Western Norway University of Applied Sciences

Faculty of engineering and science

Department of Computer Science, Electrical Engineering and Mathematical Sciences

BACHELOR THESIS

Enterprise Architecture modelling for Bergen Municipality

Jose Juan Peña Gomez

Cato Støle Robstad

Dani Myking

Information technology/Computer Science
Department of Computer science, Electrical engineer and mathematical science
Rogardt Heldal/Richard Kjepso
3/4-2020

Western Norway University of Applied Sciences

Faculty of engineering and science

Department of Computer Science, Electrical Engineering and Mathematical Sciences

TITTELSIDE FOR HOVEDPROSJEKT

Rapportens tittel: Virksomhetsarkitektur modellereing for Bergen Kommune Enterprise Architecture modelling for Bergen Municipality	*Dato:* *01.06.2020*
Forfatter(e): Cato Støle Robstad, Jose Juan Peña Gomez, Dani Myking	*Antall sider u/vedlegg: 31* *Antall sider vedlegg: 2*
Studieretning: Informasjonsteknologi/dataingeniør	*Antall disketter/CD-er:* 0
Kontaktperson ved studieretning: Rogardt Heldal/Richard Kjepso/Carsten Gunnar Helgesen	*Gradering:* Ingen
Merknader:	

Oppdragsgiver: Bergen kommune	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Tom Osnes Svellingen	*Telefon:* 97737588

Sammendrag:

I bacheloroppgaven har gruppen utarbeidet et system for å automatisk hente ut data fra kildefiler, transformere dataen, og gjøre den klar for å lage en virksomhetsarkitektur modell. Transformasjonsprosessen blir gjort ved hjelp av scripts og databaser, og den endelige ferdige dataen lagres i en GitHub repository, som kan brukes til å oppdatere virksomhetsarkitektur-programvaren for Bergen Kommune, Archi.

In this bachelor thesis the group has created a system to automatically collect data from source files, transform the data, and make it ready to create a Enterprise architecture modell. The transformation process is done by various scripts and databases, and the final datamodel is saved in a GitHub repository, where it can be used to update the enterprise architecture software Bergen Municipality uses, Archi.

Stikkord:

r Enterprise architecture	Transformation	Database

Western Norway University of Applied Sciences

Faculty of engineering and science

Department of Computer Science, Electrical Engineering and Mathematical Sciences

Høgskulen på Vestlandet, Fakultet for ingeniør- og naturvitskap
Postadresse: Postboks 7030, 5020 BERGEN Besøksadresse: Inndalsveien 28, Bergen
Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hvl.no Hjemmeside: http://www.hvl.no

1.1

1.2 PREFACE

Thanks to Bergen Municipality for providing us with this interesting project, as well as providing support and guidance for us to be able to solve it.

Thanks to our supervisor Rogart for providing us guidance and advice on how to approach the solution.

TABLE OF CONTENT

1 INTRODUCTION

1.1 Motivation and goal

The task given from Bergen municipality was designing and implementing a solution for Modelling the Enterprise Architecture for the municipality. Bergen municipality uses the Enterprise Architecture for simplifying the complexity of both the business and information technology side of the organization. In consequence they are able to identify the short- and long-term goals and make the grounds for planning on how to achieve these goals.

In the previous system, the enterprise architecture model, a model of the whole organization from the underlying infrastructure to the top level business aspects, would have to be done and updated manually every time that there was a change in the source systems. This manual process is a long process and it might end up producing mistakes in the transformation of the data. Additionally, because of the manual process being done by one person, this system doesn't allow concurrent users to work with the same model.

The primary goal of this project is to have a fully automated process to be able to phase out the previous method of solving the issue which is done manually.

1.2 Context

Bergen municipality is a large enterprise, with over 30 000 employees and delivers services to its over 280 000 inhabitants. Bergen municipality uses Archi to model the enterprise, including organizing, processes, ICT-systems, and data. These enterprise architecture models are built up from several files from source systems.

In order to ensure precision and credibility in their models of Bergen municipality it's important to have a modern, reliable and trustworthy way of updating this information. That is why, an automated way to retrieve, transform and push new data to the model is needed. This will also save a lot of time and effort for Bergen Municipality, and would greatly help them have a constant and consistent model in Archi reflecting their real-life resources, capabilities and restrictions. This way they know they can trust and rely on their model.

1.3 Limitations

The main limitation for the given project is time. Due to this, main functionalities are prioritized. Given more time, the bachelor group would be able to provide additional functionalities such as statistics for changed data or a more extensive and intuitive user interface developed as a web application.

Another limiting factor was the situation caused by COVID19 restricting the groups ability to meet with each other, but also to meet with both Bergen Municipality and the project supervisor in person. Due to this, all meetings have to be done over the Internet, either by Discord, Zoom or Microsoft Teams. Due to this the team have set regular schedules to meet both with our project manager and within the team. This way the team will continue to operate as normal, with a regular schedule.

1.4 Resources

To complete the given project from Bergen municipality, one of the most important resources has been Bergen municipality's section for digitalization and innovation. Their knowledge about the enterprise architecture and modelling was important for us to learn quickly and they gave us good follow up concerning any questions the etam had regarding any topic. Another important asset has been our supervisor, Rogardt Heldal. Rogardt decided to join every meeting with Bergen Municipality, and has been of huge help regarding the development of the project.

1.5 Organization of the report

The report is organized in the following manner:

Chapter 1: An introduction to the project.

Chapter 2: A more detailed description of the project and project owner.

Chapter 3: The design of the project and discussing different approaches and planning.

Chapter 4: Design and creation

Chapter 5: Evaluation

Chapter 6: Discussion

Chapter 7: Conclusions and further work

Chapter 8: Literature and references

Chapter 9: Appendices.

2 PROJECT DESCRIPTION

2.1 Practical background

2.1.1 Project owner

Bergen Municipality is the owner of this project, and as stated earlier consists of over 30.000 employees that deliver ICT services to over 280.000 inhabitants. Modelling the enterprise Bergen municipality with over 30.000 employees and 800 departments can be a challenging and complex problem. In order to simplify the process, our task to automate the process of the modelling would greatly enhance their overview of their capacity, vulnerabilities and provide most recent information about systems and sections. This way the model of the enterprise architecture can be used in a more advanced way than it is currently used for.

As an example, this set up for further development and expansion of the architectural enterprise such as getting extensive information about underlying infrastructure. They could get easy access to information about e.g. servers with older versions of software prone to incoming attacks.

2.1.2 Previous work

Enterprise Architecture and ArchiMate

Enterprise Architecture tries to simplify the complexity of both the business and information technology side of an enterprise. There are several definitions of EA, but in this report the team will focus on the definition “Enterprise Architecture: a coherent whole of principles, methods, and models that are used in the design and realisation of an enterprise’s organisational structure, business processes, information systems, and infrastructure. “ (2006 Jonkers, p. 3).

By this definition it is easy to see that Enterprise Architecture is a large part of any large business and is an effective tool if used right to improve the enterprise as a whole. It is used to identify the short- and long-term goals and make the grounds for planning on how to achieve these goals.

To model Enterprise Architecture ArchiMate is an open and independent modelling language and is a visual way of describing architectural objects with relations in models. It has become a standard for describing and representing Enterprise Architecture.

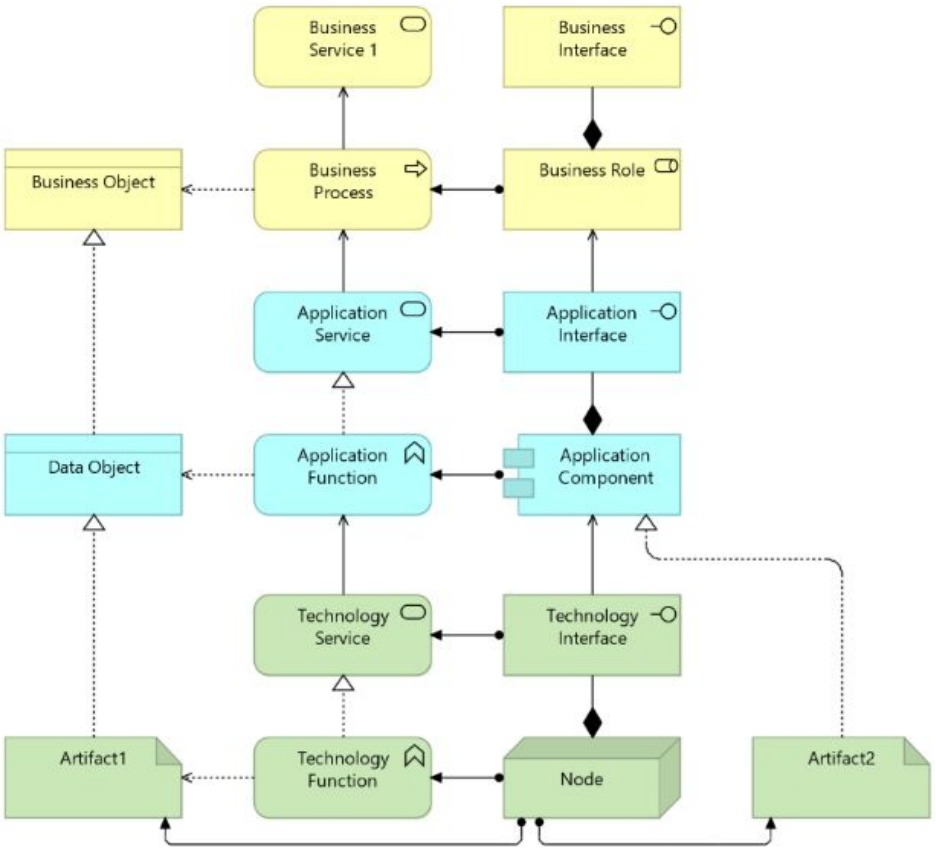

Figure 2.1: Main concepts of the ArchiMate language

Following the main concepts from figure 2.1 we can see ArchiMate has three different types of elements. Technology (green) elements, Application (blue) elements, and Business(yellow) elements. This way the whole enterprise can be described, all the way from the underlying infrastructure such as computers, servers and other technologies to what application they are running, what version of software, up until the top level, where the business services are provided.

Archi

Archi is a free and open-source tool used for visualizing and designing models based on the ArchiMate language, and is widely used within the enterprise architecture environment. Because it's built upon the ArchiMate language, Archi uses similar symbols and relations as ArchiMate to simplify the process of the visual representation of a model.

When creating models for Archi in another type of document, Archi follows strict rules to be able to read the document. There are three main categories of objects. Elements, relations, and properties.

Elements are used as the name of an object, where one element is an object. This element can then have multiple relations and properties.

The relations are other target elements related to a source element. In the ArchiMate language there are many different, where the most common relationships are

- Association: Most basic relation. Only states there is a relation between the elements, not what it consists of.
- Serving: Relation indicates one element performs a task included in another elements function
- Realization: Relation shows element A realize element B
- Assignment: Element A have ownership or responsibility of element B
- Access: Element A reads and/or writes to element B.

Properties are used as a description or information about a specific element.

2.1.3 Initial requirements specification

The initial request from the project owner was to deliver a proposed solution with the following requirements:

- Establishing a data structure for data storage
- Pull data from multiple source systems to be saved in the data structure
- Regularly updating of data when they are changed in the source systems
- Export data from the data structure to Archi
- Update Archi when the data is changed
- Synchronized a GitHub repository with the Archi model
- Automatization and scheduling of the process

Other than these requirements the software has to run in their Microsoft Azure data lake. These were only the requirements, how the team wanted to solve this issue was up to us to determine and implement it.

2.1.4 Initial solution idea

The initial solution design was to make an application to pull the files from the source systems, and then transform the raw data into a formatted table storage for viewing, still in an excel file. This would be used if the data would be shown outside of the model-form in Archi. After the data was formatted for table storage, the data would be transformed again into a SQL database, where it would be uploaded to a GitHub repository. Finally, Archi uses an extension to be able to import GitHub repositories and also to push the modification of a model to GitHub.

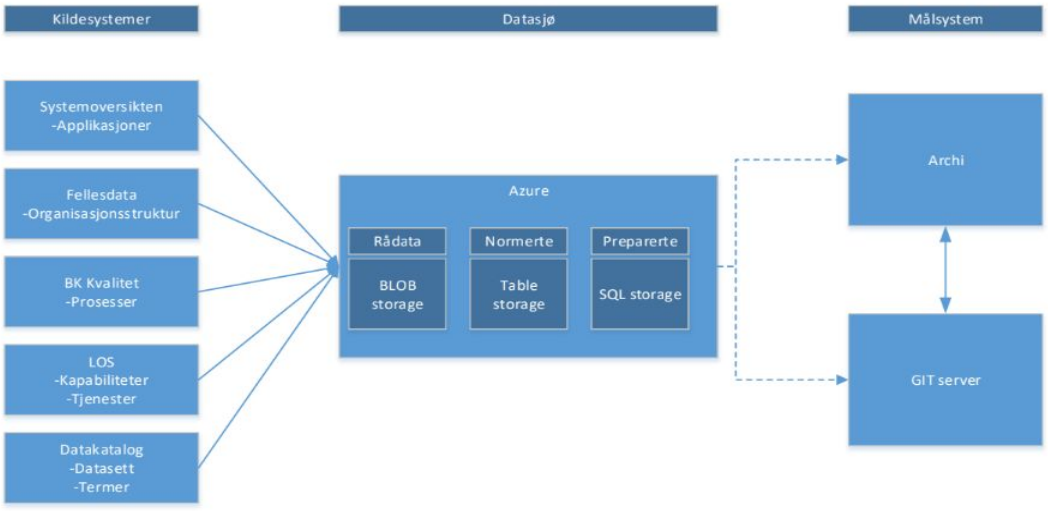

Figure 2.1.4.1: First proposed solution design of the project (Norwegian)

2.2 Literature background

During the start of the project, our team lacked knowledge about many aspects of what the project required such as basic structure and information about EA, Archi and Archimate, and a more technical side of the transformation and conversion. This required us to thoroughly research and learn these details for us to be able to create a system that would be optimal both for our project owner, and for us to be satisfied with the quality of the product.

A good introduction to EA, as well as how and why to use it effectively has been Jonker's one, 'Enterprise architecture: Management tool and blueprint for the organization'. This has shown us the importance of having a proper enterprise architecture model and plan for all organizations, as it can heavily improve both the business and technological sides of an organization.

Another source for us has been Holm's thesis on 'Automated data collection for Enterprise Architecture Models', however the main focus points are not completely relevant to our work. This research focuses on an automated network scan in order to identify all nodes in a network, to then be able to have all the data needed for an technological aspect of an enterprise architecture model. Although the main focus point does not align with our project, there are still similarities that can be drawn from their research and to our project with Bergen Municipality.

3 PROJECT DESIGN

3.1 Possible approaches

Bergen Municipality had no concrete way of approach when it came to implementation the solution. The initial meetings the team had with the Enterprise Architects was more based upon the rules and the requirements for the solution, therefore the team stood freely to implement the product as they wanted to given the requirements was taken into consideration. Other than the requirement list, the solution had to run in a Microsoft Azure environment.

3.1.1 Alternative approach 1

The first solution discussed in our team was only using Azure Fabrics with programming languages such as Java and scripts to transform the raw data to the accepted Archi format. This would include creating several scripts for handling the different source files, before all the finished data would be pushed to a GitHub repository and used with an extension to Archi, to update the Archi model.

After that, a graphical user interface would be implemented to update the Archi model, or the GitHub repository manually, or have it scheduled to a timer.

3.1.2 Alternative approach 2

A second solution that the team drafted was using mainly databases and scripts to transform the data from the source files. This would consists of a database for each enterprise architecture model, each handling models of the data at a different stage, such as one stage where the data was in the raw format from the source files, another for a normalised model of the data, and a last one associated with the model for the enterprise architecture model for Archi tool. This would, like the approach 1, also include a GitHub repository that pulls the finished model from the database. The Archi model would then be able to import the model using the same extension mentioned in approach 1.

3.1.3 Discussion of alternative approaches.

After discussing the different possible approaches alternative 2 have been selected for implementing. This seemed like an easier approach, as well as giving Bergen Municipality a database with prepared data for alternative tasks or services instead of just a model for the EA. The logic of the system is going to be the main focus and the biggest challenge, so as the team discussed with the project manager from Bergen municipality a conclusion was reached that the User Interface is going to be a secondary focus.

3.2 Specification

Our project is established from the initial solution Bergen Municipality used for enterprise architecture modelling. This original solution uses Excel to transform the raw data from the source files into a new Excel spreadsheet, which is converted in CSV format for being imported into the database. Excel transforms the spreadsheets, and after the transformations are done, the final transformed data for the enterprise architecture model is exported into a GitHub repository which can be imported into Archi to create the model for the enterprise architecture of Bergen Municipality.

With the new implementation the team will adopt the previous transformation rules into a more complicated system for this to be automated. The first step is to use scripts with the source files as input to check the format, and convert them for the correct format for the database. Once it is imported into the database, the action of importing it displays an event that activates the triggers in charge of the transformation of the data into the different models, generating two layers of transformations. This way of transforming the data makes the maintenance of the changes easier, by just updating the first model that completely depends on the source file.

The database and GitHub repository synchronization system makes sure that the day to day users of the enterprise architecture model have the opportunity to make local changes that are saved to the database. This also makes sure that everyone of the model has the correct version and everything is up to date.

For the deployment and maintenance of the system, a simple UI has been made for the people in charge of maintaining the databases and data lake system, but with the implementation of the database and GitHub synchronization this interface will not need to be used other than actual maintenance or deployment.

3.3 Selection of tools and programming languages

Python - One of the biggest object oriented programming languages, and one of the most used for scripting.

MySQL - The largest database management system (DBMS) in terms of market share. As the project is limited to Microsoft Azure Services, the decision had to be a DBMS that is compatible with Azure and also widely used elsewhere.

GitHub - Worlds leading code repository website for making storing database, Archi ready Excel files. Has integration to Archi.

Archi - Modelling tool for the Enterprise Architecture language ArchiMate widely used within the Enterprise Architecture environment.

Microsoft Azure - A cloud computing platform developed by Microsoft.

Shell Script (Bash) - Program designed to be run by the Unix shell, a command-line interpreter.

SQL - "Structured Query Language". A query language used for databases, to formulate and run instructions used for relational database systems.

Cron - Feature of Linux systems where it let's the user schedule tasks like execute scripts to specific given times

3.4 Project development method

3.4.1 Development method

For the development of the project itself the team has used an agile development method called Scrum. Scrum is built for teams varying in size of 3-9 people, where the team members work in "Sprints". A sprint is a given time interval, usually 1-2 weeks, where a set of smaller tasks must be completed, which later can be put together to complete the project.

The team has set the sprint time periods to 2 weeks per sprint, this way each member is able to complete several tasks. In order to keep everyone up to date on the progress daily sprint "standups" via discord has been arranged, and at the end of a sprint the team calls in for a meeting to do a sprint review to talk about how the newly completed sprint went.

3.4.2 Project Plan

In order to plan the progress and development of the project the team made a Gantt chart to follow during our sprints. Our Gantt chart works as an executive plan, while our sprint takes more into detail what to do, and when to do it. The team decided to use a web application named Trello to set up tasks for our sprints, to track who does what and at what

times. This makes it easy for everyone and ensures nobody works at the same tasks at any given time, unless intended to.

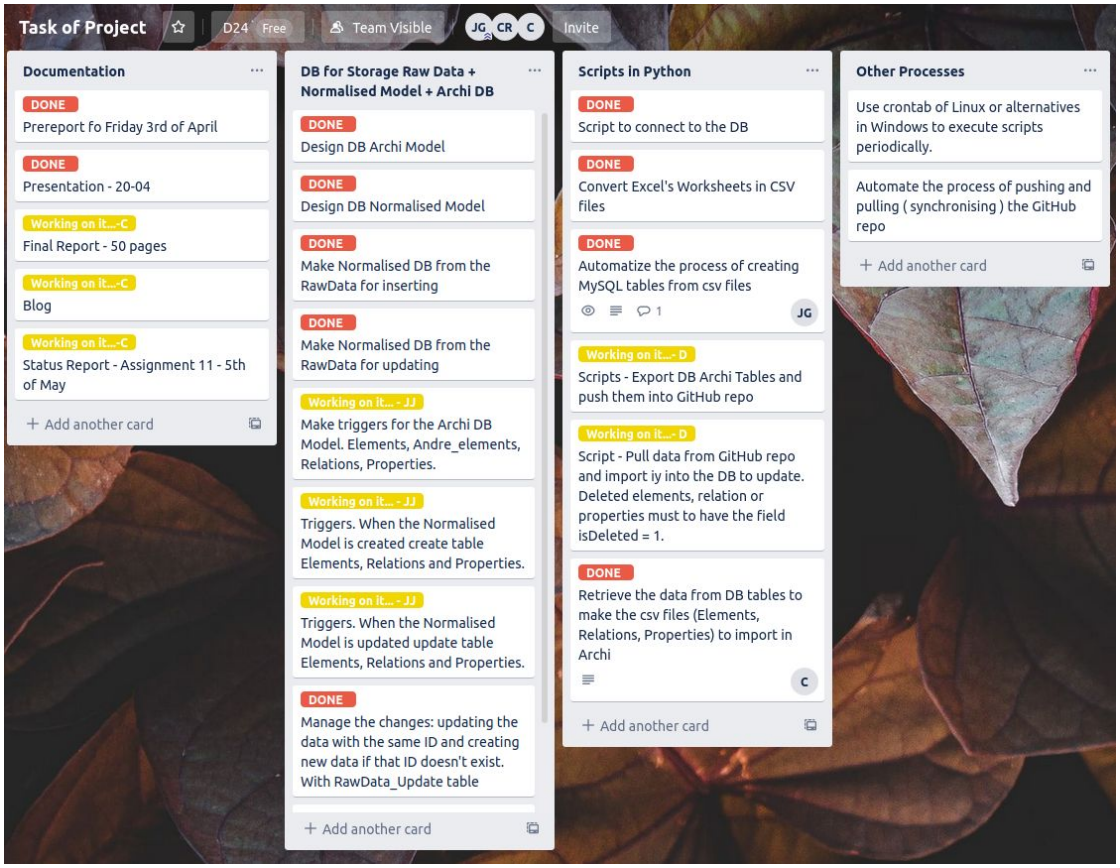

Figure 3.4: A screenshot of the Trello board used by our group for this project.

3.4.3 Risk management

There are a few risks to take into consideration when working on a large project like this. Because of this, the team decided early on in the planning phase to make a risk list to identify possible risks that could become a problem for the development of the project and possible measures the team can take in order to nullify or mitigate these risks (ref Appendix 9.1 Risk list). A risk regarding all projects at this time is of course the pandemic everyone is facing, COVID-19. Because of this, risks like misunderstanding requirements or miscommunication can prove vital to this project. This is something that has been specifically discussed and the team are working closely with Bergen Municipality to mitigate it.

3.5 Evaluation method

Constant evaluation is important in order to deliver high quality software. To ensure this, the team developed test databases in order to unit test that the developed python scripts are working as intended. For the database and transformation process, it is possible to quickly look up the correct model for Archi to see if the enterprise architecture model produced using our implemented artefacts are providing the same results as the old system for Bergen Municipality would.

After this testing, the team has to test also the integration for both the scripts and interface, to make sure everything works well together. In regards to testing the transformation logic there have also been scheduled regular meetings with Bergen Municipality. This way the team can ensure the quality and requirements are up to the standard they require, and ultimately deliver a high quality product all around that can be of use to them.

4. DESIGN AND CREATION

4.1 Updated requirements

During the development of the product and with regular meetings with Bergen Municipality the team have come to update the requirements for the solution. The updated requirement list includes several steps that makes the software better and more usable for Bergen Municipality, including tracking sources where the data comes from, rules for error handling, and tracking data created manually by architects.

The new and complete requirement list is:

- Establishing a data structure for data storage
- Pull data from multiple source systems to be saved in the data structure
- Regularly updating of data when they are changed in the source systems
- Export data from the data structure to Archi
- Update Archi when the data is changed
- Synchronized a GitHub repository with the Archi model
- Automatization and scheduling the process
- Warning when the data form the source systems changes the format
- Track the history of every event and activity that happens in the models
- Make a way of tracking the deleted, modified and created data and its respective dates.
- Track all the sources where the data comes from
- If there are conflicts between the source systems and the GitHub repository, prioritize the changes from the source systems.
- Track the data created by the architects in ArchiMate
- Make a manual maintenance of some part of the models.

This requirement list was made by having several meetings with the enterprise architects from Bergen Municipality as well as external resources and covers every need Bergen Municipality requires from a final solution currently.

4.2 Development

4.2.1 Initial phase and planning.

In the initial phase of the project the team were introduced to the current solution from Bergen Municipality and got a comprehensive overview of the limitations this solution brought with it. This was important for us to keep in mind, to create a solution that would greatly benefit Bergen Municipality when working with enterprise architecture in the future.

After the initial phase, having multiple meetings and workshops with the enterprise architects from Bergen Municipality the team started planning the project thoroughly with the idea "a well-planned project is key to develop the best possible solution". This led to an early created a Gantt chart with deadlines for important artefacts and worked accordingly to stay well ahead of schedule.

For the planning of the solution an Input-process-output diagram (IPOD) has been made to visualize the implementation and stages of our suggested final product.

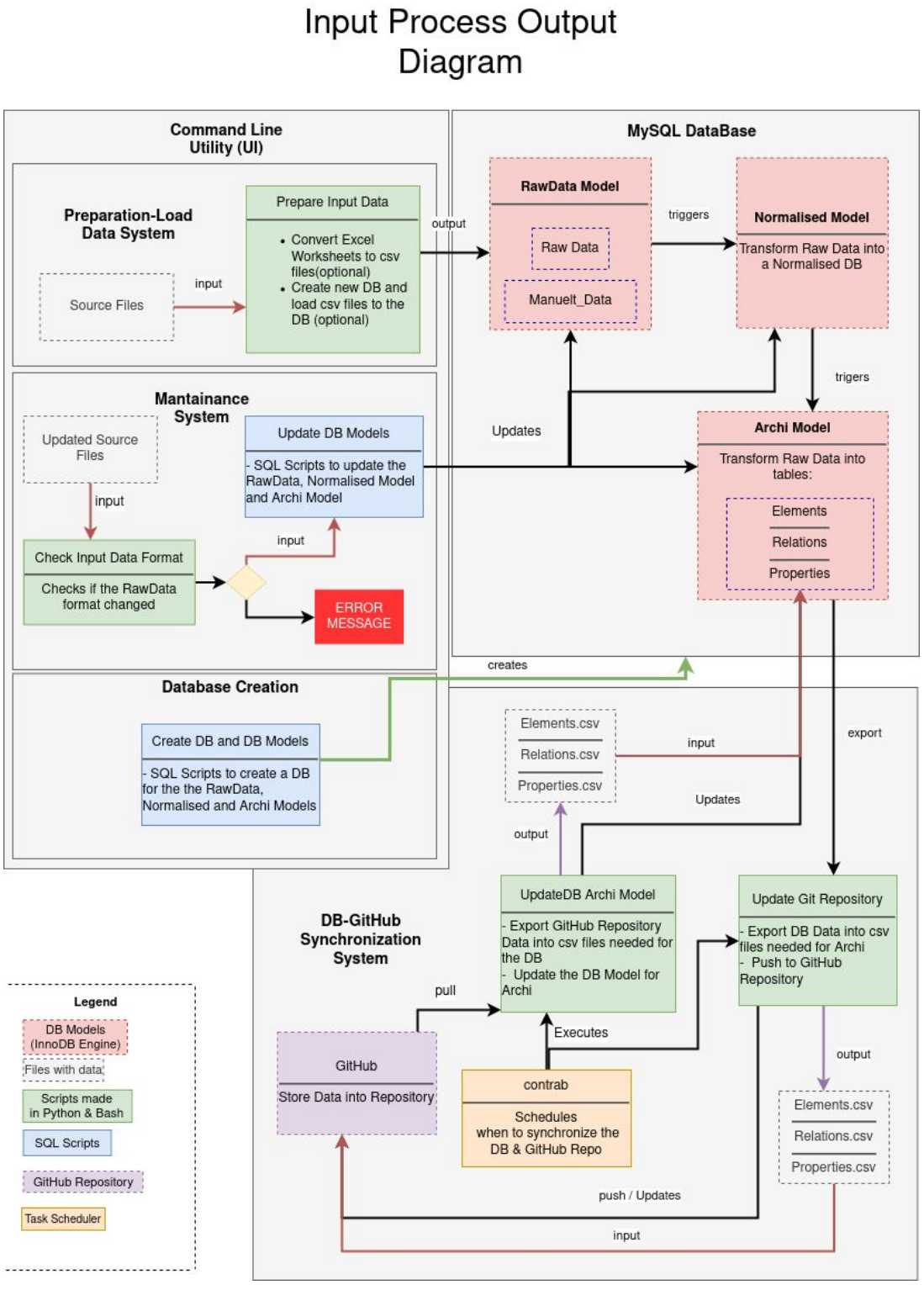

Figure 4.2.1 IPO-model of the product

4.2.2 Deployment-Maintenance Interface

Since the primary focus of the task has been on the logic and the transformation of the data, there has not been a large focus on the user interface. In order to execute the scripts, for creating and updating the models, the team have implemented a command line utility for the Azure data lake team in Bergen Municipality.

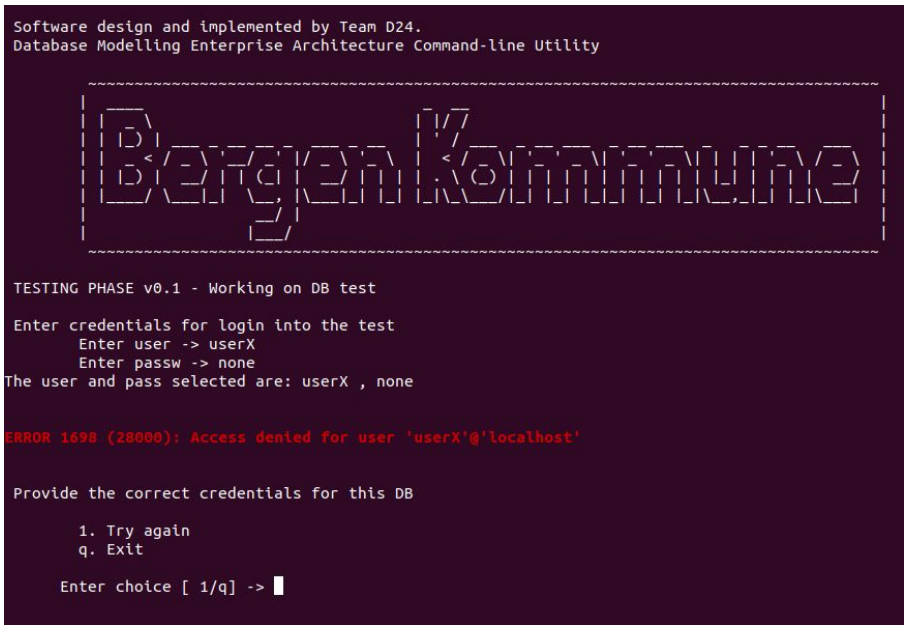

Figure 4.2.2.1: Command line utility. Log in to the command line utility

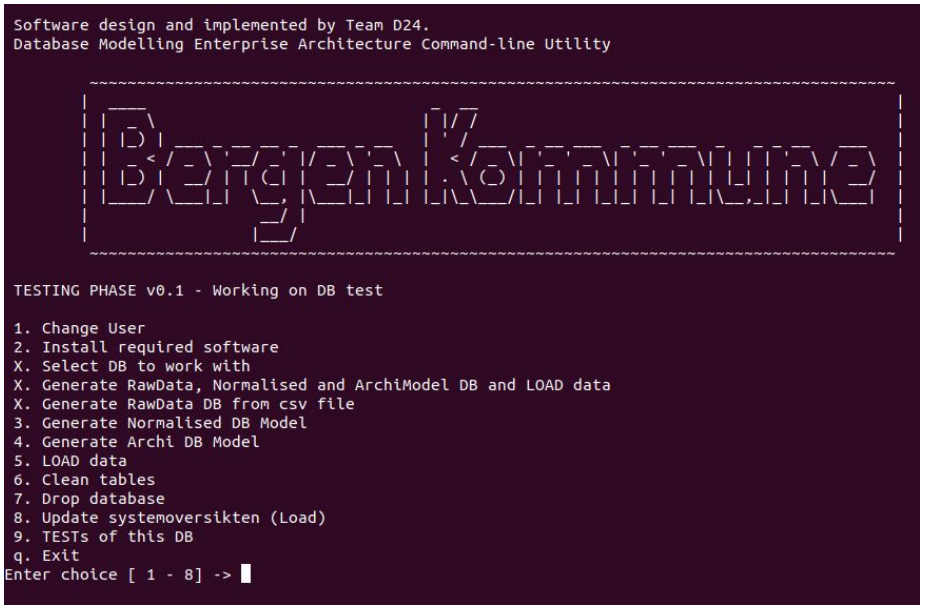

Figure 4.2.2.2: Command line utility for accessing scripts and databases

The main use of the interface is for maintaining and creating the database. The actual users of the information from the models (Enterprise Architects) will not need the interface since the complete enterprise architecture model can be pulled from GitHub and imported into Archi. If there are some changes they want to make locally, they can make these changes directly in the Archi model, and the rest of the model will be updated at a scheduled database update, from the local model.

When looking closer at the IPO model, we can see the deployment-maintenance utility line has three main uses. “Preparation-Load data system”, “Maintenance system” and “Database creation”. The first one used is “Database creation” as this creates all the three models of the database (to be explained later). This is simply a SQL script to create the databases, to initiate the system and have a place for storage of data.

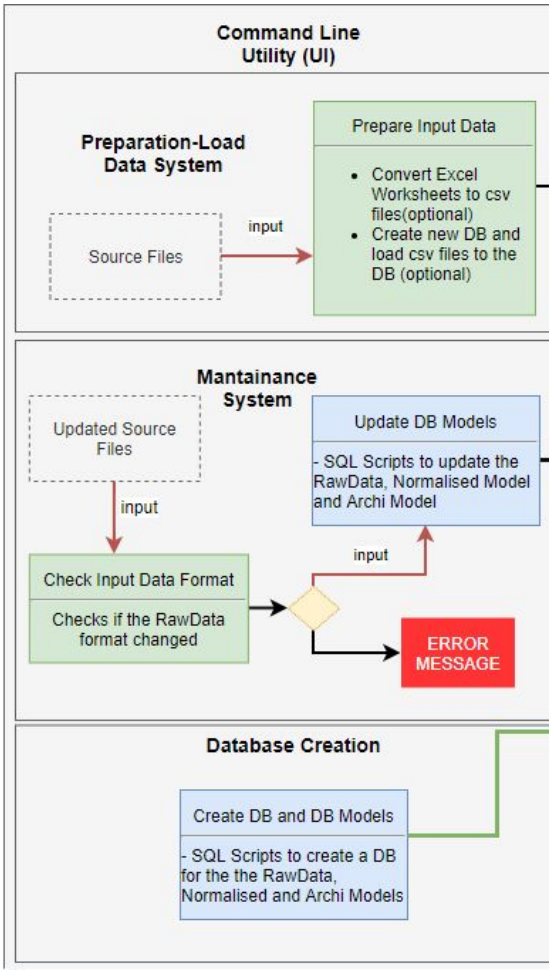

Figure 4.2.2.3: Command line utility part of the IPO-model.

The “Preparation-Load Data System” can be used after the creation of the new database. As seen in the model, this uses a python script with the source files as an input and prepares the data to be put in a database. Here, it is also possible to convert files from .xls to CSV files, which are the only accepted format for Archi. After this is done, all the data from the source files gets put in the database, ready for use there.

“Maintenance System” is used for updating the models from the source files with updated data. This way, there is no need for deleting the whole database and GitHub model, every time there is a change in the source files. The updated source files are checked by a python script, in order to validate if it has the correct format or not. If it does not have the correct format, an error message will display to the user the saying “format of the current source files does not match the format of the previous source file”. If it does match, another script is called using the source files as input, and updates the three database models with this data by inserting new data into them or updating the data that was previously stored.

4.2.3 Database

After creating the database, it will have three different models. “Raw data Model”, “Normalized Model”, and “Archi Model”. These three models serve their own purpose in the database as a place for storing current state of representational data. When the Raw data model is updated using the scripts from the command line utility, a trigger on insertion is used to automatically update the normalized model and likewise from the normalized model to the Archi model.

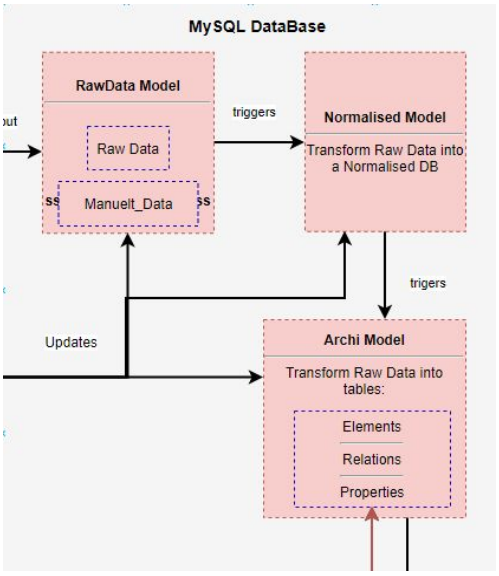

Figure 4.2.3.1: Database models, from the IPO-model

Raw data model

The raw data model is the first model the data from the source files reaches. In this database model, they are formatted just like they were in the source files only they are in the first layer, or model, of the database.

```
CREATE TABLE IF NOT EXISTS RawData (
       systemtype varchar(255),
       system_id int(11),
       navn varchar(255),
       beskrivelse text,
       systemeier varchar(255),
       systemkoordinator varchar(255),
       admsone int,
       sikker_sone int,
       elevnett int,
       tu_nett int,
       internettviktighet varchar(255),
       personopplysninger varchar(255),
       sensitive_personopplysninger varchar(255),
       createdDate DATETIME DEFAULT LOCALTIME,
       lastModified DATETIME DEFAULT LOCALTIME,
       isDeleted INT(1) DEFAULT 0,
       source INT(11) DEFAULT 1,
       PRIMARY KEY (systemtype,system_id),
       FOREIGN KEY ( source ) REFERENCES source ( srcId )
) DEFAULT CHARSET=utf8;
```

Code snippet 4.2.3.2 Database model for the Raw Data

Normalized data model

After all the data has been pulled into the raw data database, that is used to update a normalized model of the data. This is a requirement of the solution by Bergen Municipality to use the EA data elsewhere than just for the visual representation in Archi. This way, they have another clear and distinct way of accessing the data if it was to be used in another project, or needed in a normalized form that's not used for Archi.

After several meetings with the enterprise architects, on how to build the normalized model the team reached the conclusion that model in figure 4.2.3 was the most optimal way of representing the data provided. Figure 4.2.3 is a diagram of the source file "Systemoversikten" and shows how the transformation is taking place when moving the data from the raw format to the normalized format in the database.

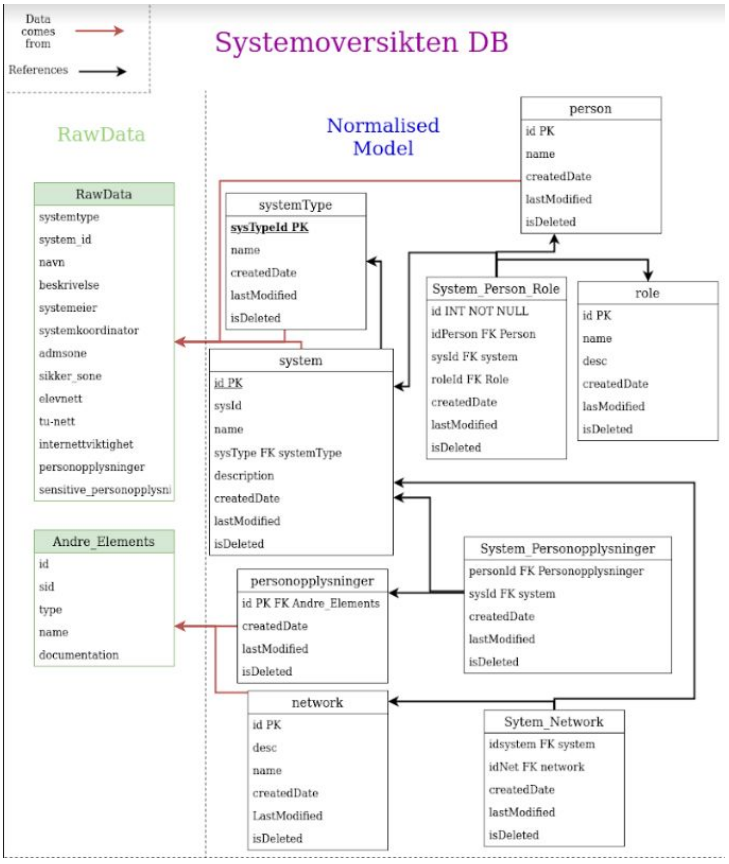

Figure 4.2.3.3 Raw data transformed to a normalized model from the source file "Systemoversikten"

Archi model

The final model of the database needed for our solution is the Archi model. In this model of the database, the data from the normalized database model is transformed to be used for a visual representation in Archi. Archi requires three types of entities for their objects, namely Elements, Relations, and Properties. This is created from the normalized data model when the data is transferred into the Archi model side of the database. After the Archi model database is created the appropriate accepted format for Archi is ready and can then be pushed to the GitHub repository.

Underneath the different tables for Elements, Relations and Properties, as well as triggers for updating these tables are illustrated.

```
CREATE TABLE Element (
        sysId INT UNIQUE DEFAULT NULL,
        ID VARCHAR(255),
        TYPE VARCHAR(255),
        NAME VARCHAR(255),
        DOCUMENTATION TEXT,
        createdDate DATETIME DEFAULT LOCALTIME,
        lastModified DATETIME DEFAULT LOCALTIME,
        isDeleted INT(1) DEFAULT 0,
        source INT(11) DEFAULT 1,
        FOREIGN KEY ( source ) REFERENCES source ( srcId ),
        FOREIGN KEY ( sysId ) REFERENCES system ( id ),
        PRIMARY KEY (ID)
);
```

Code snippet 4.2.3.4: Elements table, implemented in the Archi model database.

```
CREATE TABLE Relation (
       ID_R VARCHAR(255) PRIMARY KEY,
       TYPE VARCHAR(255),
       NAME VARCHAR(255),
       DOCUMENTATION TEXT,
       SOURCE VARCHAR(255),
       TARGET VARCHAR(255),
       createdDate DATETIME DEFAULT LOCALTIME,
       lastModified DATETIME DEFAULT LOCALTIME,
       isDeleted INT(1) DEFAULT 0,
       sourceModel INT(11) DEFAULT 1,
       FOREIGN KEY ( sourceModel ) REFERENCES source ( srcId ),
       Foreign Key ( TARGET ) References Element( ID )
)Engine="InnoDB";
```

Code snippet 4.2.3.5: Relation table implemented for the Archi model database.

```
CREATE TABLE Property (
       ID_P VARCHAR(255),
       KEY_P VARCHAR(255),
       VALUE_P VARCHAR(255),
       createdDate DATETIME DEFAULT LOCALTIME,
       lastModified DATETIME DEFAULT LOCALTIME,
       isDeleted INT(1) DEFAULT 0,
       source INT(11) DEFAULT 1,
    PRIMARY KEY (ID_P, KEY_P),
       FOREIGN KEY ( source ) REFERENCES source ( srcId )
);
```

Figure 4.2.3.6: Table for Properties in the Archi model database.

4.2.4 Database-Github synchronization system

After the data has been transformed and is stored in the Archi model database, it is exported using a python script for updating the GitHub repository. Then, the data is stored in the database and once again it is converted to CSV, and pushed to GitHub. The GitHub is used as a central base for the correct and updated information, because of the functionality from the Archi extension that allows for GitHub repositories to be imported directly. This way, the architects using this system only need access to Archi and the GitHub repository.

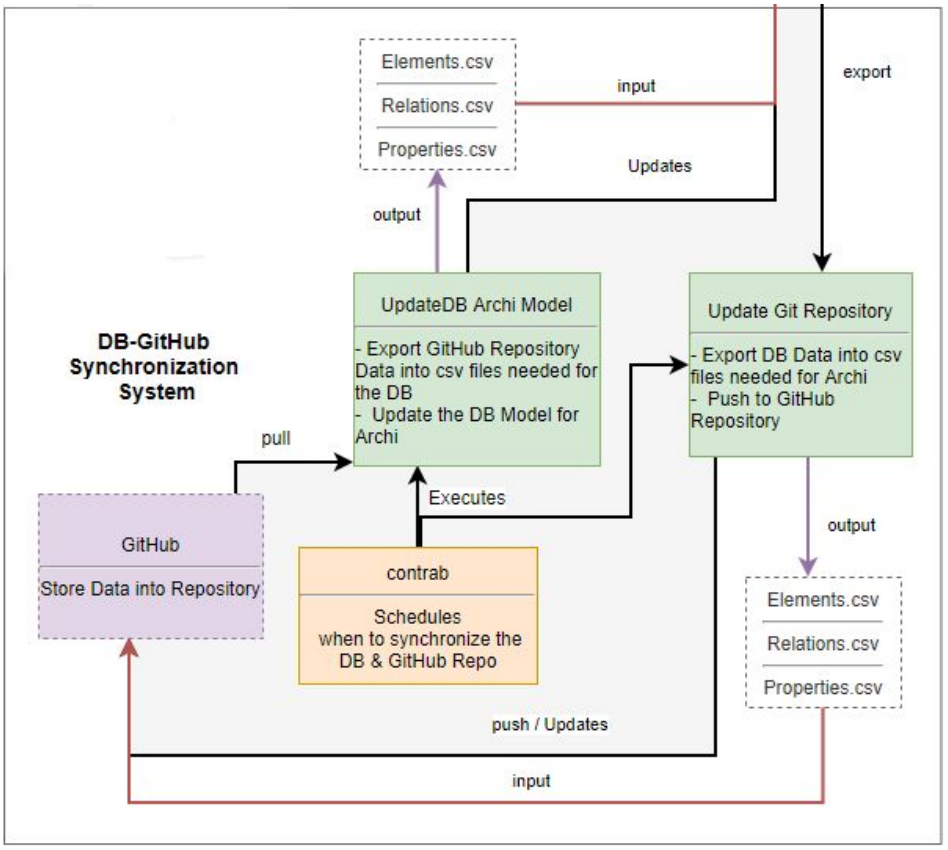

Figure 4.2.4.1: Database-GitHub synchronization part of the IPOD diagram

As previously mentioned, there might be any local changes needed to be done to the enterprise architecture model. Because of this, another functionality is needed. This is where the “UpdateDB Archi model script” comes in. This allows for users to make changes locally and push their version to GitHub. To ensure the database is up to date with the corrected model, the script is run using cron, a tool used for scheduling execution of scripts or events periodically for either set times, dates or intervals depending on needs. This ensures that ever so often (depending on Bergen Municipalities needs) the database for the Archi model is updated using the GitHub repository to reflect the changes locally made from users.

4.3 Transformation process

The process of transforming the data that are going to be represented in the enterprise architecture model have been a big part of this project. As mentioned, the transformation of the data coming in from the source files are happening in the different data models of the database. This is done by using triggers to update the two latter models (Normalized and Archi models), when getting new data into the raw model. Chapter 4.2 described how the normalized model was created from the raw data, but this chapter will mainly focus on how the transformation process from the normalized model to the Archi model is being made.

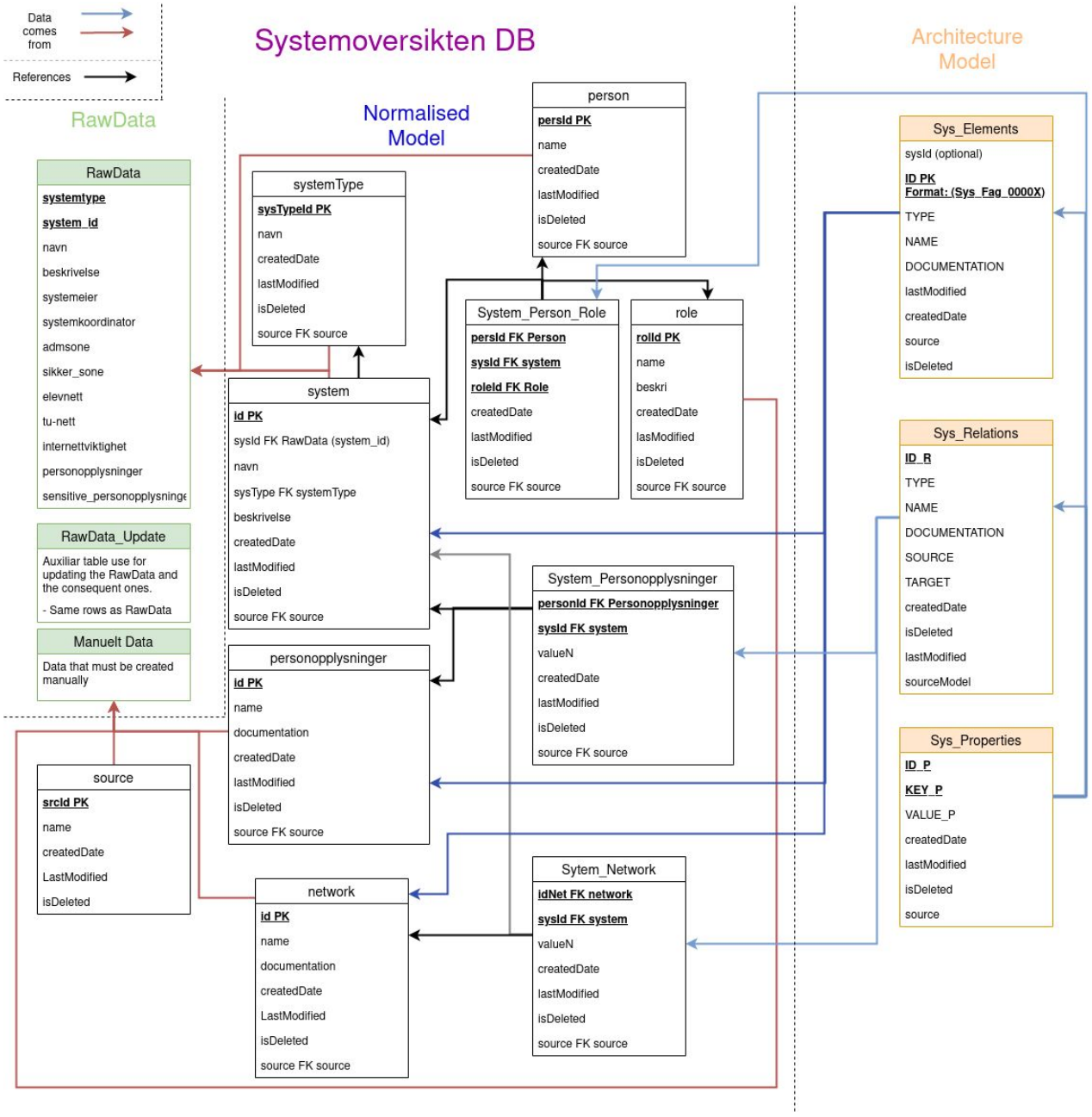

Figure 4.3.1: Model of transformation from source files to the Archi model in the database. Systemoversikten Data Model.

Figure 4.3.1 explains how the Elements, Relations and Properties are created from various tables in the normalized model, that again are created from the source files. The blue arrow-lines are the creation of Elements, Relations and Properties.

To properly update the Elements, Properties and Relations tables in the Archi model, the triggers make sure that every piece of information needed is added, so that the model is complete and compatible with the Archi import rules. Figure 4.3.2 shows one of the triggers used to update Element and Property tables (in Archi model) from the network table (in the normalized model)

```
DELIMITER //
CREATE TRIGGER trig_ElemFromNets_Ins AFTER INSERT
ON network FOR EACH ROW
BEGIN
SET @newID = CONCAT("ComNet",CONVERT(LPAD(NEW.id,4,0), CHAR));
INSERT INTO Element (ID, TYPE, NAME, DOCUMENTATION, createdDate, source)
VALUES(@newID, 'CommunicationNetwork', NEW.name, NEW.documentation, LOCALTIME, NEW.source);
INSERT INTO Property (ID_P, KEY_P, VALUE_P, createdDate, isDeleted, source)
VALUES (@newID, 'Kilde', 'Manuelt nettverk', LOCALTIME, NEW.isDeleted, NEW.source);
INSERT INTO Property (ID_P, KEY_P, VALUE_P, createdDate, isDeleted, source)
VALUES (@newID, 'Opprettet', LOCALTIME, LOCALTIME, NEW.isDeleted, NEW.source);
INSERT INTO Property (ID_P, KEY_P, VALUE_P, createdDate, isDeleted, source)
VALUES (@newID, 'Sistoppdatert', LOCALTIME, LOCALTIME, NEW.isDeleted, NEW.source);
       END;//
DELIMITER ;
```

Code snippet 4.3.2: Trigger used to update Element and Property tables.

To explain the whole process, a demonstration of the raw data from the source files transformation will be described in the following figures.

	A	B	C	
1	Systemtype	System ID	Navn	Beskrivelse
2	Fagsystem	253	24 timers BT apparat	24 timers BT apparatBlodtrykksapparat
3	Fagsystem	414	ABAX	
4	Fagsystem	290	Adgangskontrollsystem for E	Adgangskontrollsystem (Starwatch) for Rådhuset og omliggende bygninger. Leveres og
5	Fagsystem	376	Adobe Sign	
6	Fagsystem	1	Adra Match	Adra Match er et system for automatiske kontoavstemminger og for automatisk oppdate
7	Fagsystem	260	Agresso HR, kursmiljø	Kursmiljø til Agresso HR milestone 5Agresso HR er Bergen kommune sitt lønns- og HR-

Figure 4.3.3: Columns of raw format source file.

The format of figure 4.3.3 is the standard format the source files are presented as when nothing has been done to them yet. We will look closer at the “Fagsystem” with the name “Adra Match” and System ID “1”.

```
MariaDB [test]> select * from RawData;
+--------------+-----------+-------------------------------------------------------+-----------------------------------
| systemtype   | system_id | navn                                                  | beskrivelse
+--------------+-----------+-------------------------------------------------------+-----------------------------------
| Fagsystem    |         1 | Adra Match                                            | Adra Match er et system for automatiske
```

Figure 4.3.4: “Fagsystem Adra Match” in the rawData table. First part of transformation.

After having updated the rawData table, the normalized model will be updated with the implemented trigger. The “Adra Match” is a system, and will be inserted in the “System” table by the the trigger previously shown (Ref figure 4.3.1 for the whole model).

```
ariaDB [test]> select * from system;
----+-------+-------------------------------------------------------+---------+-------------------------
 id | sysId | navn                                                  | sysType | beskrivelse
----+-------+-------------------------------------------------------+---------+-------------------------
  1 |     1 | Adra Match                                            |       1 | Adra Match er et system fo
```

Figure 4.3.5: “Fagsystem Adra Match” in the system table, as part of the normalized model and second part of the transformation.

When the normalized model is updated, this again activates the triggers for the Archi model to be created. In this table “Adra Match" will be stored as an Element, with the additional information such as “beskrivelse”(description) and “sysType”(system type) will be created as properties for “Adra Match”.

```
MariaDB [test]> select ID, TYPE, NAME, createdDAte, isDeleted, source from Element;
+--------------+--------------------+-------------------------------------------------------+---------------------
| ID           | TYPE               | NAME                                                  | createdDAte
+--------------+--------------------+-------------------------------------------------------+---------------------
| SYS_Fag_00001 | ApplicationComponent | Adra Match                                            | 2020-05-17 14:30:54
```

Figure 4.3.6: “Fagsystem Adra Match” in the Elements table, the Archi model database. Third part of the transformation”

The “Fagsystem Adra Match” also has different properties described in additional columns, and all can be found in the Property table.

```
MariaDB [test]> select ID_P, KEY_P,VALUE_P from Property where ID_P="Sys_Fag_00001";
+---------------+-------------------+----------------------+
| ID_P          | KEY_P             | VALUE_P              |
+---------------+-------------------+----------------------+
| SYS_Fag_00001 | Kilde             | Systemoversikten     |
| SYS_Fag_00001 | Opprettet         | 2020-05-21 13:26:04  |
| SYS_Fag_00001 | Sistoppdatert     | 2020-05-21 13:26:04  |
| SYS_Fag_00001 | systemeier        | Mads Hagebø          |
| SYS_Fag_00001 | systemkoordinator | Bjørn Eivind Berge   |
| SYS_Fag_00001 | Systemtype        | Fagsystem            |
+---------------+-------------------+----------------------+
```

Figure 4.3.7: Additional properties for “Fagsystem Ardra Match”

RawData_Update

Another requirement from Bergen Municipality needed for the system was that no data should be deleted. This ment on the updated source files, the team would need a way to compare the previous, rawData, and new, rawData_Update, to check if the previous source files had any data that the new source files did not. This was implemented using a script to compare each row of the sources, and if any of the data was taken out from the new source files it would set the value of a column called isDeleted to “1” .This way the architects know the system or piece of information they are looking at in the finished model, might not be supported or are out of the system. In addition, if there is any new data in the new source file, this one will be inserted into the rawData table and will display the process explained previously.

4.4 Design Science

The methodology for our project has been design science. Denning 1997; Tsichritzis 1998 have described Design Science (1997, 1998, Cited in Hevner 2004 p. 76) as 'a problem-solving paradigm. It seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished.' This falls in line with our project methodology as our developed artifacts serve a specific purpose in regards to solve important organizational or enterprise level business problems, and it is a research that tries so solve a concrete problem with innovation through creative implementation of information systems.

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Figure 4.4 Hevners 7 guidelines for design science in information systems research (2004 Hevner, p 83)

Throughout the project the team have kept the key points and the 7 guidelines of Design Science (2004 Hevner p. 83) as seen in figure 4.4 in mind for developing our product. This has made our team more aware of the fundamentally important stages of developing artefacts and also helped us better analyze and process the information needed to develop our solution.

5. Evaluations

5.1 Evaluation methods

To ensure quality of the product extensive testing has been very important. As previously stated the team started early implementing testing for developed artefacts. This consists of test-databases for our team to be able to test the various scripts and database implementations, and check the full cycle of getting the data from source files into the database, transformed correctly and pushed to the GitHub repository.

Because of the way our project is built up it is relatively easy to test if the implemented artefacts are working correctly when finished. With the given source files, the team is able to create another Archi model based on the previously used method for modelling from Bergen Municipality. By using this model, it is easy to check if the new implemented product fulfills the requirements and that the model created from the new solution is correct by comparing the two different models to each other. Additionally, our team has made their own test source files for testing the system with all the possible cases, being able to perform deeper and more extensive testing to be sure that there are no possible bugs or errors when running the system.

Some of our smaller artefacts such as scripts are created so solve one specific problem. Due to this, it is also very easy to test whether the script is working as intended or not. One example is a specific script for checking if a file is in the correct format (.CSV file). Testing this script would ultimately just take files with different formats as input and check if the script provided the correct answer by checking the format of the files by ourselves.

Another important part of our evaluation have been the regular meetings with Bergen Municipality. The team has had meetings every two weeks. During these times the team has had the opportunity to show our current state of the project and our plans ahead, and the enterprise architectures from Bergen Municipality have had the opportunity to come with specific feedback or comments regarding anything they wanted to change or correct if they found anything lacking or not clear. This process has greatly helped specifically in areas concerning the transformation process, where understanding how to correctly implement the formatting proved difficult.

5.2 Evaluation results

The constant evaluation has been very important to us, as the system the team has developed is a big part of the information about the organizational structure in Bergen Municipality.

The benefits of working on a project like this where it is possible to have the correct answer of the final product (EA-model) are very noticeable. This has immensely helped the evaluation process by reducing uncertainties whether the team has implemented our solution correctly or not. As a result, this has made us be able to focus on other aspects of the project that might require more attention.

The meetings with Bergen Municipality have also proved to be very efficient in regards to the evaluation. This has made the process of our implementation more effective in the sense that the team is getting direct feedback on both our current and planned solution, so that the team knows what works and what needs to be changed.

Another area of evaluation the team has applied to the project during the course of this work, is regular self evaluations using the scrum methodology. Scrum has multiple phases, and on various stages it has 4 major scrum meetings: Initial planning meeting, a daily sprint meeting, a sprint review meeting and a retrospect meeting after the project is finalized. During specifically the sprint review meetings the team have had opportunities to address our perspective of the finished sprint in regards to how it went, what went well and what could have been better for our next sprint. During one of the first sprint reviews the team revealed certain weaknesses with our initial internal meeting frequency and the in-effectiveness of only having daily sprint meetings in such an intertwined project like this. This way an adaptation was made early on in the planning phase of the project, and it was decided to have daily meeting-sessions for regular scheduled hours where everybody in the team would be available either for discussion regarding implementation, technical questions of the software, methodology or other difficulties members of the team might run into during the work. Both the sprint review and the daily scrum meetings have also played a part in the evaluation process as a way for every member of the team to coherently progress with the project in such a way that no time is wasted due to misunderstanding within the team, or in regards to our external resources.

6. DISCUSSION

Approach

The teams approach for managing and delivering a solution have been driven by a solution oriented point of view. At the start the team sought out to gain a comprehensive understanding for the appropriate areas needed in order to solve the project in the best manner possible, and have been steadily researching topics for either improving the already implemented artefacts or in order to further develop the project in the best manner possible.

Another important approach for our team has been to develop with a deliberate focus on accuracy and usability. Together with extensive testing, this allows for our team to be sure of the implemented features are working as intended, and there is no need to redo tasks that might take up a large portion of time.

Consequences of approach

Due to the approach the team have had, regarding the research and "implementing right the first time", this has slightly come as an expense of our planned schedule. In order to make up for this and still continue on track with the Gantt chart the team have had to extend the daily schedule for some parts of the project. This has been well worth it for the sake of the project, and because of this the team have had sufficient time if any complications of any sort should arise.

Improvement

One of the things the team were struggling most with during the course of the project has been the transformation and logic behind this. When the team realized the time spent on this was going out over the initial plan, meetings with Bergen Municipality was set up and additional external resources brought in to help solving the problem regarding the transformation logic in the most efficient way possible. Together with the project manager and other external resources the team managed to solve our difficulties quickly and were able to resume to our planned schedule. This problem could be prevented if the team identified the magnitude of the logic needed and were able to schedule meetings early when the problem was initially recognized.

7. CONCLUSIONS AND FURTHER WORK

Goals

The bachelor projects goals of creating an automatic way for Bergen Municipality to update their enterprise architecture model have been achieved. Bergen Municipality also had an additional wish that they addressed at a later stage of development, regarding another test case for the system, so the structure the team has made could have additional uses. The team would have liked to complete this task too, and discussed the possibility to complete this, but it would ultimately require a complete change of the current model, again requiring a lot of time. Due to this, our only possibilities was either extend the due date and implement this additional feature, or leave the due date and solution as it is.

Usability for other conditions

Our results can also be used for other conditions. In every municipality there's certainly a need for modeling the enterprise architecture, in order to be able to deliver their services to their inhabitants in the best possible manner. To do this, they need to know their capabilities, something that comes automatically with a good enterprise architecture model. This does not only apply to Municipalities in Norway either, but all organizations of a larger scale. What the team has provided for Bergen Municipality is an efficient way of getting the tools every organization needs in order to improve themselves.

Additional further work:

Other than the additional test case discussed earlier, there is still some work left, namely the deployment of our project into their systems. Due to COVID-19 the team has not been able to come into their offices, and most of the employees the team has interacted with are working from home. When the situation is over and Bergen Municipalities offices open up, the team will come into their office with our finished project to deploy it, and give them a thorough run down of how to use the system.

8. LITERATURE AND REFERENCES

Archi (n.d.) *About Archi* [online] Available at https://www.archimatetool.com/about/ [Accessed at 5. February 2020]

ArchiMate (n.d) *Welcome to ArchiMate® 3.1 Specification, a Standard of The Open Group* [online] Available at https://pubs.opengroup.org/architecture/archimate3-doc/ [Accessed at 14. April 2020]

Bergen kommune (n.d.) *Fakta om Bergen* [online] Available at https://www.bergen.kommune.no/omkommunen/fakta-om-bergen [accessed at 23. April 2020]

Code Blocks (n.d.) *Syntax highlighting for google docs* [online] Available at: https://gsuite.google.com/marketplace/app/code_blocks/100740430168 [accessed 1. June 2020]

Crontab (n.d) *Tables for driving cron* [online] Available at: "crontab(5): tables for driving cron - Linux man page" [Accessed at 3. April 2020]

Diagrams.net (n.d) *Diagram with anyone, anywhere* [online] Available at: https://www.diagrams.net/ [Accessed at 12. May 2020]

GitHub.com (n.d) *the worlds leading software development platform* [online] Available at: https://github.com/ [Accessed at 19. March 2020]

Google drive (n.d.) *Using drive* [online] Available at: https://www.google.com/intl/no_ALL/drive/using-drive/ [accessed at 19. May 2020]

Holm, H. (2012). Automatic Data Collection for Enterprise Architecture Models [online] available at: https://www.researchgate.net/publication/220198712_Enterprise_architecture_Management_tool_and_blueprint_for_the_organisation [Accessed 23. April 2020]

Jonker, H. (2006). Enterprise Architecture: Management tool and blueprint for the organisation [online] available at:
https://www.researchgate.net/publication/220198712_Enterprise_architecture_Management_tool_and_blueprint_for_the_organisation [Accessed 14. April 2020]

Hevner, A. R. (2004). Design Science in Information Systems [online] available at:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.1725&rep=rep1&type=pdf
[Accessed 12. April 2020]

MySQL (n.d.) *The worlds most popular open source database* [online] Available at:
https://dev.mysql.com/doc/ [accessed at 22. March 2020]

MySQL Triggers (n.d.) *Triggers* [online] Available at:
https://dev.mysql.com/doc/refman/8.0/en/triggers.html [Accessed at 13 April 2020]

Python (n.d.) *About Python* [online] Available at https://www.python.org/about/
[Accessed 12. March 2020]

Scrum.org (n.d.) *What is scrum?* [online] Available at:
https://www.scrum.org/resources/what-is-scrum [Accessed at 16. April 2020]

SQLAlchemy (n.d.) *Database toolkit for python [online]* Available at:
https://www.sqlalchemy.org/ [Accessed at 28. March 2020]

Trello.com (n.d.) *About trello* [online] Available at: https://trello.com/about
[Accessed at 25. March 2020]

Microsoft Azure (n.d.) *Cloud computing [online]* Available at:
https://azure.microsoft.com/en-us/ [Accessed at 28. March 2020]

9 APPENDIX

9.1 Risk list

Explanation of risk list:
P = Probability of risk happening.
C = Consequence of the risk.
RF= Risk factor, the probability times the consequence of a risk.
Values P and C ranges from 1-5 where 1 is the lowest probability and consequence and 5 is the highest.

Risks	P	C	RF	Measures taken to reduce risk factor
Illness in bachelor group	1	5	9	Good communication within the bachelor group to ensure everybody is updated, in case someone's workload must be relieved.
Misunderstanding of tasks/requirements in the bachelor project	2	5	10	Keep a good dialogue with the project owner and supervisor and have frequent meetings to keep every part updated on their view of the project. Good feedback
Low competence within areas that leads to a product of low quality	3	3	9	Learn new technologies needed. Feedback from the project owner and good testing, to be sure of implemented artifacts are of required standards. Also help from supervisor and external resources can ensure the team can keep a satisfactory standard of the product
Poor time management/unable to finish project	2	3	6	Start with the project quickly and work consistently. Develop a plan to follow from start to end of the project. This way it is easy to get an overview of where the project is relative to where it needs to be in the timeframe to be able to finish it.
Poor communication/difficulties within bachelor group	3	2	6	Have good communication and frequent meetings with feedback and comments on both work done, and

				work to be done.
Technical difficulties/hardware or software not working properly	2	2	4	Keep backup copies of developed artifacts in case of crashes. Keep software and drivers updated. Learn new software needed to efficiently troubleshoot if needed.

9.2 GANTT diagram