Migration process evaluation and design

by

Henrik Bylin

LIU-IDA/LITH-EX-A--13/025--SE

2013-06-10
Final thesis

Migration process evaluation and design

by

Henrik Bylin

LIU-IDM/LITH-EX-A--13/025--SE

2013-06-10

Supervisor: Mikael Svensson (Medius)
Anders Fröberg (IDA)

Examiner: Erik Berglund
Acknowledgements

I would like to thank my girlfriend Anneli and my family for their strong support during the hard work with this thesis.
Abstract

Many organizations find themselves at one time in a situation where one of their IT systems has to be replaced with another system. The new system has to get access to the legacy data in order to function.

This thesis examines a migration case where the company Medius has made a new version of their product Mediusflow that has a completely new data model. The problem is then how to migrate data from the old to the new version.

The thesis first examines different data migration processes to find one suitable for the specific case. Then a prototype migration solution is developed using the chosen process to show that the migration can be done.

In the end conclusions are drawn from the experiences gained during the thesis to what key factors are important for successful data migrations.
# Table of contents

Acknowledgements ................................................................. 1

Abstract ..................................................................................... 2

1 Introduction ................................................................................. 5
  1.1 Background ............................................................................ 5
  1.2 Objective ............................................................................... 5
  1.3 Problem description ............................................................ 6
  1.4 Limitations ............................................................................ 6
  1.5 Method .................................................................................. 6

2 Theory ......................................................................................... 7
  2.1 Database technologies ......................................................... 7
    2.1.1 Relational database ....................................................... 7
    2.1.2 Object-relational database ........................................... 7
    2.1.3 Object-relational mapping ............................................ 7
  2.2 Object-relational impedance mismatch ............................... 8
  2.3 Persistence ............................................................................. 8
  2.4 Hibernate/NHibernate .......................................................... 8
  2.5 Transact-SQL ........................................................................ 8

3 Data migration processes ......................................................... 9
  3.1 Definition of migration and why migration is performed .... 9
    3.1.1 Related data movement processes ................................ 9
  3.2 The Butterfly methodology ............................................... 10
  3.3 ETL ...................................................................................... 11
  3.4 Meta-modeling approach .................................................. 12
  3.5 Process model .................................................................... 12
  3.6 The Data Warehouse Institute best practice model ......... 14

4 Background for the process design ......................................... 15
  4.1 Migration environment ....................................................... 15
    4.1.1 Source system ............................................................. 15
    4.1.2 Target system ............................................................ 15

5 Design of the migration process .......................................... 16
  5.1 The process ......................................................................... 16
6 Prototype migration solution .........................................18
7 Analysis ..................................................................................21
  7.1 Analysis of the prototype ..................................................21
    7.1.1 SQL script approach ..................................................21
    7.1.2 Platform API approach .................................................21
    7.1.3 Reasons for the chosen approach .................................22
  7.2 Process improvements ..........................................................23
    7.2.1 Process model .............................................................23
    7.2.2 Stakeholders involvement .............................................23
    7.2.3 Weak and changing requirements .................................23
    7.2.4 Static source and target systems .................................24
    7.2.5 Lack of system documentation ....................................24
    7.2.6 Changing development environment .........................24
8 Results ....................................................................................25
9 Discussion ................................................................................27
  9.1 Unfinished target system ....................................................27
  9.2 Alternative solutions ..........................................................27
  9.3 Cooperation troubles ........................................................27
10 Future work .............................................................................28
  10.1 More complete testing ......................................................28
  10.2 Cutover and fallback strategies .......................................28
11 References ..............................................................................29
1 Introduction

1.1 Background

Medius offers solutions to increase efficiency and to automate business processes. The product is the workflow platform Mediusflow. Mediusflow currently provides applications such as invoice automation and purchase-pay. Mediusflow makes it possible to build different workflow applications on top on the platform.

The latest version, version 11, of Mediusflow has been significantly re-written using a different database model than earlier versions. In version 11 an object-relational database model is used instead of a relational model as in previous versions. This means that possible solutions have to be evaluated for the migration of the data from the old to the new system.

In previous versions, master data from the Enterprise Resource Planning system has been downloaded to Mediusflow database from the customers ERP. This data has been mapped into relational database tables generated by the Mediusflow platform.

1.2 Objective

The main objective of the thesis is to develop a process to migrate data from one relational database to another. The database models are different but contain almost the same information, they are used by an application that performs the same task. The process will be applied on the Mediusflow database where application settings data from Mediusflow version 10 to version 11 will be migrated.
1.3 Problem description

When using agile development techniques the software structure can change due to changing requirements. The database model might need to be changed due to this.

Changing database models can be a complex task and therefore a process for migrating data between database models is required. The purpose of this thesis is to develop a process to aid the migration of data between two relational databases of different structure.

Medius AB are currently in a situation like this were their database model have changed significantly. Therefore the process will be applied by us on the database of their Mediusflow software and later evaluated. The questions we like to find an answer to are.

Can the migration between the source and target system be performed?

Are there processes available that can support the migration case?

What are the key factors for a successful migration?

1.4 Limitations

Migration of data from other modules such as transactional data, integrations, purchases, agreements and master data will not be included in this thesis. Testing of the prototype beyond simple unit testing is not a part of the thesis. The evaluation of the migration processes is constricted to the specific case.

1.5 Method

The following steps describe the general working process of the thesis:

1. Basic theory study
2. Design of development process
3. Iterative development
4. Evaluation of prototype
5. Literature study as base for possible future work

This method follows the principles of experimental software development. Were knowledge of a subject is explored though a practical experiment in which a prototype for the problem is built. This prototype can later be evaluated to come to conclusions on what knowledge can be learned from the experiment. After this contained experiment is completed there is often more questions raised that can lead to more experiments, these are described in a future work section.
2 Theory

In this section the theory behind different migration processes and database technologies are described.

2.1 Database technologies

2.1.1 Relational database

In relational databases is structured and stored in two-dimensional tables. These tables consists of rows and columns, each of these rows is called a tuple.

A tuple consist of a collection of related values which belongs to an attribute which are represented in columns. Each of the attribute has a data type for example string, integer, char, date, text which defines the type of data it can contain. This means a string cannot be stored in an attribute with integer as data type.

To avoid inconsistencies in the database there are restrictions on the data, which is called constraints. The main constraints are the integrity constraints which are: key, entity integrity, referential integrity, domain and null.

The order of the tuples is insignificant in an relational database, though each tuple has to be unique. This is achieved by an identifier. The identifier is often called the primary key in the table.

In most cases the primary key is an id with the data type integer which is incremental for each tuple. By using the primary key a tuple can refer to another tuple by using it as a foreign key. This is the way tuples create relations to other tuples, in the same or in another tables. (2)

2.1.2 Object-relational database

Object-relational databases are similar to the relational databases but instead use an object-oriented model layer on top of the relational database.

It combines the advantages of the relational databases robust transaction and performance management features with the modeling features of the object oriented databases such as flexibility and scalability.

It also includes other object oriented capabilities such as identifiers, references, inheritance, operations, abstract data types and collection data types. (3)

2.1.3 Object-relational mapping

Object-relational-mapping (ORM) is a technique for converting data between objects oriented languages and relational database systems. This creates a layer that gives the illusion of an object database for the developer, while in practice a relational database is used.

The ORM layer lets developers take the same advantages of the object oriented techniques for persistent objects as for transient ones. ORM software enables use of a relational database in an object oriented manner.(4)
2.2 Object-relational impedance mismatch

The object-relational impedance mismatch is a name for the conceptual and technical issues that are encountered when using a relational database by a program written in an object oriented programming language. This mismatch happens because the ideas of object orientation and relational tables are very different in concept and function. (4)

2.3 Persistence

Persistence refers to the characteristic of objects and states that outlives its creator process. Without persistence all data in software would be lost when the execution of the process is terminated. In practice persistence is achieved by saving the data from the random access memory to a nonvolatile storage such as a hard drive. To manage the persistence a database management system (DBMS) is often used to relieve the developers from some of the issues involved in persistence for example data recovery and concurrency control. The DBMS also provide the developers with functionality for access control and auditing.(5)

2.4 Hibernate/NHibernate

NHibernate is a object-relational mapping framework for Microsoft's .NET platform that provides transparent persistence for Plain Old CLR Objects (POCO). POCO is a name for a simple CLR object with a no arg constructor. NHibernate maps the .NET classes to relational database tables and provides query support for retrieving/storing objects in the database. For a more detailed explanation of object-relational mapping see the object-relational chapter.(4)

2.5 Transact-SQL

Transact-SQL or T-SQL is a proprietary extension to the SQL language used in Microsoft's SQL server products. T-SQL expands the SQL standard so it includes procedural programming, local variables, various additional functions such as string processing, date processing, mathematics. These additional features makes T-SQL a Turing complete language.(6)
3 Data migration processes

In this section the different migration processes that were found during the initial theory study to be suitable to the migration case in this thesis. These processes have inspired the actual process used in the migration case of the thesis. That process is described later in the report in the section 5, Design of the migration process.

3.1 Definition of migration and why migration is performed

The definition of migration can vary a bit but can be loosely described as a movement of data. This chapter takes up some of the definitions on the subject.

Mattes & Schulz (2011) defines migration in the following way: “Tool-supported one-time process which aims at migrating formatted data from a source structure to a target data structure whereas both structures differ on a conceptual and/or physical level”

John Morris (2012) defines data migration in a slightly different way in his book as: ”Data migration is the selection, preparation, extraction, transformation and permanent movement of appropriate data that is of the right quality to the right place at the right and the decommissioning of legacy data stores”

IBM, a large company in the data migration and storage business defines it as: “Data migration is the process of making an exact copy of an organization’s current data from one device to another device – preferably without disrupting or disabling active applications – and then redirecting all input/output (I/O) activity to the new device”

Reasons to migrate could be the following:

- System runs on obsolete hardware, maintenance becomes expensive
- Due to lack of understanding the system, maintenance becomes expensive
- Problems with integrating the system due to the lack of interfaces
- Technological progress and upgrades
- Regulatory requirements
- Introduction of a new system
- Relocation of data centers

While the reasons to migrate and definitions may vary the data migration in its core is often similar.(7,8,9)

3.1.1 Related data movement processes

The concept data migration is often used misunderstood or mistaken for other forms of data movement. This table describes different movement techniques. Russom (2006) defines the different types of data moving techniques as the following table shows. (10)
### Table 1: Data movement techniques

<table>
<thead>
<tr>
<th>Type of technique</th>
<th>Number of data sources and targets</th>
<th>Diversity of data models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consolidation</td>
<td>Many to one</td>
<td>May be homogeneous or heterogeneous</td>
</tr>
<tr>
<td>Migration</td>
<td>One to one</td>
<td>Always heterogeneous</td>
</tr>
<tr>
<td>Upgrade</td>
<td>One to one</td>
<td>Usually heterogeneous</td>
</tr>
<tr>
<td>Integration</td>
<td>Many to many</td>
<td>Extremely heterogeneous</td>
</tr>
</tbody>
</table>

#### 3.2 The Butterfly methodology

The Butterfly methodology is a gateway free approach to migrate legacy data. Often organizations want to keep their old legacy systems operational and use gateways between them and newer systems to exchange data. In this way there is no actual permanent migration of the data. The old system and its data is accessed through the gateway from the new system. The use of gateways has the drawback of increasing complexity, therefore the best way to reduce future problems is to make the migration gateway free.

When the migration starts the legacy data store is frozen in a read only state, changes in the old system is directed by an Data-access-allocator service to auxiliary temp stores that contain the changes from the first store. Changes from the first temp store are saved in another temp store and so on. A Chrysalizer transformation application is then used to transform the data from the legacy data store into the target system.

When the transformation of the first store is complete, transformation will begin on the next store. This process goes on until the current temp store is smaller than a certain threshold size. The legacy system is then stopped and the final changes are transformed to the new system. The new system can then be brought online with the same information as the legacy system had when it was taken offline. This approach gives the advantage that under no circumstances are the legacy system unavailable for a longer period of time.\(^{(11)}\)
3.3 ETL

ETL or (Extract, Transform, Load) is a migration process that does exactly what its name says, it extracts it transforms and loads the data into the new platform. Firstly it downloads and filters the data from the legacy system to a temporary storage. Filtering is done because often the complete data set from the old system is not needed in the new system, it might for example be old records of things that are not in use or data that is not applicable in the new system. An advantage of downloading the data to a temporary storage is so that the migration won’t interfere with the old system still in use.

The transform phase changes the data to conform to the target data model. There might for example be conversion of data types or structural changes. There is also possible that the data have to be expanded or reduced depending on the different architectures. If the data is reduced some information is lost in the transformation because the new data model has fewer details. Expansion occurs when the new data model has more details than the old.

After the data has been transformed to fit the new data structure it can be loaded into the new platform. The loading of data into the vendors new system can be done in several ways direct approach, simple API and workflow API.

Using the direct approach data is inserted into the new systems internal database directly. This means there is no built in error support except for triggers that might exist in the database. The advantage is that there is a low cost for the vendor because it requires very little effort for them. Conversely the cost for the migration team can be high if they need extensive training or has to implement many consistency checks.

The simple API approach means that there is an API that the migration team can use for insertion of data. This API has to be constructed by the vendor and copies data from its API tables into internal system tables. This means less work for the migration team but can require some work from the vendor instead to design and build the API if it is not already present in the system.

The workflow API approach uses the same API that is used under normal operations in the system. This means that there are the same consistency checks as there would be if the data is entered by a user. After the API has been constructed there are low costs for both the migration team and the vendor. (12)
3.4 Meta-modeling approach

The meta-modeling approach is a way to migrate data between relational databases. When using the method both the source and target models are presumed to be known in advance. Meta models describing the data models of both the source and target databases are created. This means that the conversion process is not dependent upon the original data models. After the creation of these meta models of the source and target database, the mapping of data between the databases can be done.(13)

3.5 Process model

The Process model is a migration model that consists of 14 phases. This in turn can be divided into four stages, Initialization, Development, Testing and Cut-Over.

Figure 3: Process model for data migration

The Initialization stage consists of phases 1 to 3. The first phase, Call for tenders and bidding, is performed if the organization decides that the migration will be done by a third-party.

The second phase, Strategy and pre-analysis, are performed to identify which business concepts are related to which tables and attributes. It should also define how the migration should be carried out and the project organization shall be set up.

The third phase, Platform setup defines the infrastructure and which technical solution will be used for the migration. This includes both hardware and software. The platform is just used for the migration and is separated from both the source and target application.

Development stage consists of phases 4 to 7. The fourth phase, Source data unloading are also called the extraction phase and unloads relevant data from the source system. This might be done several times to keep the migration data up-to-date to the source system, depending on the size of the data.
The fifth phase, Structure and data analysis for source and target is performed to analyze and learn the structure of the source data.

The sixth phase, Source data cleansing are performed to clean data and improve the quality of the source data before migration. It is recommended to do this before the next phase the transformation phase. Because of the added complexity when the data are transformed.

The eighth phase, Data transformation are an iterative process and is performed to map the source data from to the target data structure by using transformation rules.

The Testing stage consists of phases 9 to 12. The ninth phase, Data validation validates the correctness, completeness and consistency of the data that has been transformed to ensure that it complies to the requirements.

The tenth phase, Data migration process tests validates the whole migration process from the unloading of the data from the source to the staging area and finally to the target data source. It can also measure the migrations performance.

If the results of these tests are not satisfactory the transformation rules can be adjusted to better fit the requirements.

The eleventh phase, Target application tests focuses on validating data when it is used in the target application. This asserts that the target application works as intended with the migrated data.

The twelfth phase, Integration tests and final rehearsal tests the integration between the target business application and other dependent business applications.

Finally the Cut-over stage, consisting of phases 13 and 14. The thirteenth phase Productive migration, are done when all the tests have passed and fulfilled the requirements. This is the actual migration step when the data are moved.

The Finalizing step are when the cut-over is performed which are the transition between the source and the target application.
3.6 The Data Warehouse Institute best practice model

The Data Warehouse Institute’s data migration development and deployment model are divided into six phases. First one preliminary startup phase called Solution Pre-Design, followed by an iterative process with 5 phases: Solution Design, Data Modeling, Data mappings, Solution Development and Solution Testing.

In the first phase, Solution Pre-Design the requirements for the migration are gathered to be able to build a detailed project plan and define the deliverables for the project and its constraints.

In the second phase, Solution Design tasks are separated depending on their dependencies to each other.

In the third phase, Data Modeling the target database structure are built.

In the fourth phase, Data Mapping data fields are mapped from the source to target database. Transforming of the data is done according to the different data models.

In the fifth phase, Solution Development the actual migration program which migrates the data from source to target is developed using the mapping rules constructed in the previous phase.

Once the development sprint of the solution is done, it’s time for the sixth phase, Solution Testing where a subset of data from the source is selected to test the current solution that has been built.

As this is an iterative process, once one module is done another can be started, or improve the one which is being built.

Once the solution has passed all tests and covers all requirements needed it is time for Solution Deployment.

This is the final phase where the solution is handed over and deployed into the production system. The deployment includes final tests with end users and administrators have to ensure that performance and data quality requirements are fulfilled.
4 Background for the process design

Medius has earlier upgraded their customer systems, but in most cases the database has only needed to be extended, due to that the database model only has had smaller changes. These changes have been for example, added a column in a database-table with info that has been added due to new requirements or features.

The solution which will be built in this thesis will however be different. The new database model is completely changed which requires a migration tool, migrating data from the old database model to the new database model. Besides having different relations between objects in the database, the data is also presented differently, which requires that the data has to be transformed using transformation rules.

The amount of tables to be migrated is limited. Another issue to take in consideration is that most articles mentions migration as a one-time-task, whilst in our case the migration tool will be used at every customer who will upgrade their system. This means it can be hard to test the solution for every possible dataset, depending on what kind of data the customer has entered into the database. The data quality will also most probably differ between different customers as well.

Medius will be working on the new system while this thesis is under development, which means some parts of the system might not be ready. The requirements of which parts of the system won't be presented in the thesis and will be referred as modules.

4.1 Migration environment

This section describes the different layouts of the source and target systems of the migration solution.

4.1.1 Source system

The source system is accessed via the ADO.net interface which is a component in the .net framework to access data and data services. The database engine itself is a Microsoft SQL server 2008. The database is called directly from the application itself though the ADO.net interface.

4.1.2 Target system

The target system is using a nHibernate for object relational mapping form the business logic objects to the relational database tables. The actual processing logic of the system is built on top of a platform that takes care of tasks such as database communication or communications to external systems such as an enterprise resource planning system.
5 Design of the migration process

Most of the articles found mentions migration as a one-time-task, while in the case of this thesis the migration will be ran over and over again as a part of upgrading customers using earlier versions of Mediusflow.

Due to this and the relatively small scale of the migration a decision to make a custom process were taken. Besides taking concepts from the existing processes that was found, we also added some own concepts which we thought would fit this specific migration. The main layout of the process follows the ideas of the Data warehouse best practice model.

5.1 The process

The migration process is divided into three phases where each phase has a major objective. The phases are: requirement analysis, design and validation.

During each phase there can be several iterations/sprints of a specified length, for example one week.

In every iteration there is a list of work items that is the focus for the iteration. These are recorded in a document so that it is clear to all project members what is to be done. In the end of every iteration there is a meeting where the progress of the current iteration is recorded and evaluated and the focus of the next iteration is decided.

Because of the separate nature of the database different tables/modules can be in different phases during the development if the database model allows for that. For example the migration software of a smaller part of the database that is loosely coupled with the reminder can be almost finished, while at the same time a more complicated part can still be in the requirements phase.

Requirements capture and elicitation

The requirement analysis is divided into two steps, one is identifying the requirements to the tables of the old database and the other step is identifying the requirements to the tables of the new database.

This is done by studying the database and discussing its structure with employees that have knowledge of them. Studying the database can be done in a variety of ways such as study its database diagrams to see relationships.

Once the project team has a decent understanding of how the database relations are connected to each other the study of how to map the old data into the new database model can start. Once the tables containing data that is to be migrated are identified it is time to identify the attributes and the foreign and primary keys of each of the tables to understand the relationships in the database.
A good way to start is to identify the tables containing data to be migrated that is dependent on as few as possible other tables. This data can then be relatively easy migrated to the new model. After confirming these modules/tables are correctly migrated work can start on tables referencing to them.

To capture and elicitate the requirements there might be a need to have workshops with domain experts in the different data models if they are not already part of the migration team. The experts can provide detailed information about the data models that can be hard or impossible to discover otherwise.

**Design of migration software**

When migrating the data to the new database the primary keys might have changed and therefore a temporary relation table will be constructed. The temporary table will consist of the primary key or keys of the old table and the primary key of the new table. It will be used to map the new relationships.

The actual migration is then done in an exploratory and iterative manner with one module migrated at the time. Starting with the simpler data that have fewest relationships to the other tables in the database. The relationships is then explored and verified against someone with domain knowledge. Some data might have change format in the different models or can be present in one model but not the other. A committee of stakeholders in the migration process such as domain experts for the respective data models will have to take decisions on how to treat the migration in each specific case where there are conflicts in the database models.

The data might be migrated in different ways depending upon what is best suited for that particular dataset. For instance some data can be migrated by manipulating the database itself using SQL-queries, some can be extracted from the old database and inserted into the application as new objects via some migration application that is constructed to migrate the data.

**Verification and transition**

In the final phase the objective is to verify that the software works according to specification. The extent of testing can differ between different projects. In our case we will perform simple unit tests to confirm that the migration is performed correctly. During testing it can be revealed that the module does not work as intended. The work on that module has to be iterated back to the requirements phase to ensure that no unknown requirements have been discovered and thus let to a faulty implementation. If there is new requirements the specification has to be updated and the additional functionality has to be implemented by going through the design phase again. Finally after reimplementation of the new requirements the module is back in the verification phase and can be tested again.
6 Prototype migration solution

The prototype solution developed with our process is based upon the use of SQL scripts to migrate the data from a standard MS SQL server 2008 to another using the object relational mapping tool called nHibernate. The two database engines is therefore of the same kind but the database schemas have changed completely.

Before the migration solution is executed some data called masterdata is loaded into the new system from the company's enterprise resource planning system by the target system itself. This was simulated in our prototype because the loading solution was not developed yet.

The migration scripts are loaded into the old database by a windows shell script for easier use, after the scripts have loaded the migration script itself can be run to execute the migration.
When the actual migration is started a series of checks is performed on a copy of the source database by the migration solution. These checks assert that certain restrictions are met on the old data that is required in the new database. Such as for example fields that don't allow null are set to a non-null value, otherwise errors will be thrown when trying to insert them into the new database.

```
CREATE PROCEDURE check_database @fail int output
AS
BEGIN
    SET NOCOUNT ON;
    -- User
    DECLARE @count int;
    SET @fail = 0;

    SET @count = (SELECT COUNT(DISTINCT EMAIL) FROM [USER] WHERE EMAIL is NOT NULL)
    DECLARE @diff int = ((SELECT COUNT(*) FROM [USER]) - @count);
    IF (@diff > 0)
        BEGIN
            PRINT CAST(@diff as nvarchar(10)) + ' values of User->Email are not UNIQUE'
            SET @fail = @fail + 1;
        END
    IF (@fail = 0)
        PRINT 'Medius 10 database is OK'
    PRINT '===================================='
END
```
The migration script is built up of a central main procedure that calls the migration of other separate parts/modules of the database. For example there is the user, role and authorization group data.

```
CREATE PROCEDURE migrateData @dbName nvarchar(400)
AS
BEGIN
   SET NOCOUNT ON;
   DECLARE @check int = 0;
   EXEC check_database @check OUTPUT;
   IF (@check = 0)
   BEGIN
      BEGIN TRAN;
         -- ==MIGRATION PROCEDURES==
         -- User, UserConfiguration, UserContact
         EXEC migrate_User @dbName;
         IF ((SELECT COUNT(*) FROM migrate_errors) > 0) GOTO ERR_HANDLER
         EXEC migrate_Responsible @dbName;
         -- Role
         EXEC migrate_Role @dbName;
         IF ((SELECT COUNT(*) FROM migrate_errors) > 0) GOTO ERR_HANDLER
         -- RoleCompany
         EXEC migrate_RoleCompany @dbName;
         IF ((SELECT COUNT(*) FROM migrate_errors) > 0) GOTO ERR_HANDLER
         EXEC migrate_AuthorizationGroup @dbName;
         PRINT 'COMMIT'
         COMMIT TRAN
         RETURN 0
      ERR_HANDLER:
         PRINT 'Unexpected error occurred -> Rollbacked data!'
         SELECT TOP 1 * FROM migrate_errors;
         ROLLBACK TRAN
         RETURN 1
   END
   END
   GO
   EXEC migrateData 'HflowDatabase';
```

This approach gives the possibility to only migrate the parts of the database that is needed, or perform the migration in several steps by simple changes to the main migration script.

Each of these are modules that have their own data but are connected to each other through relations. If there is an error in some part of the migration the whole migration will be aborted and changes roll backed.
7 Analysis

This chapter contains the analysis of our migration. Both the migration process itself, the resulting migration solution and other experiences gained during the thesis will be analyzed.

7.1 Analysis of the prototype

During the initial stage of our development phase we had two different technical options in how to carry out the migration.

As described in the chapter Prototype migration solution we chose to implement the migration software as a couple of SQL procedures that are executed on the source database in order to transfer the appropriate data to the target system.

The alternative solution would have been to extract the data from the target database and then create new appropriate objects in the source database through an API.

A third possible solution could have been to use a migration software, but that was not an option according to the company.

These approaches to the migration have a number of advantages and disadvantages.

7.1.1 SQL script approach

The script approach only deals with the two respective databases and their design, the upper layers of the system can be ignored. The designing of the migration consists of studying the target database to see what relevant data can be extracted from the source and how its relations transfer to the target database.

This method has the advantage that it requires no change in the target system, the migration is done on the database level.

This means there has to be error checking implemented in the migration software to make sure the data conforms to the constraints of the target database.

7.1.2 Platform API approach

The API approach uses functionality in the target system to create new objects with data extracted from the source database. These objects are then saved into the target database as if they were created by normal usage of the target system.

This gives the advantage of the same error checking as if objects were saved by a normal user.

A disadvantage is that there has to be an way to access this API from outside the target system or someway to execute migration code inside the platform. If this functionality is not present it has to be created for the migration software to be able to insert the data.
7.1.3 Reasons for the chosen approach

After some investigation the SQL approach was chosen for our migration. The choice was based on several factors such as less dependency upon external knowledge, we only had to deal with the databases themselves.

Because of the unfinished nature of the target system there was no documentation of appropriate APIs.

We found no way to access the API from outside platform or execute our own software from within the platform. This made it difficult for us to continue on that path. We were also recommended by a architect of the target system to use the SQL approach.

These reasons together made us chose the SQL approach for the migration.
7.2 Process improvements
This chapter describes some of the most important issues encountered during our migration. These issues would be the first thing to improve or try to avoid when conducting a similar migration.

7.2.1 Process model
The process model itself and its lightweight iterative nature was well adapted to the development of the migration software. The iterative workflow made it possible to start with the migration of the smaller modules and use the knowledge learned there to ease the design of the more complicated modules.

7.2.2 Stakeholders involvement
A key factor in the success of every migration or software development project is the involvement of all the stakeholders in the development. Because only the stakeholders can know how important certain data is for their usage of the systems they need to work together with the requirements capture team. If for example the users of the data that is to be migrated is not part of the process, there might be crucial data missing after the migration because its importance was not understood by the migration team.

The problem in our case was that the stakeholders of the migration had no support from the organization in the form of allocated time to help the migration team. This resulted in that the success of the migration was reliant upon the goodwill of certain employees with key knowledge.

7.2.3 Weak and changing requirements
Weak requirements will almost always lead to changes in the requirements, this impacts the migration because work might have to be redone in parts where data was migrated in the way it was specified in the old requirements. The later in the migration process the more damage is inflicted when requirements change, therefore it is important to have them as clear possible as soon as possible. This is best done with close cooperation by the migration team and the systems users.

In our case the requirements had to be refined over the whole project because of the uncertainty if the specific modules priority.
Some modules were found to not exist in the target system and some were not implemented yet. This knowledge had to be discovered by the migration team when we refined the requirements by ourselves.
The requirements were vague because of the lack of effort put into the migration project from the other stakeholders then the migration team.
This on the fly refinement of the requirements could have been made smoother by stronger requirements in the beginning, as it cost extra time for the migration team to find out what was already known by other employees.
7.2.4 Static source and target systems

For a successful migration it is important that both the source and target system is in a static state. To be able to map the data from the source to the target the systems cannot be changing, if they are the mapping will have to be adapted.

This results in additional work for the migration team. Changing data in the source system can be handled to some extent with iterative migrations but if the data structure changes in either the source or the target the migration solution has to be changed according to the new environment.

The problem in our case was that the target system was still under development during the time we designed our migration solution.

We were told in the beginning that the development was quite finished and that there would only be minor changes to the data structures. That was not the case however, there was quite substantial changes made.

To make the migration even more difficult we were not informed of these changes, but have to find them out when the migration solution simply did not work after an update of the system.

7.2.5 Lack of system documentation

Lack of system documentation makes the migration extremely reliant on the domain knowledge of certain individuals in the organization. If these experts are unavailable or busy the progress of the development can be slowed or halted.

In this project the documentation was almost non-existent, this made the migration team reliant on employee knowledge. The problem was that there was only a few employees that had knowledge of the both the target and source systems. Most of them only worked with their small parts in either the target or the source system. This made information gathering of the systems more difficult than if there was at least some documentation.

7.2.6 Changing development environment

This problem is related to the non-static nature of the target system. During the development of the migration solution the development environment changed because of the unfinished state the target system was in. This required more extra time in order to change the environment and all required packages that were needed for the target system to function.
8 Results

This chapter explains the conclusions of the thesis. Answers to the questions raised in the chapter problem description are given.

Can the migration between the source and target system be performed?

The thesis has resulted in a migration solution that was working at the time it was tested for the modules that were possible to migrate. Because the system still was in development at the time the development of the solution stopped it is unknown if the solution still works, and if not how much changes are needed for it to function again.

Around half of the modules tasked for migration were successfully migrated, the rest were either not applicable to the target system or they were not implemented yet.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access control groups</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Chart of accounts settings</td>
<td>Masterdata</td>
</tr>
<tr>
<td>Agreement categories</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Authorization groups</td>
<td>Is migrated</td>
</tr>
<tr>
<td>Authorization groups / Price limits - general</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Authentication groups</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Distribution lists</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Exception types - general</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Matching rules</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Factoring settings</td>
<td>Masterdata</td>
</tr>
<tr>
<td>Purchase request templates</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Roles</td>
<td>Is migrated</td>
</tr>
<tr>
<td>Exception types - supplier</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Accounting templates</td>
<td>Is migrated</td>
</tr>
<tr>
<td>Price limits - supplier</td>
<td>Is migrated</td>
</tr>
<tr>
<td>Reference objects</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Authorization rules</td>
<td>Is migrated</td>
</tr>
<tr>
<td>User event settings</td>
<td>Does not exist in MF11</td>
</tr>
<tr>
<td>Authorization groups</td>
<td>Is migrated</td>
</tr>
<tr>
<td>Company settings</td>
<td>Masterdata</td>
</tr>
<tr>
<td>User</td>
<td>Is migrated</td>
</tr>
</tbody>
</table>

Figure 5: Migrated modules
Are there processes available that can support the migration case?

Most of the processes found were suited for large scale migration projects. They were also one time migrations for specific system setups. Because of the large scale of many of these migrations the processes were extensive, which in this thesis case would just be a burden.

The migration case in this thesis was of a much smaller scale and with multiple similar migrations that would be carried out in different organizations.

This lead to the decision to make a custom lightweight migration process based upon the Data warehouse institute best practice model.

The migration methodology's iterative workflow functioned very well for the migration case. It made it possible to migrate one module at the time, and use the knowledge learned from the first modules to greatly reduce the time needed for the development of further modules migration procedures.

What are the key factors for a successful migration?

The most important changes are in the organizations support of the migration as mentioned in the process improvements part in the analysis.

Of these changes the most important one is to have the organization and stakeholders support in the migration, as it was now the migration team was left on its own. If a number of key employees had not taken their time that was assigned to other projects to help us, it is unknown if the migration would have succeeded at all.
9 Discussion

Discussion on what has been achieved and what could be done to improve the result or what could have been done different.

9.1 Unfinished target system

Because of the target system a full migration could not be done. The target system were much less finished then what was apparent in the beginning, not even our supervisor knew how much more development that had to be done on it.

Enough modules were successfully migrated to know that the rest of them would be possible to migrate too, it would just require more work. So the unfinished state of the system did not impact the thesis except that it made developing the prototype harder. We had to find out for ourselves that parts of the migration broke when the target system had changed without us knowing about it.

9.2 Alternative solutions

An alternative solution to the migration case could have been to use a migration tool instead of developing a own custom solution. This could save a lot of time but can of course cost to buy and that cost could be higher as multiple migrations would be done at Medius customers.

Because this was a master thesis the tool based approach was not an option. If it would be done as part of regular operations at a company the option should be considered before starting to develop an in house solution. Tool based migrations can potentially save a lot of the employee’s time that makes it more cost effective than designing a custom solution.

9.3 Cooperation troubles

This thesis started as a two man thesis. During the report writing stage the cooperation between us failed, we had totally different views on how the report should be done. This has led to delays in the writing of the report.

The problem was amplified more when we both started to work fulltime and even in different countries. This has resulted in the decision that we would write our separate reports on the same thesis work. The conclusion that can be drawn from this is that it is not always easier to be more people in a thesis or a project.
10 Future work

This chapter describes topics that were outside the scope of the thesis but that could be explored in the future. Because the thesis only focused on the process of developing a migration solution there is a lot of things that could be done as future work.

10.1 More complete testing

The scope of the thesis was to only test the migration with basic test data and simple unit tests. While this detects a lot of errors that can be made, more complicated issues can be missed. By using a system testing the application can be assured to work as intended with the migrated data. Acceptance testing could further decrease the issues encountered by end users when the target system is taken into production. To not test a migration thoroughly is to ask for problems when the target system is bought into production.

10.2 Cutover and fallback strategies

The shift from a prototype migration solution into an actual migration was not part of the scope. This could be examined in a future study, were cutover strategies for how the target system should be taken into production would be an important topic. Is it for example possible to run the systems in parallel for a period or is big bang migration the only way.

The possibilities of a fallback strategy could also be investigated. In which the actions needed to rescue a failing migration and how to reduce the damage to the production environment would be the main topic.
11 References

2. Codd, E.F. A Relational Model of Data for Large Shared Data Banks. 1970
5. AM Keller, R Jensen, S Agarwal - ACM SIGMOD Record. 1993
7. F Matthes, C Schulz. Towards an integrated data migration process model State of the art & literature overview. 2011
Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/