chapter 3.6.

3.4.2 Contract signing

The customer knew that the chosen supplier was having trouble delivering in time in a similar project based on the same foundation. Since the customer had failed to prolong the contract with the current provider, the deadline of the project was critical; the system would be shut down leaving SSF with no system working.

In the suppliers' contract a standard clause of delay penalty fees was considered too low to be a deterrent for the supplier to deliver on time. Before accepting the contract the customer negotiated a clause which increased the risk for the supplier by furthering the penalty fee substantially.

3.5 Supplier working method

Scrum is a method in the XP family employed by software development companies.

The reason the supplier gave for working with Scrum as a method was:

- Our experience is that the customer becomes more satisfied with the result.
- It focuses on good communication between supplier and customer and assumes a close collaboration between the two parties.
- The strength of the method is that all parties have insight into the project.
- It ensures a delivery in tune with the customer's requirements and expectations.
- It puts emphasis on handling changes instead of following a static plan, making it possible to direct the project to a better result based on the knowledge gained during the project.

3.6 Project plan

The original plan was to launch the new system in January 2011. In October 2009 a failure to prolong the contract with the current system provider meant that the old system would be shut down by 1 July 2010. A revision of the plan was made where the development of the new system was split into a sequence of three development cycles, and only the first was scheduled to be developed prior to launch (see Figure 4).

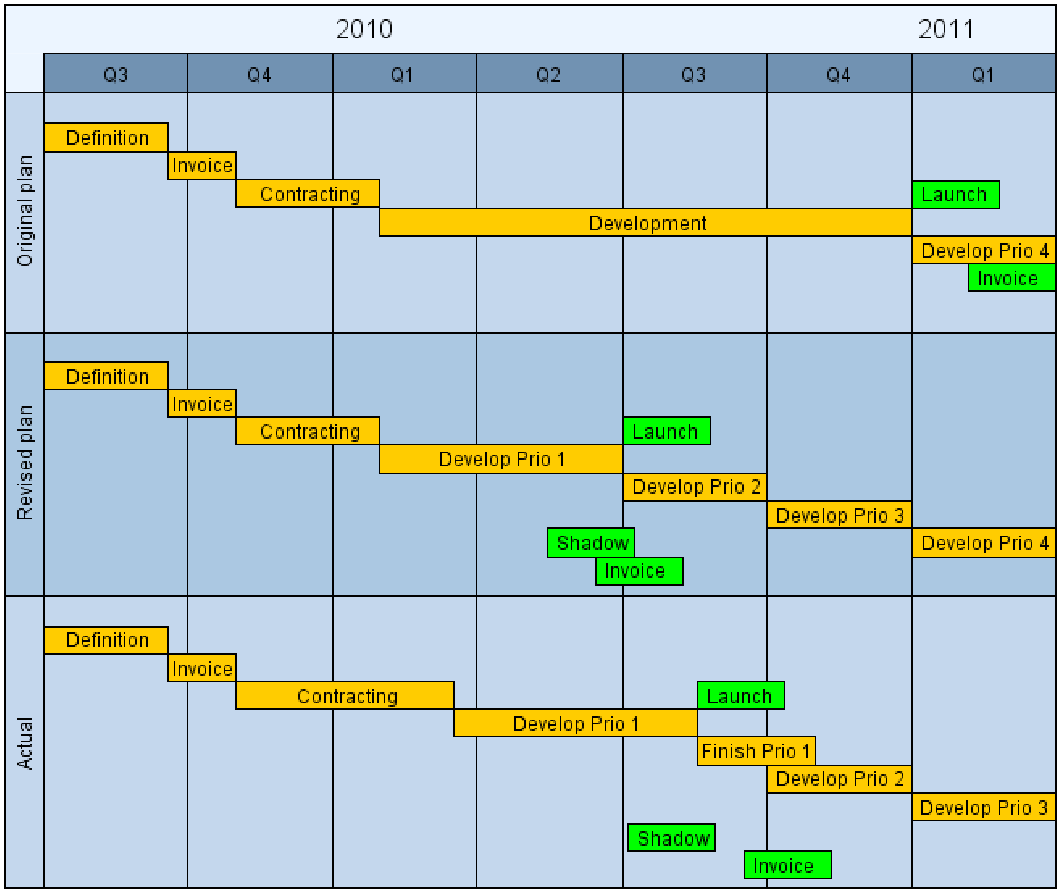


Figure 4 – Three revision of plans

3.6.1 Function priorities

All the planned functions were of importance for the system, thus the functions could not be divided into their priority cycles based on their importance for the overall system. The functions were however used at different times of the year, some functions could thus be divided based on when they were next going to be used in the system. Others were dropped from the original plan for not being of vital importance.

3.6.2 System functions in first development cycle

3.6.2.1 Membership data

The member is the base unit of the system. The system keeps track of member data, such as names, birth data and contact information. The data is inserted and maintained by administrators on all levels.

3.6.2.2 Organization data

The system is divided into a hierarchical structure of organizations. The system keeps track of each organizations data and which members are in each organization. Each member is a member of one or more organizations. The previous system did not support multiple memberships and thus member data was duplicated for each membership. Organizations are handled by administrators on their respective levels.

3.6.2.3 Invoicing

Used by the organization administrators for creating invoices to the members.
The next invoicing round was to be done in September 2010, but for SSF who wanted to avoid a repetition of earlier errors, it was of major importance that a smaller scale test was made in advance. This function thus had to be put into the first development cycle. All functions on which invoicing was dependent was thus also put into first priority.

3.6.3 System functions excluded from first development cycle

Originally all functions of the old system were planned to be implemented in the new system in one big development cycle. After the deadline had been cut short, a prioritization of system parts to be transferred was necessary. All functionalities that were not required for basic operation and invoicing were cut from the first development cycle.

3.6.3.1 Event management and registration

A module that allows administrators to create events and handle incoming member registrations to those events. Only sporadically used locally, because of its complex user interface. Commonly used for organization level events.

The new system had a preexisting event management which was incompatible with the old and the old data had to be dropped. Users of this module were contacted and had to find alternative methods for managing their event registrations if their use period overlapped the switch created between the old and the new system.

3.6.3.2 Access rights

A module for handling administrator access rights. Commonly used by the organization administrators to give rights to group administrators. This system was replaced to allow for more automatic grants of access rights in the new system and was thus not transferred.

3.6.3.3 Government grant reports

This part allowed for assigning members to subgroups and to report these as independent organizations. The data was used in SSF’s government grant applications. While this part was system-critical, the next reports did not need to be created until 6 months after the end of the first development cycle, it was thus not necessary to include it in the first cycle. The data in the old system, where existing, was however used to group the members into subgroups in the new system.

The functionality of government grant reports is of vital importance for the economy of SSF. The next round of reports was not due until December, and it was thus decided that it could be developed in a later cycle and thus not be included in the first version of the system.

3.6.3.4 List creation

In the old system this part allowed for the creation of complex lists of members, such as printable address notes or printable member rosters. The new system contained basic such functionality and while unsatisfactory, it was deemed to be adequate for the first development cycle.

3.6.3.5 Email lists

Functionality to create emailing lists based on groups or criteria in the system. While a necessity in the long run, it was traded off as not being system-critical.

3.6.3.6 File system

The old system had a file storage system where administrators could upload and download files. It was only used by a small number of groups and those groups were informed to handle their files in other fashions until it could be developed in a later development cycle.

3.6.3.7 Polls

The old system allowed for the creation of browser based polls to make evaluations and questionnaires. It was not widely used and was dropped in its entirety.

4 Planned activities

4.1 Engineering activities

4.1.1 Studies

In order to be able to assist the suppliers' contractor in mapping the old database the T-SQL database script language has to be learned. This is to be done by studying tutorials on the subject and documentation of the T-SQL language. Furthermore a general understanding of what parts and activities are involved in a data migration project is necessary. This is to be done by studying technical literature based on experience from data migration projects.

4.1.2 Mapping database

The database of the old system was exported in a raw format, containing only tables, table definitions and column names. A database will normally include different kinds of constraints. The constraints detail the relations of columns in the tables. No such constraints were included in the export. To use the export effectively the basic relation constraints had to be recreated.

The database will have to be mapped and understood in order to explain its intricacies to the supplier's database migration contractor. The most complex operation in the migration will be the merging of duplicate members, logics for that operation will have to be created together with the supplier.

4.1.3 Database migration

The old database contains data which either has no place in the new database, or are difficult and costly to fit into the new system. Deals have to be made with the involved data stakeholders on what is to be done with every part of data in the old system. This will be done by mapping and evaluating the value of data in the database versus the costs of migration versus alternative options.

Work will be conducted together with the supplier's contractor to support him in decisions, actions and coding solutions pertaining to the old database.

4.2 Advisory activities

To act as a mediator between the customer and the supplier by being available for explaining difficult concepts, recommend and give advice in what paths to take and in defining technical concepts. The actual activities vary depending on the needs of the customer and the supplier.

4.3 Evaluation of scrum

There are some research on the efficiency and deployment of the scrum development method in software developing companies. In this report we will look at the interactions between the Scrum process and the customer buying from a Scrum oriented developer.

The scrum method places certain requirements of the participation of the customer. The thesis attempts to identify the requirements and to measure the resources needed from the customer to meet those requirements.

Not all of the methods of the Scrum development process are visible to the customer, in this report only the interaction between the customer and the supplier working with the Scrum method are explored. It will look at what requirements are put on the customer, what advantages and disadvantages that are experienced with the working method and will evaluate whether the three pillars of Scrum; transparency, inspection and adaptation are present in the working process from a customer perspective.

5 Studies

5.1 Scrum

Scrum was studied for the primary purpose of understanding how the developer worked internally and how Scrum interacts with the customer. '**SCRUM AND XP FROM THE TRENCHES**' [5] is a very hands-on book with vivid examples and was used as main reference along with the defining paper '**SCRUM GUIDE**' [1].

5.2 Data migration

As responsible for the data migration a good source with examples and guidance from beginning to start was needed. Morris book '**PRACTICAL DATA MIGRATION**' [2] was studied at the start of the project and was used as reference throughout the project.

5.3 T-SQL

Fianga.com [5] was used as a tutorial for the T-SQL (Transact Standard Query Language) language which is used to interact with a Microsoft SQL Server. Except for a step-by-step introduction to most of the functionality of T-SQL it also contained a glossary of the keywords with explanations being combined with examples.

6 Data migration

Using an iterative development model, introducing changes as problems were discovered in transform or after tests after loading

6.1 Old database

6.1.1 Scheduling the export

6.1.1.1 Problem

SSF did not have direct access to the database of the old system. As the preparation of a database migration is a lengthy process the old database needed to be exported on several occasions.

6.1.1.2 Solution

At the beginning of the project an agreement was reached with the old supplier on when the database was to be exported. One immediate export for mapping purposes, a second export to test that invoice generation was equal in new and old system and a third export for the actual migration.

The agreement included what format the database was to be supplied in, on which dates the exports would be made and that the provided export was in the format of a backup of a Microsoft SQL Server 2008.

6.1.2 Handling sensitive information

6.1.2.1 Problem

The database contained sensitive personal information and according to the Swedish Privacy Protection Law [12] had to be treated with care regarding safe-guarding the data from ending up with third parties.

6.1.2.2 Solution

The old supplier required a signed document from the Secretary-General as a proof of a person being allowed to acquire copies of the export. This safeguarded that they did not deliver the database to the wrong person.

The new supplier signed a contract on the limits of their use of the database. The contract detailed that the supplier was not allowed to search the database or transfer any parts of it to third parties and that it was to be deleted once no longer required to fulfill their obligations towards SSF.

The transfer of the database from one site to another, whether physically or via internet transfer was always done in encrypted form. The password for the encryption was transferred via separate means, for example by SMS or phone call.

6.2 Mapping the database

6.2.1 Establishing an interface to the database export

6.2.1.1 Problem

The database was supplied in the format of a backup file of a Microsoft SQL Server 2008 database. Programs to utilize the data and achieve an interface to interact with the data had to be set up.

6.2.1.2 Solution

To load the backup file the free version Microsoft SQL Server 2008 Express was installed on a computer with Microsoft Windows XP Professional.
The standard scripting language of Microsoft SQL Server 2008 was PowerShell [12]. Instead of learning a new language a Python module, PyODBC [7] was used. ODBC [15] provides standard software interface drivers to hundreds of database systems and PyODBC utilizes this connection allowing Python scripts to connect to for example Microsoft SQL Server 2008.
A script was then created in Python utilizing the columns() function of PyODBC to retrieve a complete mapping of all the tables and their columns which was exported to a CSV text file.
The script also mapped for each column how many unique values each column contained and

whether all the entries of a column were unique. This extra data was used to identify unused columns (chap. 6.2.2) and primary key columns (chap. 6.2.3). This database map was utilized by all future scripts running over the database. The original database contained 566 columns in 39 tables.

6.2.2 Unused tables and columns

6.2.2.1 Problem

In the database there were a lot of columns which did not contain any data or just one and the same data in all fields. The previous system had developed organically, functionality being added in small chunks and the format of the database reflected this. Functionality had also been removed, with the accompanying data persisting in the database, leaving columns that were no longer used. Some whole tables had been discontinued, but not removed. This could also be noted by some older data not having fields which were in newer data, or the opposite, older data had fields which newer ones did not.

6.2.2.2 Solution

A uniqueness check was introduced in the database mapping script. NULL values were counted as a value of its own, as it may also bear a meaning. If a column contained only one and the same value in all of its fields, the data did not affect the system and could be ignored in the migration. This check allowed more than 20% of the database columns to be ignored for not containing any data.

All columns with very few values in comparison to the size of the table were also checked. In about 10 cases this data could also be ignored. The reason for the different values could for example be fields containing NULL and an empty string.

On the request of the supplier’s database migration manager these columns were included in the export for completeness but were marked in the documentation as ‘empty’.

6.2.3 Recreating database relations

6.2.3.1 Problem

The primary key / foreign key relation is the foundation of a database; it tells what pieces of data belong together. The enforcement of the relations can be implemented in the database layer or in the application layer. In the old system the enforcement was implemented in the application layer. As SSF did not have access to the code of the application; the relations of the database received in an export were unknown.

6.2.3.2 Solution

The primary key could usually be identified as being the first column of the table, and in other cases as being prefixed by “pk_” or having the table name appended with “Id”. A foreign key could usually be identified by having an equal column name to its parent primary key, such as Adr.OrgId being a foreign key to Org.OrgId primary key. The automatic mapping of the database had supplied information about whether all entries in a column were unique, a necessary trait for any primary key candidate which thus could be used as an indication on what columns that could not be primary keys.

As relationships were found they were manually inserted into the map of the database. An automated script was created which tested all primary / foreign key relations that had been documented in the map. This was based around the SQL-code (see Code example 1), it finds all values in a proposed foreign key column and checks which of those values do not exist in the proposed primary key column. A correct relation proposition would normally generate 0% hits. Because of corrupt data (See chapter 6.2.4) many correct propositions would however generate some hits, up to 10% in these cases. An incorrect proposition would however generate something in the lines of 50-100% hits.

As the database will not allow insertion of foreign key restrictions on data where there are any errors, corrupt data were sometimes manually fixed by deletion of entries not having a primary key to relate to. Even after these manual fixes and the fix described in chapter 6.2.4, other errors still caused that 17 out of 84 foreign keys could not be introduced because of corrupt data. This was however not a major issue as the relations could still be used, whether the restrictions were in place or not, and the remaining relation problems were decided as having minor or no effect on the migration.

```
SELECT fk_table.fk_column
FROM fk_table
LEFT OUTER JOIN pk_table
ON fk_table. fk_column = pk_table. pk_column collate
WHERE pk_table.pk_column IS NULL AND fk_table. fk_column IS NOT NULL
```

Code example 1 – T-SQL code for finding any instances in a proposed foreign key (fk) column which do not have a corresponding primary key (pk) in a proposed primary key column

6.2.4.1 Problem

The mapping of relations was severely complicated by the export unbeknownst to SSF being partially corrupted. A lot of time searching for errors in migration code was spent before the cause of the problem was found: Different tables in the export had been exported at different dates, some tables having been exported up to 10 days prior to other related tables. Since the system was live during those days, new entries had been created which had relations to tables that had already been exported or vice versa.

6.2.4.2 Solution

In all the future exports the old system supplier was ordered to shut down the system while the export was being done. While this caused concerns with the users during the 5 hours the system was shut down, it was vital for having a non-corrupt export. The export would still contain some relation errors, but they numbered in the tens instead of in the thousands. A code piece introduced to combat the initial corruption later made it into the final migration, leading to data loss, see chapter 6.6.3.3.

In the future exports the integrity of the exported data was tested with the relation-test script described in chapter 6.2.3.2.

6.2.5 Missing data in export

6.2.5.1 Problem

The mapping of relations was further complicated by the export having some tables and partial data in tables not being in the export. This led to the fact that some relations could not be found, or some relations generating extensive amounts of errors.

6.2.5.2 Solution

The problems were relayed to the old system supplier, which resolved them by sending a partial data table which could then be imported in the existing database.

6.2.6 Creating a diagram of the database

6.2.6.1 Problem

To gain an overview of the relation of the database tables a diagram over the relations was needed. Microsoft SQL Server 2008 has functionality for automatically creating a graphical diagram over a database and the relations between the database tables. Because of the corrupt data in the database (see chap. 6.2.4) it was however not possible to insert all relations in an existing set of the database.

6.2.6.2 Solution

A new empty database with all tables, columns and relations included was created through the same scripts that were created for the migration (see chap. 6.3.1.1). This empty database was then used to create a graphical diagram, see Figure 5. Each block in the diagram represents a table. Each arrow represents a foreign key / primary key relationship between those tables. The complexity of the diagram was reduced by removing 27 tables that were not included in the first migration cycle and then manual rearrangement to reduce the amount of crossover lines. A special table containing explanations of types in all other tables was also dropped; this table had at least one relation to each other table and the inclusion did thus not add any knowledge value to the diagram.

Once the diagram was complete three central parts stood out:

1. Organization units on all levels, from main organization down to the patrol.
2. Membership data and data related to the membership.
3. Invoicing of membership fees, this part has no obvious central table.

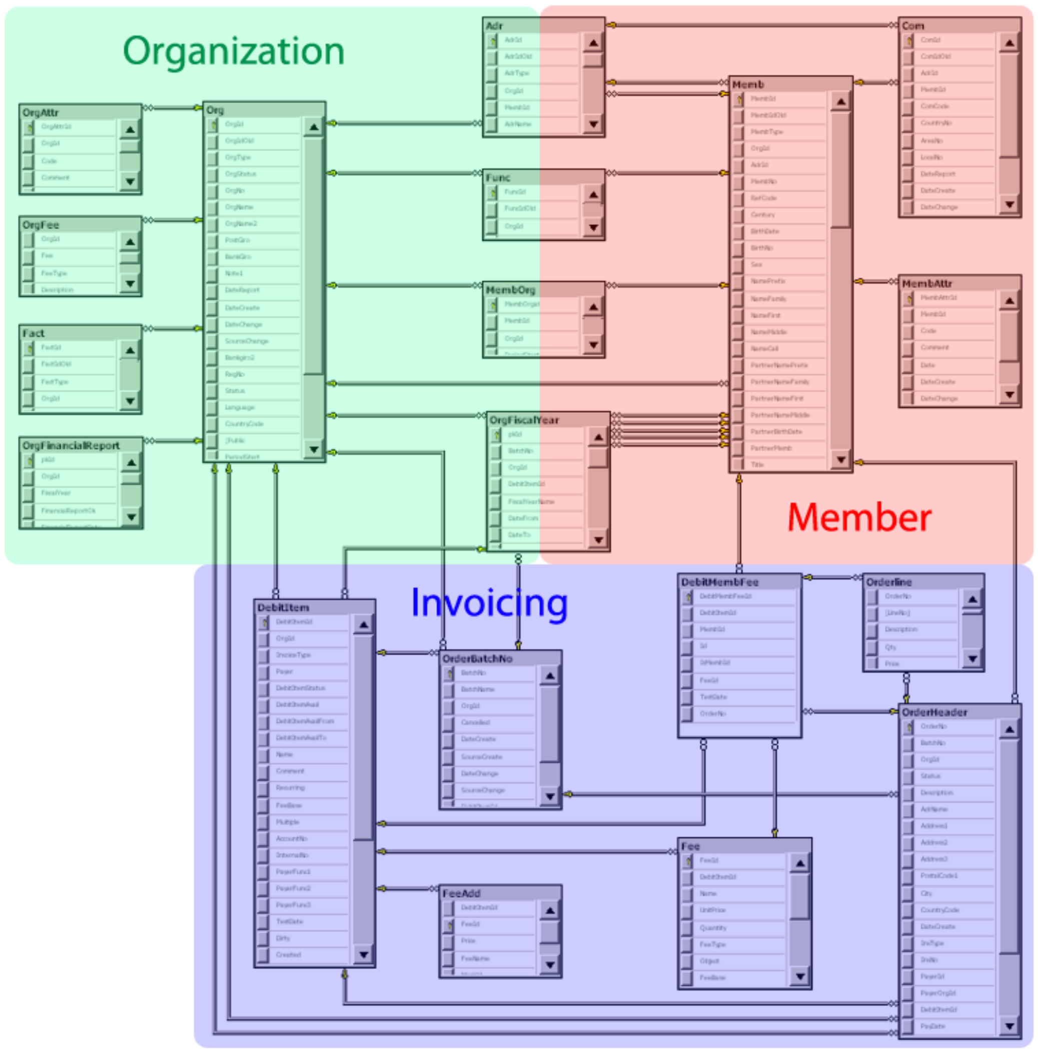


Figure 5 – Database diagram over migrated parts of old system

6.2.7 Fixing minority problems

6.2.7.1 Problem

In many areas of the database there were minor amounts of entries causing logical problems on import or after their entry into the new system. These entries were often exceptional cases that for different reasons did not follow the standard logics.

Such problems could be:

Data that was inserted in the database for testing.
Members with double primary memberships (see chap 6.6)
Data that had fallen through the logic of the old database and did not have appropriate relational data
Manual economical transactions
Data circumventing the logic to attain a goal which was possible in the old system even if the function was not intended for that use:

- Organizations that were except from invoicing.

- Entering the same SSNO for different persons to circumvent SSNO validity check.

6.2.7.2 Solution

Most such data had to be deleted through manual fixes or in one case an automated script.

6.2.8 Keeping track of manual database fixes

6.2.8.1 Problem

During the research of the database structure both the customer and the supplier continuously discovered minor database errors that required manual correction by insertions, alterations or deletion of data. The database was to be migrated on three separate occasions. If the manual changes were not exactly equal in the future migrations any problem fixed in a previous version would resurface in the next.

6.2.8.2 Solution

Any data altering script applied on the working database was manually recorded in a T-SQL script file. In future migrations this script was run on the newly exported database to reintroduce all changes made to the previous version.

6.2.9 Minimizing downtime between systems

6.2.9.1 Problem

During a migration the old system has to be shut down from alterations to the data to be exported. This makes it impossible for the users to work with system. For this reason it is normally desirable to make the migration in as short a time as possible.

6.2.9.2 Solution

For unknown reasons the export by the old supplier took about 5-7 hours to complete.

On the customer side the time was minimized by creating a strict and tested procedure for those parts for which it was responsible. The procedure employed about 1 hour of time. The new supplier imported the data in a few hours, making the total necessary downtime less than 12 hours.

At the time of the final migration and shutdown of the old system, the new system was not ready for deployment. Thus the minimizing had little real impact; the downtime became 4 weeks of users having only read access in the old system after final migration and an additional two weeks where no system was available at all.

6.2.10 Structure of export changing

6.2.10.1 Problem

The database export was done on three separate occasions. As the old system was undergoing development during this time, the database structure could be altered from export to export.

6.2.10.2 Solution

A script was created that checked whether all tables and columns in an export existed; it also found any new tables or columns. The result from this process was used to update the database map. The necessary changes were minor, but through the script they were discovered at an early stage instead of causing bugs further along the migration process. This had the double advantage of checking whether the export was complete; it was believed that the old supplier did the export manually, and thus could not be relied on to be entirely accurate.

6.2.11 Minor problems

Some tables were used for different kinds of data, invoices were mixed with member reports data for example. Items in such tables could have inconsistent logics which had to be treated differently.

The system had reused some columns for storing unrelated data, making the name of the column being completely unrelated to what it was really storing. For example an insurance related column was used for storing the status of whether the member should receive a magazine. Finding the logics behind such names required research into the existing system comparing and finding the actual differences between entries.

6.3 ‘Free software’ problem

Three weeks after receiving the database backup file the supplier reported that they could not use the database in its native format; this was caused by the supplier being a ‘Free Software’ company.

GNU defines ‘Free Software’ [15] as living up to the following criteria:

"Free software is a matter of the users' freedom to run, copy, distribute, study, change and improve the software. More precisely, it means that the program's users have the four essential freedoms:

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and change it to make it do what you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this."

The database from the system was in Microsoft SQL Server 2008 (MS SQL Server) format, a format which can only be read by the Microsoft range of products which are not living up to the ‘Free Software’ standard. The supplier, being a "Free Software" company, could not use Microsoft products.

The supplier had developed the new system with PostGre SQL (PG-SQL). PG-SQL is open-source database software adhering to the ‘Free Software’ policy.

6.3.1 Database transformation

The database had to be transformed into a format which the supplier could use. No tools enabling a direct transformation of data from the format of MS SQL Server to the format of PG SQL could be found.

6.3.1.1 Transforming database format

The format of the database can be described in SQL code which can be inserted to recreate the database format in another system. For example Microsoft SQL Server Database Publishing Wizard 1.1 accomplishes this [16], but did not work with a 2008 version. Instead a Python script was created which took the database map and generated T-SQL code from it. As the only formatting required were the table definitions, primary keys and foreign keys, the Python script was only about 115 lines of code.

6.3.1.2 Transforming database values

The new supplier suggested that the data in database was exported to CSV (comma separated values) text files. CSV-files can have slightly different formats [18]; to avoid complications an agreement was made on which format was going to be used.

The export to CSV-files was made with Python scripts using Pythons in-built csv-module [18]. The map of the tables was used to query the database for every column. The column data could then be written to CSV files. The default way for Python to print values did not always retain the same format of the data, thus some types of data values had to be transformed into strings using certain rules. For example strings in the database would have their end-of-line characters escaped and date variables had their precision cut down from Pythons 6 decimals to the 3 decimals used in the database.

For an unknown reason there were errors in exporting the Swedish letters åäö in one of the tables. As the table was small, about 20 rows, and unchanging, it was exported manually instead of trying to solve the highly localized problem.

The more advanced versions of Microsoft SQL server do have the capability to export directly to CSV-text files [19]; however the Express version did not. SSF did at this time only have the free version of Microsoft SQL server, called Express. SSF later acquired an edition of the server with this capability, the transformation procedure was not changed then in the scheme of ‘if it ain’t broke, don’t fix it’.

6.4 Quality versus cost

SSF were sensitive to quality problems to not cause havoc with its lively user-community, who were already tired of their tasks after years of problems with the earlier system.

6.4.1 Handling mandatory data

In the new system it had been decided that some member data would be mandatory, that data did not always exist in the old system. Project management made a suggestion to let group administrators find and enter all such missing data. Since the missing amount of data was in the thousands, it was decided that this task would be insurmountable in the time given.
This decision created a problem in the design of the new system. In the new system it was impossible to save an entry which did not have all mandatory fields filled in. This would lead to all old incomplete data being impossible to edit without filling out all the previously incomplete data.
To still enforce the mandatory data a solution was created in the new system:

1. When creating new entries all the mandatory data was enforced to be entered
2. When editing data, a solution was devised where it allowed saving entries which were previously empty to remain empty, but mandatory data that had once existed could only be altered, not deleted.

This solution allowed the old data to be incomplete, but without risking that incomplete data was ever entered again.

6.5 *Reexports*

6.5.1 Invoice data

Both systems handled invoices, to keep track of their payment status files of bank transactions were imported into the system. This was a similar process in both the old and the new system.
The old database thus did contain payment statuses of the invoices. It was decided that these would be migrated by a separate means. Instead of transferring the data from the database, the data was inserted in the new database by importing the same files that had already been imported in the old database. This scheme was chosen by the new supplier because it created a test of the invoice capabilities of the new system.
Manual changes in the old system, corrections and cancellations of invoices for example, were imported via a separate insertion.

6.6 *Merging duplicated members*

In the old system a membership was defined as the data of one person. Thus if a person was a member in two organizations, there existed two unrelated instances of that persons data, one for each membership. Since the instances were unrelated they could have different data for the same person. For example:

1. Slightly different names because of typos
2. Contact information and address being different because one of the memberships not being up to date or being defined differently
3. Birth date or last four digits of SSNO being defined in one instance but not the other

In the new system a person was defined by one instance of person data, with several memberships assigned to that instance. It had been decided that a goal of the migration would be to remove the double instances of person data. The data in double memberships would thus have to be merged as in Figure 6. Some persons had more than two memberships; these instances were merged with the same method.

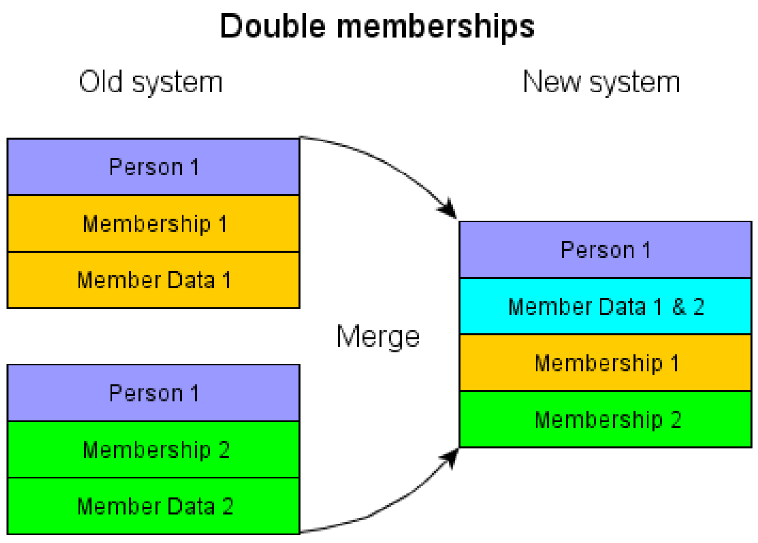


Figure 6 – Merging model of duplicate memberships in old database

6.6.1 Merging method

The actual merging of data was done by the developer on import into the new system. The logics to use for merging were however complex (see Figure 7) and had to be developed in iterations, testing imports live in the new system to discover errors. Early in the development misunderstandings were common, to combat this SQL code was used as a means of communicating the decisions with a high level of precision.

6.6.1.1 Merge logic – Member number

The old system had the ability to make copies of members. This feature was rarely used, but in those cases the two instances would retain their membership number and would obviously be of duplicate memberships.

Rule: Member number were equal

6.6.1.2 Merge logic – SSNO

The old system had incorporated the entry of a complete SSNO some years earlier. The SSNO is unique for every person. It was thus assumed that

- SSNO are equal

Some of the times users had entered the same SSNO several times, this resulted in mergers of members obviously not being the same person. The following had to be added to exclude those cases. The reason for allowing any combination was that users had often entered the first and family names in an opposite order.

- any of first / family names being equal to one of the other first / family names

Memberships with equal SSNOs often had different, but very close name matches. This was most often caused by user spelling mistakes. Those mistakes were fixed manually. In the cases where SSNO were equal on differing persons, the SSNO had to be deleted, as it was invalid data[1].

Final rule: SSNO are equal

Exceptions handled manually: SSNO are equal but names are not

6.6.1.3 Merge logic – Birth date

The old system had not enforced the entry of a complete SSNO from the beginning, and thus only the birth date was defined in about half of the entries. In a larger set of data this method would create errors on common names, but in this set it was deemed from manual checking to be sufficiently accurate.

- birth dates are equal

and

- first and family names are equal

1 In the new system the entry of SSNO in a new member would be mandatory by technical enforcing. This rule was enforced in the old system as well. In the new system two members would also not be allowed to share the same SSNO. This follows from the fact that the SSNO is unique to every person and every person should only be entered in the system once. Thus at migration time the last four digits of all non-unique SSNOs in the old system had to be deleted to not immediately compromise the rules of the new system.

In this case there were memberships where one instance would have a complete SSNO and the other only birth date defined. The complete SSNO was then manually inserted into the other instance, effectively moving that instance into the SSNO rule.

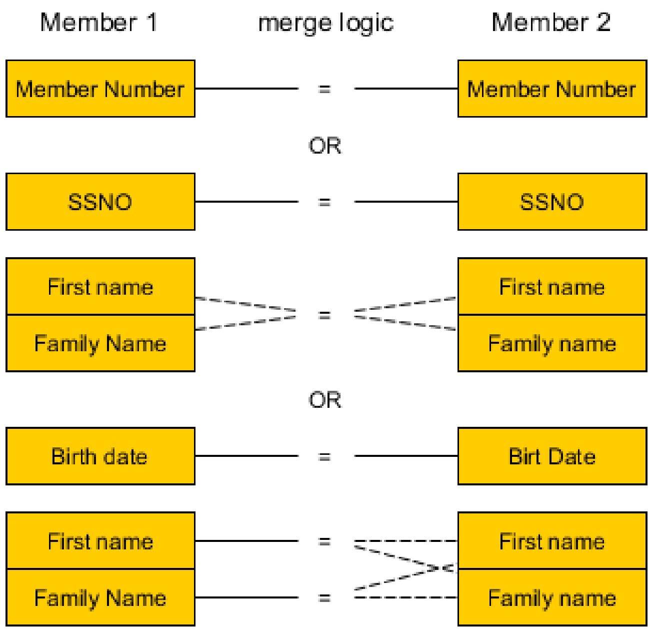


Figure 7 – Final logic for merging members, dotted lines meaning only one has to be true

A lot of the time users had entered the first and family name in an opposite order, thus the following option was added.

- first name equals family name and family name equals first name

Final rule:

birth dates are equal

AND

(first and family names are equal

OR

first name equals family name and family name equals first name)

6.6.2 End quality

From the golden rules of data migration (see chapter 1.3.2.1) we had that: “No organization needs, wants or will pay for perfect quality data”.

Since it is not possible to reach a perfect level of quality of data in a migration, some rules had to be decided. These quality rules were used in the project:

- There should be as little information loss as possible
- Further fine-tuning would have been more costly than fixing the errors manually as they were discovered
- As many members should be merged as possible, but without breaking rule 1 and 2.

Having a low grade of erroneous mergers was deemed to be the most important because errors:

1. would make the system lose information which would have to be recreated by local administrators
2. would result in losing track of members, the local administrators might not be able to contact those members
3. would be less likely to be discovered by the members or the local administrators

The end result was deemed to be adequate when it had reached a stage where testers no longer reported issues which were not already known to be unsolvable. At this stage it was estimated that a high grade of merging and a very low, supposedly zero grade of erroneous mergers had been achieved.

Memberships that should have been merged but weren't would be discovered when members received double invoices. The organization administrators would then be able to manually merge those persons in the system by deleting one instance of the person.

Known issues which were not resolved were cases where one of the memberships:

1. did not have a birth date defined or had an erroneous birth date defined
2. had correct birth dates defined, did not have SSNO defined, and had typos in their names

A solution to improve the number of correct merges that was not explored would have been logics which used other factors, such as equality of email, addresses or telephone numbers in a combination with names.

An example of a logic that could not be used was to merge instances with the same first and last names. This logic could have worked in a very small set of data, but would have resulted in a large number of erroneous merges in this larger set of data. Another such logic was birth date combined with family name. This could have avoided some spelling mistakes on the first name, it was however not usable because of the large number of twins in the system.

6.6.3 Identified problems and errors after migration

6.6.3.1 Non-members receiving invoices

In the old system 'non-members' were people that were tracked in the system, for example board members, that were not actual members of the organization.
In the new system non-members were initially being reported as members because the new system had no way of registering non-members. A new member state had to be introduced to make it so that non-members did not receive an invoice.
An initial fix to this problem was a simple marker in the new system that noted whether a membership was 'not billable', and such a membership would thus not receive any invoices. Sometime before the first invoice round an unannounced fix removed this status and replaced it with a new functionality which did not accurately catch all the previous non-members. A partial fix was done to the members of a single group, but this left members in other groups in a still billable status. In the first invoice round this led to about 160-200 erroneous invoices being sent. See Figure 8 for an overview of the changes in chronological order.

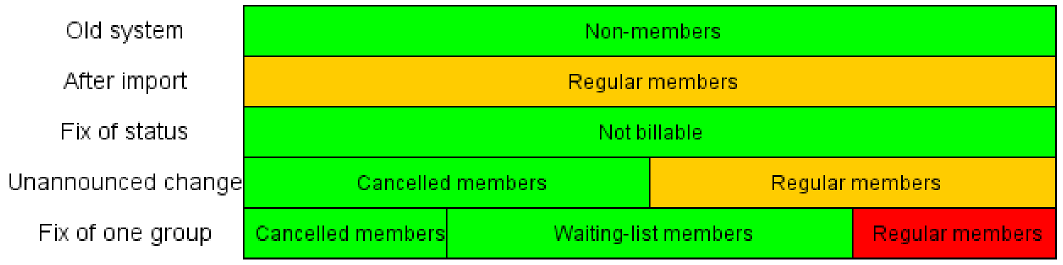


Figure 8 - Status changes of non-members from old system, unannounced changes led to erroneous invoices being sent to non-members (size of chart pieces does not represent actual proportions)

6.6.3.2 Missing memberships for some non-members

Some combinations of non-memberships were missed in the import scripts. This led to about 250 instances where non-memberships were not assigned to persons in the new system. This was resolved by manual insertions.

6.6.3.3 Deleted group

In the export transformation scripts a previous bug-fix led to a group of 30 members being deleted. As this was known before the closing of the old system a complete report of this group was exported in the old system which allowed all members to later be imported by hand into the new system. Importing data by hand from the database data would have been more difficult seeing to how data there is spread out over many tables.

6.6.3.4 Duplicated data

The merger of members presented problems when importing some of the data. A choice was made where all the members' memberships would be preserved, at the cost of importing contact data such as addresses and phone numbers once for each membership, thus sometimes ending up with double or triple data when the data were not exactly equal. This decision was made in line with the quality rule: "There should be as little information loss as possible" (see chapter 6.6.2).

6.6.4 Historical records not existing

The old system was not capable of accurately storing changes made to the data; it normally overwrote the previous data when making any change. The new system used a versioning table, such that whenever, a change is introduced, a copy is made and stored of the previous data. In this way the new system was able to go back to any point in time and recreate the systems status at that point.
Since this data was not complete in the old database, no such versioning could be inserted from the old system and this led to complications when the new system was attempting to recreate historical conditions prior to the import.
Specifically SSF needed for its government grant applications to recreate which members had been a member some time during the current year. The logic of finding this data in the previous system was complicated and could not be made completely accurate. Eventually a solution was made where the group administrators could select from a list of all their potential member candidates which members to actually report for the government grants, thus moving the responsibility for the accuracy of the report away from the system and on to the users.

6.7 Shadow invoicing

It was planned to make one invoicing round which created invoices in both the new and the old system simultaneously. The idea was that the output of the new system could be compared to the output of the old system. This comparison would give a chance to possibly finding errors in the database import and logical errors in the invoice generation procedure in the new system.

The comparison was originally planned to be done by 15 May, in the end it was done by 10 July. At this time the old system produced 115 invoices while the new system first produced about 1800 invoices which were quickly cut down to 188. The reason for the number of invoices being low, was that most members had already received their invoices, the next invoicing round would be generating about 35 000 invoices.

The remaining difference matching was done by scripts and then manual research. This matching uncovered a bug in the generator which created the invoice printer data, resulting in erroneous sums. Some of the errors were ignored for requiring a lot of manual mangling and pertaining to a one-time error concerning very small sums and the loser being the organization. Other errors could be ignored for being actual system differences, the previous system had a rebate on non-magazine subscribers, and the new system did not.

This process mangled out two serious bugs and ensured that the systems were coherent. Later changes in the new system made that erroneous invoices were still sent in the first invoice round sent from the new system.

7 Advisor activities

7.1 Improving the user-interface

A screen capture program for producing video help tutorials was tested as a possible mean to create a video based manual for the system users. As it was used to create a video tutorial of the scenario ‘create new member’, the process proved to lead to a lot of mistakes because of difficulties in finding the correct buttons. A comparative video tutorial was created in the user-interface of the old system and seemed to be much simpler. Both tutorials were minimized as to simulate a ‘perfect user’ making no mistakes. The result was that the old system won with 3 to 13 button presses / page loads and 40 to 80 seconds compared to the new system.
Being one of the most common processes made by the users it proved a considerable setback compared to the old system. A layout was created and suggested to the supplier.

An agreement was reached to replace some unneeded stories with the story of a ‘one-page member creation’ that reduced the complexity to equal that of the old system.

7.2 Changing of access rights

In the previous system access rights were granted on a user basis and was handed out by the organization administrator. The new system had a goal of making access rights being assigned automatically and logically based on a user’s roles within the organization.

The problem was that there were 55 different roles, in four different levels of the organization, and SSF wanted each of them to have their own set of access right rules, and each rule was also set to read, create or edit access. For example a member assigned the role of course leader would have the right to see limited information on members and have the ability to create and edit activities. A member could further have multiple roles assigned, thus granting rights for each role.

SSF had defined the wishes on access rights and roles in a lengthy text document and as an advisory activity the documents were transformed into a structured comma separated text file with clear and well defined rules. Based on these rules the developer choose to even implement an in system user-interface to allow the administrator access to changing the access rights via the user interface.

The developer was not able to limit access rights to one level in the organization, all rights propagated downwards in the organization. A user with some rights at a higher level in the organization hierarchy would thus have the same rights in all lower organization units. While SSF wished for limiting the access to organizational units, the cost of rebuilding the system to allow this would have been too great to merit the benefit of extra control over rather few members.

Since the access systems were so different in layout, the previous access rights from the old system were not imported into the new system. The new access rights were instead based on what roles were assigned to the members in the old system, something which had previously not automatically given any access rights.

7.3 Other advisory activities

- Standardizing member procedures

  Handling person information is regulated by the Swedish Personal Data Act [10]. This states among other things that a person has to agree to their data being inserted into and managed by a computerized system. To help local administrators and group leaders in doing this correctly, a standardized printable form was created which included the correct legal information.

- Family invoices

  In the original plan there was an idea for sending member invoices to only a single person in a family in the cases where several persons within a family were members. This was to make it easier for families and reduce on the costs of sending multiple invoices.

  A reason for SSF starting to send central invoices was to gain a better control of how many members were in the organization, as required by government institutions granting funding based on member numbers. It was questionable whether invoices being sent to another person would constitute an act of declaring a membership in the legal sense and thus this idea was advised against. The idea was replaced through a change request with another story needed better.

- SSNO procedure

  SSF required that only valid SSNOs were entered in the new system. A programmatic specification (see Code example 2) of the Swedish SSNO format and checksum system was written in a generic language such that the developer could implement this in the system.

```
SSNO = ‘yymmdd-nnnn’ # replace string with SSNO to be checked

multiplier = 1
checksum = 0

for i in [0,1,2,3,4,5 , 7,8,9]
   if multiplier == 1:
      multiplier = 2
   else:
      multiplier = 1

   num = int(SSNO[i]) * multiplier
   if num > 9:
      checksum += 1
   checksum += num % 10

if int(SSNO[10]) = (10 – checksum % 10):
   print ‘correct SSNO’

#PHP for example has another definition of the modulo operator % [23]
#last check is then: (10 – (checksum % 10))  % 10
```

Code example 2 – A programmatic correctness check of a Swedish SSNO

8 Project organization and management

The supplier defined their wishes on how the project be run for them to be able to work efficiently.

8.1 Overall organization

The supplier’s organization:

- Local project manager, locally based by the customer and also part of steering committee
- Scrum master, locally based with the development team
- Development team, consisting of a varying amount of members
- One member for Steering committee, from the suppliers management

The supplier’s demands on the customer’s organization:

- Project manager
- Project owner
- Two members for steering committee, preferably with a secretary general or similar
- Project team and resources for testing and providing feedback on sprint results

The supplier’s demands on the customer’s activities:

- Every week: Steering group meeting – Participation by all Steering committee members
- Every 2 weeks: Sprint planning – Participation by customers project team
- Every 2 weeks: Sprint testing – Provide feedback and acceptance testing

8.2 Steering group

The steering group consisted of one project manager and one project owner from both the customer and the developer. The steering group met for up to one hour once a week and the meetings were recorded on paper.

The steering group meetings had a number of functions:

- Making a review of the accomplishment, costs and projected costs
- Meetings needed in the future were planned
- Change requests were relayed through the steering group
- Strategic decisions such as changing of deadlines and decisions on problem areas

8.3 Telephone conferences

When details of a story or a problem were complex a telephone conference was arranged with the affected parties from the customer and supplier. This would clarify misunderstandings in implementations and agreements could quickly be reached on how certain problems should be handled or functions implemented.

8.4 Stories

The suppliers internally used a program called ScrumWorks [21]. In ScrumWorks the project is divided into stories. Each story had a description for a function in the system and an estimate of the length it would take to develop expressed in 'points'. The project was initially divided into such stories by the supplier. This was done from the requirement specification provided by SSF.

Because SSF did not have access to ScrumWorks, it was more difficult to keep track of the items in it and the progress on the stories. SSF only had a list with the names of each story, sometimes with such obscure names that it could not be deduced what functionality those stories related to. The list was all that SSF had available to make priorities for sprints.

8.5 Sprint meetings

Sprint meetings were performed every second week, usually performed by video conference in the morning and usually taking 3 – 4 hours. The supplier started with a presentation of the stories which had been completed in the previous sprint. The presentations were usually demonstrated directly in the system on screen. When bugs occurred they were usually fixed during the presentation. In the cases that they could not be immediately fixed, they were logged in the bug tracker system.

The second part of the meetings was used for planning the next sprint. The supplier presented a suggestion for stories which were to be implemented in the next sprint. Discussions between customer and suppliers over details in the stories could further clarify how the stories were to be implemented. The customer also had the chance to reprioritize stories, exchanging items in the proposed list with items from the backlog.

In the early stages of the project most of the programmers attended the sprint meetings, with many not being an active part of the meeting. In the later meetings it was decided that it was better to have only the programmers necessary for the presentation present to not use up those personnel resources unnecessarily.

After the sprint meeting the customer and supplier would spend the afternoon on their own documenting what had been decided and making final decisions on priorities and stories to be implemented in the list. As more tickets started to accumulate in the bug tracker system, the bug tickets would also be prioritized together with the stories.

8.6 Ticket system

The supplier employed a project system called Redmine [22] for the project. Its main purpose in the project was to act as a ticket tracker system where both the customer and the supplier could create tickets of problems, bugs, features, change requests and questions. The tickets could be assigned priorities, responsible person, status and progress.
Each entry or change among the tickets would generate an email to a list of persons generally ‘watching’ the system and the ones directly involved with the ticket. It was the supplier’s project leader who had the responsibility to assign new items to specific developers.
The customer was given a large responsibility in updating the status and priority of the items. As the project progressed and it was realized that it was difficult to prioritize between the tickets and the stories in Scrumworks, some stories were moved into the ticket tracker system as well.

8.6.1 Ticket status

The supplier would put an issue in status 'waiting for acceptance'. This item could then not be immediately checked for acceptance because it was waiting for a new deployment or a dependency. Thus tickets had to be checked in intervals, by the user trying to find tickets that could be tested.

The status of the item had limitations to what status a user could change the status of a ticket. The rules were not clear and often led to items becoming stuck in the wrong type of status. A ticket with status ‘waiting for acceptance’ could in some cases for example not be changed back to any ‘needs further development’ or to ‘closed’ status. There was some background logic in the system behind who could change the statuses, but how it worked was never understood by the customer and the problem was instead worked around by sending messages to the suppliers prompting them to change the ticket status.

8.6.2 Ticket questions

8.6.2.1 Problem

The system allowed directing an entry inside a ticket as a question to another user. The user would be specifically mailed the question and the user would have a specific link to list all its questions.

If the question is answered by another user than to who the question was aimed for, the question would persist in the system, and the questioneer would not directly receive the answer. The question would have to be manually removed by the original target by the insertion of a new nonsense entry, which is then unnecessarily relayed to all watchers.

8.6.2.2 Suggested improvement

If a question is asked generally and to anyone, there will be no specific user responsible for the answer and the chance of not being answered becomes higher. The idea of directing an entry as a question to a specific person is thus good. There is however often more than one user able to answer a question and this was not supported. Technically this should be possible to solve by assigning an ‘answer’-button to entries with questions in them.

8.6.3 Ticket status

8.6.3.1 Problem

A ticket could accumulate a lengthy discussion, changing the goal and purpose as the discussion went along. The only way to know the status of a ticket was to read most of the generated content, making it a lengthy process to go through the existing items.

8.6.3.2 Suggested improvement

Allowing the attachment of a note to an issue, with a short status description could improve the handling. The note could be displayed directly in the listing of issues, thus a user would not have to enter the individual item and possibly read through all entries inside it to understand the current status and requirements of the item.

8.6.4 Minor updates

8.6.4.1 Problem

There was no way of making updates without sending an email to all listeners. Thus making smaller corrections, mass changes or bookkeeping generates unnecessary emails to a lot of recipients.

8.6.4.2 Suggested improvement

An extra checkbox marking a change as ‘minor’ would give an editor a choice of whether there needs to be sent an email or not.

8.7 Chat

The developers were available to the customer through a chat interface. The chat was used to directly discuss details of items and to reach the developers when needing quick answers. A copy of the resulting chat was often copied into the ticket system for reference on agreements and discussions.

8.8 Email

Email was used for discussions and for forwarding information that were not a part of the ticket system. The ticket system would also generate emails to appropriate recipients for each update in the ticket system.

8.9 Acceptance testing

Acceptance testing was done by the customer. Except for the project team testing, some external users were also given access to the system to help with testing releases. Most bugs were discovered in this fashion and were reported as tickets.

9 Evaluation of engineering activities

9.1 Achievements

9.1.1 Studies

A good understanding of T-SQL was achieved, initially from studies of tutorials and documentation and then later furthered by learning from the examples of the supplier’s database migration expert.

The studies on data migration projects gave a good understanding of what activities were necessary, and allowed for a better flowing project with an initial understanding on what parts were to come.

9.1.2 Mapping old databases

The mapping of the database was largely successful, only a single of the necessary tables could not be mapped. The end result detailed the old database column by column and helped the supplier’s data migration expert in making an accurate transfer of the data between the systems.

9.1.3 Database migration

As a ‘technical consultant‘ I spent about 300 hours on the data migration project and the suppliers migration expert spent about an equal amount of time, totaling about 600 working hours. The majority of the time was spent on researching and understanding the old database and figuring solutions for how to map it to the new system’s database.

The final solution for preparing data for migration between the systems was documented and can, which is unusual for migration projects, be partially reused when other organizations make the same move. Further work would have to be done on the import side since it is more complex to import data into a running system than it is to import into an empty system.

The process also involved a lot of communication with external stakeholders. The old system contained a variety of modules, a lot of which could not easily be transferred to the new system. A lot of decision calls were made both on quality versus cost and whether data should be migrated at all. While all the data remains in the data export, it is not easily obtainable when residing in a database without the accompanying software displaying the data. The stakeholders had to be informed and take measurements prior to the final ‘deletion’ of this data.

In the few cases were losses of data were discovered after the final migration, the data could be recovered by the insertions of scripts on the new database.

9.2 Lessons learned

9.2.1 Studies

9.2.2 Mapping database

As Morris proposes, data migration is a large part in the development of a new system based on old data.

9.2.3 Data migration

Cost and value are major factors in deciding the extent and the quality of the migration project. There is thus a need to weigh the value of the different parts of the old system; making decisions on where quality is a necessity, where errors are acceptable and which data may not be needed at all. These values are necessary to define breaking points where acceptable levels of quality have been reached.

As there will be inherent problems in the system which are unknown at the project start, the values will likely be difficult to define with any useful level of accuracy. If defined with accuracy “99% correct” it is possible that inherent problems may make it impossible to reach and/or know when a breakpoint has been reached, possibly making costs soar or ending when there are still critical errors present. Levels such as ‘system-critical’ and ‘low importance’ allows for fine-tuning as the migration project progresses.

Four different kinds of break points were used in the project:

- The data was deemed to have been transferred successfully with no known errors.

- The perceived value of migration of the data to the new system does not exceed the cost of the data migration (used for sets for which migration were complex and the data not critical to transfer, or where the target system could not contain the data thus costing in system development)

- Migration will be cheaper to make manually than automatically (used for small sets of data with complex migration logic)

- The cost of improving quality further becomes greater than the perceived value of the quality errors (used in large sets of data containing many exceptions and errors)

Other items to note:

- Peer review is recommended on migration code to find errors, since it’s a ‘one-time’ process and errors become more difficult to correct the longer those data are used live. This could have led to the discovery of some errors which led to costly corrections further up the road.
- Take special care of all tables that are partially filled with null values, it is possible they require another value to be filled in. Some of these cases were discovered, but at least one caused errors in the new system. A more consistent approach to these tables could have discovered such problems.
- User testing needs to be employed to find errors in the migration, users will take note of items that a system developer will not.
- Temporary exceptions introduced during development need to be documented in such a way that they can be tracked and resolved at the right time. As it were, at least two temporary solutions made it into final migration, causing loss of data.

9.3 To do

Having open invoices on the migration day complicates the migration procedures and should be avoided if possible. A possible solution for this would be to keep the old system running in parallel until all old invoices are closed.

The old system did in this case have a lot of different statuses applied to its members and many exceptional rules in the logics of the system. To decrease the cost of a migration to a new system, an overview of the existing business rules could be done, with the goal of decreasing the complexity.

10 Evaluation of advisory activities

10.1 Achievements

As a technical advisor I would step in when there were complex technical issues which the customer could not explain to the supplier or when the supplier had misunderstood the customer’s requirements.

In this advisory function I created for example:

- A straightforward excel document which expressed what the customer had documented about access rights in a hard to read text document cross-referencing over many pages.
- Several graphical images suggesting the functions necessary in the user-interface.
- A programmatic specification of the Swedish social security number checksum
- A specification detailing the requirements on how to handle mandatory data in old versus new data.
- Detailed descriptions of the different member statuses and how they should affect the system
- Tools for comparing invoice output of the old and the new system

10.2 Lessons learned

The supplier may have difficulties in understanding the exact demands of the customer, leading to problems, when the customer cannot accurately describe those problems. The supplier may make wrongful assumptions when the specifications are not clear enough. The customer having a person with technical knowhow combined with an understanding of the business process gives an advantage in getting a system that delivers what the customer wants instead of a system which is the supplier’s interpretation of what the customer wants.

11 Evaluation of Scrum method

11.1 Achievements

I participated in 11 out of 12 sprint planning meetings during the first development period.
I continuously monitored all the movements of the bug tracker system, adding input wherever needed, creating a total of 48 issues of the 359 issues generated in the period April – September 2010. In some instances chat, email or telephone conferences were used to communicate around more complex or urgent issues.

11.2 Lessons learned

The short iterations of the Scrum method are centered on accomplishing publishable results. A problem with this approach is that the ‘technical debt’ (see chap 1.4.2), may not be dealt with in the effort to reach the attained goals. In Scrum a fix can be applied rapidly, but there seems to be no mechanism in Scrum to analyze whether an application of a fix will introduce errors in other parts of the system.

From an outside view it seems that Scrum may have a lack of a natural point where the team takes a step back to gain an overview of how minor changes introduced will affect the system on a whole.

The focus on fixing the immediate problems did in this project lead to long-term damage evolving into more complex problems as the system evolved and the data was changed over time. A fix of one problem, without an analysis of the effect of that fix on the rest of the system lead to errors propagating through the system. As the data evolves over time, it became difficult to filter the original errors compared to data that should have a similar status or which have been manually fixed in the meantime.

Working on a live system, the short iterative cycles of Scrum might introduce errors which quickly propagate through the system. In a live environment that development can incur problems with real results. The live system will dynamically change, and after some iteration of changes and manual user input, the errors introduced become more complex to fix. The problems need to be filtered to find which are caused by the original error, such that the fix is not applied to data which has a similar status as the erroneous data or which have already been fixed manually.

11.3 To do

11.3.1 Technical debt management

It seems that Scrum as a method has as one of its weaknesses that it will veer toward deliberate technical debt generation, both in the reckless and the prudent form (as defined in chapter 1.4.2). This is caused by the fact that the goal is iteratively deliver a product improvement that can be demonstrated to the customer, which is external quality. The other goal of keeping good internal quality (the opposite being technical debt), might thus easily be put secondary. Scrum might need to integrate further activities to better control the technical debt in a project.

Some suggested solutions:

- Setup a measurement system to some degree measures the amount of extra time spent on a story caused by technical debt. In this could be included the time spent researching code because of missing documentation or time spent on bugs caused by ill-structured code. The supplier can then communicate with the customer an actual value of the debt, saying for example: ‘the debt cut down productivity in this sprint by 35%’ or major identified obstacles can be setup as stories of code improvement.
- Make the project test-driven, such that a change in one part of the system does not as easily wreak havoc in another part.
- Force cleanup sprints into the project, just as Scrum contains mandatory activities that might not be immediately productive (retrospectives, slack time or even daily scrums), a cleanup activity is not productive from the customer’s point of view. A cleanup sprint could be otherwise arranged in the same fashion as a normal sprint with stories defining what needs to be cleaned up.

12 Conclusion

12.1 Data migration

Understanding a database that someone else designed without any documentation is a complex task. If you further do not have an understanding of how the functionality that created the data it becomes difficult to achieve a good result.
My role in mapping the old database and all the requirements took of the weight from the person responsible for importing the database and the project manager. As I had an insight into how the old system worked and the importance weight of different kinds of data, it allowed for quicker and more precise decisions. This necessity of insight into what the old data represents confirms Morris´ Golden Rule #1 of migration: “Data migration is a business not a technical issue”.

12.2 Acting as advisor

My participation as a technical consultant was much appreciated by both the customer and the developer. There were sometimes misunderstandings, or difficulties in explaining concepts between the customer and the developer. As a person with insight into both the customer organization and an understanding of programmer thinking I could explain in the explicit detail necessary for a programmer the requirements or options available. My understanding of both sides also allowed designing solutions which met the requirements of the customer in a technically feasible and thus cost-efficient way.

12.3 Evaluating the Scrum method

The Scrum working method both allowed and required the customer to participate actively in the development. Scrum thus enforced a close collaboration that enhanced the development process and could concentrate the focus of the development team on problem areas. The method thus allowed for changes in the original request in the middle of development and allowed for the functions to be developed in the order preferred by the customer. For a project such as this where the requirements were not precisely defined from the beginning and where there was a very tight deadline with functional requirements changing during development, the Scrum method seemed to fit very well.

This benefit came at a cost however; the customer has to spend a lot of working time on assisting in the prioritization of the project. The tools for actually managing the project on the customer side could definitely be improved. The developer and the customer not having access to the same documentation increased the workload for the customer unnecessarily and probably increased misunderstandings.

In a stricter environment, with precise requirements, I would imagine that the Scrum method would be more ‘internal’ to the developer and not require as much interaction and participation from the customers side.

13 References

[1] Wikipedia, "Scrum (development)," 2 Mar 2012. [Online]. Available: http://en.wikipedia.org/wiki/Scrum_%28development%29. [Accessed 3 Mar 2012].

[2] K. S. a. J. Sutherland, "Scrum Guide 2011," July 2011. [Online]. Available: http://www.scrum.org/storage/scrumguides/Scrum%20Guide%20-%202011.pdf. [Accessed 29 January 2012].

[3] J. Morris, Practical Data Migration, Chippenham: BCS, 2006.

[4] Skatteverket, "SKV 704 utgåva 8," September 2007. [Online]. Available: http://www.skatteverket.se/download/18.1e6d5f87115319ffba380001857/70408.pdf. [Accessed 29 January 2012].

[5] M. Fowler, "Technical Debt Quadrant," 14 October 2009. [Online]. Available: http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html. [Accessed 29 January 2012].

[6] Svenska Scoutförbundet, "SSF OffertUnderlag - Bilaga 1," Stockholm, 2009.

[7] Svenska Scoutförbundet, "SSF OffertUnderlag," Stockholm, 2009.

[8] Svenska Scoutförbundet, "SSF Offertunderlag - Bilaga A," Stockholm, 2009.

[9] Svenska Scoutförbundet, "Bakgrund - Svenska Scoutförbundet," Svenska Scoutförbundet, 2009. [Online]. Available: http://www.ssf.scout.se/service/scoutnet/ovrigt/bakgrund/. [Accessed 19 Feb 2012].

[10] H. Kniberg, Scrum and XP from the Trenches, United States of America: infoQ, 2007.

[11] fianga.com, "Microsoft SQL Server," Fianga.com, 29 January 2012. [Online]. Available: www.fianga.com. [Accessed 29 January 2012].

[12] Riksdagen, "Svensk författningssamling - Personuppgiftslag (1998:204)," 2010. [Online]. Available: http://www.riksdagen.se/webbnav/index.aspx?nid=3911&bet=1998:204. [Accessed 29 January 2012].

[13] Microsoft, "Windows Powershell," 15 February 2012. [Online]. Available: http://technet.microsoft.com/en-us/library/bb978526.aspx. [Accessed 4 Mar 2012].

[14] Python ODBC library, "pyodbc - Python ODBC library," pyodbc, 13 Januari 2012. [Online]. Available: http://code.google.com/p/pyodbc/. [Accessed 29 January 2012].

[15] Open Link Software, "Open Database Connectivity Without Compromise!," 1993. [Online]. Available: http://www.openlinksw.com/info/docs/odbcwhp/tableof.htm. [Accessed 4 Mar 2012].

[16] gnu.org, "Definitionen av fri programvara," gnu.org, 2007. [Online]. Available: http://www.gnu.org/philosophy/free-sw.html. [Accessed 19 July 2010].

[17] Microsoft, "Download: SQL Server Database Publishing Wizard 1.1," Microsoft, [Online]. Available: http://www.microsoft.com/download/en/details.aspx?id=5498. [Accessed 19 Feb 2012].

[18] Creativyst, Inc, "How To: The Comma Separated Value (CSV) File Format," Creativyst, Inc, 2010. [Online]. Available: http://www.creativyst.com/Doc/Articles/CSV/CSV01.htm. [Accessed 4 Mar 2012].

[19] python.org, "13.1. csv — CSV File Reading and Writing - Python v2.7.2 documentation," python.org, 19 Feb 2012. [Online]. Available: http://docs.python.org/library/csv.html. [Accessed 19 Feb 2012].

[20] msdn, "Features Supported by Reporting Services in SQL Server Express," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/cc281020.aspx. [Accessed 19 Feb 2012].

[21] Collabnet, "CollabNet - The Leader in Agile Application Lifecycle Management," Collabnet, [Online]. Available: http://www.collab.net/products/scrumworks/. [Accessed 19 Feb 2012].

[22] Redmine, "Overview - Redmine," Redmine, [Online]. Available: http://www.redmine.org. [Accessed 19 Feb 2012].

[23] php, "PHP: Arithmetic operators - Manual," php, [Online]. Available: http://php.net/manual/en/internals2.opcodes.mod.php. [Accessed 19 Feb 2012].