Tampere University

MIKKO PUONTI

Continuous Delivery in Data Warehousing

Tampere University Dissertations 706

Tampere University Dissertations 706

MIKKO PUONTI

Continuous Delivery in Data Warehousing

ACADEMIC DISSERTATION
To be presented, with the permission of
the Faculty of Information Technology and Communication Sciences
of Tampere University,
for public discussion in the auditorium RG202
of the Rakennustalo, Korkeakoulunkatu 5, Tampere,
on 9 December 2022, at 12 o'clock.

ACADEMIC DISSERTATION
Tampere University, Faculty of Information Technology and Communication Sciences
Finland

Responsible supervisor and Custos	Professor Hannu-Matti Järvinen Tampere University Finland	
Pre-examiners	Professor emeritus Bernhard Thalheim University of Kiel Germany	Professor Jérôme Darmont University of Lyon 2 France
Opponent	Professor Pekka Abrahamsson University of Jyväskylä Finland	

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

Cover design: Roihu Inc.

ISBN 978-952-03-2652-4 (print)
ISBN 978-952-03-2653-1 (pdf)
ISSN 2489-9860 (print)
ISSN 2490-0028 (pdf)
http://urn.fi/URN:ISBN:978-952-03-2653-1

Carbon dioxide emissions from printing Tampere University dissertations have been compensated.

PunaMusta Oy – Yliopistopaino
Joensuu 2022

PREFACE

My dissertation project started when Timo "Rafu" Raitalaakso invited me to a Solita Science meeting. Rafu, your support made this dissertation complete – thank you for everything. Since then, we have become true friends. Our collaboration is so complete that it is no longer possible to specify which one came up with the idea. In that Solita Science meeting, I also met Timo Aho, Tommi Mikkonen and Timo Lehtonen, each of you deserve mention in this book. Timo Aho has been inspiring and guiding me on this journey. Tommi Mikkonen has followed and boosted my scientific career, you have been the co-author in several articles with me, and it has been delightful to co-operate with you. Timo Lehtonen, in hockey terms, you have fed in the blade everything needed to graduate PhD.

My supervisor Hannu-Matti Järvinen has guided me through studying for the doctorate. Elina Orava, you have helped me fill out the required documents at the right time, and in what a friendly way you did it. Writing a dissertation is full of ups and downs. Thank you, Niina Herttuala and Essi Peltonen, for sharing and caring with the writing process. My family, you are my support when needed. Thanks for getting support for this dissertation.

Real project cases are the best places to solve problems. Juha-Petri Järvi, you created an environment where a team could achieve success, and I could test the theory in practice. Solita has created an inspiring atmosphere working and writing publications. Just to mention few of you: Pekka Ahola, Manu Setälä, Aki Aapaoja and Timo Honko.

Special thanks I would like to give to the following crew: Laura Hokkanen literature review and general support, Paavo Toivanen proofreading, Kari Systä arranged inspiring seminars to obtain credit points.

Mikko Puonti, 15.10.2022, Tampere, Finland

Dedicated to the memory of Janne Pirkkanen.

Some Further Acknowledgements

The scientific work that led to this PhD. thesis was primarily supported by Solita and Finnish Funding Agency for Innovation (TEKES) projects: DIGILE Need for Speed program, Gateway to Mercury Business. Today, TEKES is Business Finland, government organization for innovation funding and trade, travel and investment promotion. I have received a grant from the city of Tampere Science Grant Committee for the publication and printing costs of my dissertation.

ABSTRACT

Continuous delivery is an activity in the field of continuous software engineering. Data warehousing, on the other hand, lie within information systems research. This dissertation combines these two traditionally separate concerns of continuous delivery and data warehousing.

This dissertation's motivation stems from a practical problem: how to shorten the time from a reporting idea until it is available for users. Data warehousing has traditionally been considered tedious and delicate. In data warehousing, distinct steps take place one after another in a predefined unalterable sequence. Another traditional aspect of data warehousing is bringing everything at once to a production environment, where all the pieces of a data warehouse are in place before production use. If development follows agile iterations, why are the releases in production not following the same iterations?

This dissertation introduces how reporting and data warehouse teams can synchronously build business intelligence solutions in increments. Joint working enhances communication between developers and shortens the feedback cycle from an end-user to developers, and makes the feedback more direct. Continuous delivery practices support releasing frequently to a production environment. A two-layer data warehouse architecture separates analytical and transactional processing. Separating different processing targets enables better testing and, thus, continuous delivery. When frequently deploying with continuous delivery practices, automating transformation creation in data warehousing reduces the development time. This dissertation introduces an information model for automating the implementation of transformations, getting data into a data warehouse and getting data out of it.

The research evaluation followed the design science guidelines. Research for this dissertation collaborated with the industry. These ideas have been tested on real projects with promising results, and thus they have been proven to work.

TIIVISTELMÄ

Jatkuva käyttöönotto (continuous delivery) on ohjelmistokehityksen tutkimusala. Tietovarastointi (data warehousing) puolestaan kuuluu informaatio- ja tietojärjestelmätutkimuksen alaan. Tässä väitöskirjassa yhdistyvät nämä kaksi perinteisesti erillistä tutkimusaluetta jatkuva käyttöönotto ja tietovarastointi.

Tämän väitöskirjan motivaatio kumpuaa käytännön ongelmasta: kuinka lyhentää aikaa ideasta analysoida jotain siihen, että analyysi on käyttäjien saatavilla. Tietovarastointia on perinteisesti pidetty monimutkaisena ja siten herkkänä virheille. Tietovarastoinnissa erilliset vaiheet tapahtuvat peräkkäin ennalta määritellyssä järjestyksessä. Perinteinen tapa tietovarastoinnissa on ottaa koko ratkaisu kerralla tuotantokäyttöön, jossa kaikki tietovaraston palaset ovat paikoillaan ennen tuotantokäyttöä. Mikäli kehitys seuraa lyhyitä iteraatioita, miksi käyttöönotot tuotantoon eivät seuraa näitä iteraatioita?

Tämä väitöskirja esittelee kuinka raportointi- ja tietovarastointitiimit voivat rakentaa yhtäaikaa raportointiratkaisuja (business intelligence) vaiheittain. Yhteistyö tehostaa kehittäjien välistä kommunikaatiota ja lyhentää palautesykliä loppukäyttäjältä kehittäjille mikä tekee palautteesta suorempaa. Jatkuvan käyttöönoton käytännöt tukevat julkaisemista usein tuotantoympäristöön. Kaksikerroksinen tietovarastoarkkitehtuuri erottaa analyyttisen ja tapahtumapohjaisen käsittelyn. Erilaisten käsittelyjen erottaminen mahdollistaa paremman testauksen ja siten jatkuvan käyttöönoton. Käytettäessä jatkuvaa käyttöönottoa, voidaan kehitysaikaa lyhentää myös automatisoimalla tietomuunnosten toteutustyötä. Tämä väitöskirja esittelee tietomallin tietomuunnosten automatisoinnin toteuttamista varten, niin tiedon saattamiseksi tietovarastoon kuin tiedon hyödyntämiseen tietovarastosta.

Tutkimuksen arvioinnissa noudatettiin suunnittelutieteen suuntaviivoja. Tutkimus tehtiin yhteistyössä teollisuuden ja yliopistojen kanssa. Näitä ideoita on testattu todellisissa projekteissa lupaavin tuloksin ja siten ne on todistettu toimiviksi.

CONTENTS

List of Figures

List of Tables

ABBREVIATIONS

BI	Business intelligence (BI) solutions combine data gathering, data storage, and knowledge management with analytical tools to present complex internal and competitive information to planners and decision makers [41].
DV	The Data Vault (DV) is a detail oriented, historical tracking and uniquely linked set of normalized tables that support one or more functional areas of business [32].
DW	A data warehouse (DW) is a subject-oriented, integrated, non-volatile, and time-variant collection of data to support management decisions [21].
DWA	Data warehouse automation (DWA) automate data warehouse implementation with the principles of design patterns and using a metadata repository. .
EDW	Enterprise Data Warehouse (EDW) create an integrated and standardized data foundation that facilitates improved analytics and reporting [49]. The enterprise data warehouse is a layer in the data warehouse in this dissertation, whereas other may use EDW in the meaning of the data warehouse.
ELT	Extract, load, transform (ELT) load data directly into the data warehouse without external transformations, data cleansing or normalization [10]. The target is to convert the compute-intensive parts of the ETL process into database processing.
ETL	Extract, transform, load (ETL) is set of processes to read, modify and store data into a database or into a file.

LIST OF INCLUDED PUBLICATIONS

[P1] Mikko Puonti, Timo Lehtonen, Antti Luoto, Timo Aaltonen, and Timo Aho. "Towards Agile Enterprise Data Warehousing". In: *ICSEA 2016, The Eleventh International Conference on Software Engineering Advances* (Aug. 2016), pp. 228–232.

[P2] Mikko Puonti, Juha Järvi, and Tommi Mikkonen. "A Continuous Delivery Framework for Business Intelligence". In: *Frontiers in Artificial Intelligence and Applications* 301 (2018), pp. 248–262. DOI: 10.3233/978-1-61499-834-1-248.

[P3] Mikko Puonti, Timo Raitalaakso, Timo Aho, and Tommi Mikkonen. "Automating Transformations in Data Vault Data Warehouse Loads". In: *Frontiers in Artificial Intelligence and Applications* 292 (2017), pp. 215–230. DOI: 10.3233/978-1-61499-720-7-215.

[P4] Mikko Puonti and Timo Raitalaakso. "Data Vault Mappings to Dimensional Model Using Schema Matching". In: *Research and Practical Issues of Enterprise Information Systems* 375 (Dec. 2019), pp. 55–64. DOI: 10.1007/978-3-030-37632-1_5.

Author's contribution

[P1] The author was the main responsible person for the idea of how to convert from the currently serial workflow to parallel by using a dimensional model as an agreement with the different teams. Antti Luoto did a literature review and wrote related work. The author wrote chapters III. Data Warehousing and Reporting Process, and IV. Data Modeling. All other authors edited and approved the manuscript.

[P2] The author was the main responsible person for documenting a continuous delivery framework for business intelligence. Juha Järvi introduced the framework in real-life implementation and commented on the manuscript. Tommi Mikkonen advised the methods and par-

ticipated in the writing.

[P3] The author was the main responsible person of writing the publication and of literature review. Timo Raitalaakso implemented PL/SQL proof of concept implementation of the data map tool. All other authors edited and approved the manuscript.

[P4] The author was the main responsible person of writing the publication. Timo Raitalaakso came up with the idea to take advantage of data flow to create a schema matcher, he experimented the theory with the Northwind example and edited the manuscript.

1 INTRODUCTION

How long does it take to add one new attribute to an existing report and publish that for production use? This dissertation's motivation stems from practical problems: how to shorten the timeframe from a reporting idea to that it is in production use and how to do it in a reliable and repeatable way. Business intelligence tools almost invariably require manual steps when moving code from one environment to another. Deployment to production is a stress-filled moment and requires a maintenance break to the data warehouse and reporting portal. On the other hand, we get genuine feedback from users when reports are available in a production environment. This frame our research: How to release often to production, even if the tools do not support continuous delivery.

1.1 Motivation

Data warehousing contains a collection of tools integrating data from operational systems in one place for decision support [56]. Data warehousing has traditionally been considered tedious and delicate [21]. Distinct steps take place one after another in a predefined unalterable sequence. Another traditional aspect of data warehousing is implementing all the pieces of a data warehouse first and taking them into production use at the end of a project. This approach is no longer justified. Traditional practices are preventing people from changing their way of building data warehouses.

Business intelligence is a collection of activities to understand business situations and based on analysis making decisions [47]. In a business intelligence solution, both data and that it is correct are essential. The key user is the best person to validate that business intelligence solutions offer correct information [48]. Unfortunately, the key user is a human and has only limited resources to validate the data in a business intelligence solution. What if we utilize the expertise of a key user, but instead of a person performing the testing, we would automate the tests?

The root cause of failure in data warehousing projects is disconnected understanding

and expectations between developers and users [7]. Feedback is a source of satisfaction to developers and the end-user (client) [15]. At the beginning of the research work, it took a long time to get a report available for users. Users of a report involved in the development work provided feedback only after it was available for users. Business intelligence developers are developing new all the time. When they received feedback, it focused on the past implementation, which they did not fully remember. It was frustrating to the developers.

Traditional data warehousing uses development, test, QA (Quality Assurance), and production environments. In the test environment, technical tests are executed by a developer. The QA environment is for key user, and it should match with the data used in production. If we automate the tests, we could start using continuous delivery. Continuos delivery [20] incorporates build, test, and release phases. The key is to concentrate on your cycle time, or actually that you can release as often as you want. Poppendieck asked, "How long would it take your organization to deploy a change that involves just one single line of code? Do you do this on a repeatable, reliable basis?" [45, page 92].

In business intelligence solutions, shared environments usually mean shared version control, if version control is in use at all. There are different tools for different tasks; data modelling, ETL loadings, and the actual report development. These tools almost invariably require manual steps when moving code from one environment to another. The tools do not work together but separately. Deployment to production is a stress-filled weekend, rather than executing a couple of scripts from your own computer. It leads to the state that cycle time is weeks or months.

There exist several tools for data warehouse automation (see Section 2.3). However, the tools use development, test, QA, and production environments where developers share these environments. Technically we have tools that enable us to deploy in different environments. Still, we are hanging in shared environments.

The question is how long it takes to add one new attribute to a report, which includes all needed steps - at least following.

- Create a backlog item for adding an attribute to the report
- Implement changes in a database
- Create transformations to populate that attribute
- Modify a report
- Validated that modified report have added attribute
- Publish modified report to the end-users

Modifications are done first in a development environment. Then the report is deployed to a test environment where it is validated. After validation, publish the report in the production environment. Is this in a specific organisation days, weeks, or even several months?

1.2 Research Problem

In business intelligence, we get genuine feedback after users access the reports. Several tasks must be finished before a reporting idea is a ready-made report. These tasks include at least data modelling, data integration, report implementation and deployment for users. The traditional workflow follows these tasks sequentially, so the time from idea to a finished report is long. We aim to shorten this cycle time: from idea to a report available in production use.

We can't bite the moon out of the sky, so we focus on how to get reports more often for production use. The continuous delivery approach releases software often with build, test and deployment automation. We focus our work mitigation efforts on data integration, which is the most laborious task [25]. With the scope of data integration, we frame our research to continuous delivery in data warehousing.

This dissertation combines continuous delivery to data warehousing, see Figure 1.1. Continuous delivery is an activity in the field of continuous software engineering [9]. Information systems research includes business intelligence [41] and especially data warehouses [21]. Business intelligence combines data gathering and presenting information for users as planners or decision-makers. The dissertation research question is how to shorten the time from a reporting idea until it is available for users? Implementing reports is part of business intelligence. Research focuses on the most laborious phase of business intelligence, data gathering. The data warehouse automation mitigates the work effort needed in data warehousing. In Figure 1.1, the related concepts for the research problem are business intelligence and data warehouse automation.

1.3 Research Environment

Research work for this dissertation started in the Need for Speed program [22]. Need for Speed program was an industry-driven research program in Finland for the years 2014-2017. The budget exceeded 80 million euros, annually around 20 million euros, which makes it the biggest national investment in software-related research. The program was executed

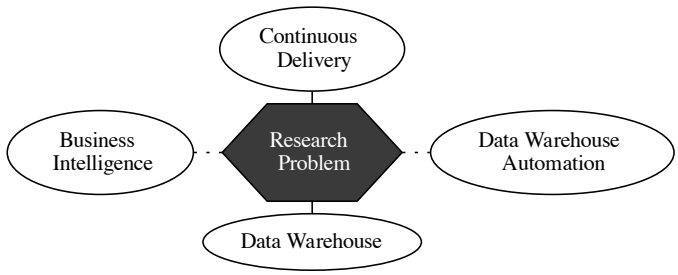

Figure 1.1 The research problem is continuous delivery in data warehousing, related concepts are business intelligence and data warehouse automation.

jointly by the industry and academia. Numerous research institutes and companies participated in the program. The program had business case driven short-term goals [38]. The only restrictions for a business case were that there should always be more than one participant and that no business case is populated by only research institutions. Research for publications [P1, P3] was part of business cases. There were quarterly reviews in the N4S program. Each quarterly review consisted of workshops, training events, presentations, strategy discussions and planning sessions. The idea for publishing our publication [P3] came up in one of the quarterly reviews.

Data warehouse projects are very labour-intensive; there was a need to improve the methods, tools, and processes, for example, by automating tasks. In research program Solita decided to seek a way to increase the degree of automation in these projects. In addition, good practices and tools to manage project resources, such as version control tools, needed to be put into more active and systematic use. The goal was to increase the productivity of the data warehouse and analytics projects with the aim of gaining at least a six-fold increase in the speed of development. The initial problem originated from the industry. For solving the problem, some solution candidates were created and validated in collaboration with academic partners. Validation for the feasibility of the solutions came directly from applying the solution candidates in real client projects. [38]

1.4 Objectives and Scope

We aim to create meaningful reporting in a managed way. Meaningful reporting answers to questions that reporting users ask. In a managed way means that we release often, and reporting requirements guide the development backlog. When the reports are available for users, we get feedback from the users. In a traditional sequential workflow, the time from a reporting idea to the report available in production is long. Our unifying research question is: how to shorten the time from a reporting idea until it is available for users?

The continuous delivery approach releases software often with build, test and deployment automation. We research how to use continuous delivery in the domain of business intelligence. To reduce the time from a reporting idea to the report available, we mitigate the manual work with automation (see Figure 1.2). Data warehouse implementation is the most laborious phase in a business intelligence solution implementation. Automating the data warehouse implementation reduces the time used in report implementation. This focus scopes our work to continuous delivery in data warehousing. We divided this into more detailed research questions. Research questions are grouped in Figure 1.2 to enablers and automating implementation.

- RQ1: How to implement a business intelligence solution in increments jointly with reporting and data warehousing teams?
- RQ2: What is suitable architecture for a business intelligence solution to enable continuous delivery?
- RQ3: How to automate the implementation of getting data into an enterprise data warehouse, modelled with data vault methodology?
- RQ4: How to automate the implementation of getting data out from an enterprise data warehouse, modelled with data vault methodology?

Research about business intelligence systems is limited, although the industry uses business intelligence systems widely [41]. A business intelligence system fetches data from different sources to an integrated data warehouse and visualises the information from the data warehouse with an analytical tool. After the reports are available, we get feedback from the users. Unfortunately, getting the reports available contains many work phases and may take a long time. We are interested in shortening the time from a reporting requirement to publishing a report to the users. The problem is that developers will get feedback from users only after the report is available to users. We noted that a reporting team can only imple-

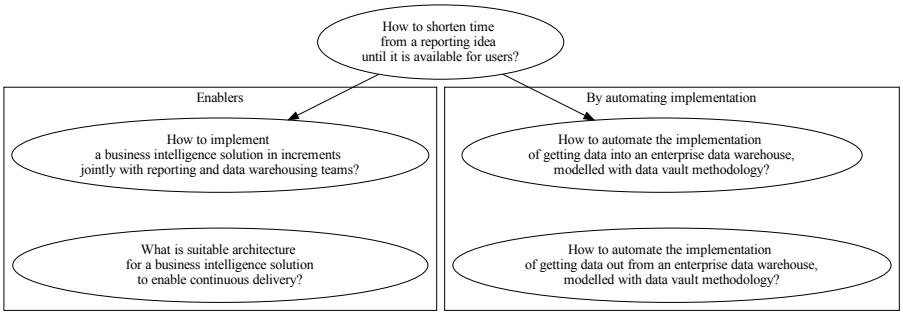

Figure 1.2 Research questions as enablers or by automating implementation work

ment reports that have data available in a data warehouse. There is a dependency between a data warehouse team and a reporting team. Because of the dependency first, a data warehouse team needs to make data available in a data warehouse. After the data was available, the reporting team continued implementing the report. In agile development, especially in the scrum, this needs two increments (sprints) to publish a report to the users. This raised us the research question 1: How to implement a business intelligence solution in increments jointly with reporting and data warehousing teams? To mitigate the need for two increments, we conformed the research question 1 (RQ1).

A solution to research question 1 would shorten developing time from a reporting requirement to a published report to the users. Currently, this is an issue in the industry. The users do not want to be involved in the development of reports due to the development timeline.

Traditionally business intelligence projects use shared environments to develop and test the solution before deploying it to production. Deploying implementation between environments often requires manually copying binary files from one place to another. Manual file copying is error-prone. There may be different configurations between different environments. Using shared environments allows only one implementation at a time. Shared environments cause a dependency on implementations and things can only be taken into production when the previous implementation is ready in production. Shared environments complicate development work. Deployment with manual steps is not repeatable and reliable. Lack of a repeatable and reliable way of deploying to production leads to scheduled service interruptions. Depending on the frequency of service interruptions, the deployment may contain several items, and the problem is how to verify that everything will

go smoothly. Continuous delivery [20] is quite the opposite, releasing software with a repeatable and reliable process. We started to investigate how to enable continuous delivery in business intelligence projects. Fort that purpose is research question 2: What is suitable architecture for a business intelligence solution to enable continuous delivery? Suitable architecture for continuous delivery in business intelligence would allow us to release often, perhaps without the service interruptions. At least there is no extra delay getting a report available for users.

The most laborious phase of building a data warehouse is to create data transformations. These data transformations are called the extract, transform load (ETL) process. As a rule of thumb, ETL consumes 70 per cent of the resources in a typical data warehouse implementation [25]. The data vault modelling technique [32] has become more common in data warehousing. Data vault splits the model into different entity types, which increases the number of tables and thus the amount of ETL work. The usage of the data vault encourages templating or even automating the transformation implementation [19, 36]. Unfortunately, majority of the automation tools are proprietary and intended only for specific technologies. Because of this, each project uses different templating or different script sets to automate data vault loadings. Each project creates its way of automating data vault loadings, and they all start from scratch. There is not much scientific research literature about data vault automatisation. To mitigate work, we present research question 3: How to automate the implementation of getting data into an enterprise data warehouse, modelled with data vault methodology? Automating implementation should decrease the needed effort to create ETL loads. By generating the implementation code, we reduce the possibilities for human errors, and we enable deployments with a reliable process. By publishing the results to a scientific audience, we enable others to automate their solutions, and they will not start from scratch nor be dependent on a proprietary solution.

Getting data into a data warehouse is widely automated. There are proprietary automation solutions available. Automating getting data out from a data vault is not so much studied. There exists scientific literature regarding the design of a dimensional model from a data vault [12, 27]. As we are following the model-first approach, we have in place the dimensional model, and we do not benefit from design automation. We should automate the mapping between a source (data vault) and a target (dimensional model). Mappings between a data vault and a dimensional model based on a hub or a link granularity, therefore, data flow is possible to suggest. The actual transformation code is generated based on mapping information. The generated transforms can be XML-files for ETL-tool, or SQL-

queries or other programming language code. In this research, the generated code was SQL-queries. Getting data out contains manual work, and implementation varies depending on who has written the SQL query. To automate this part and harmonise the implementation code, we raised research question 4: How to automate the implementation of getting data out from an enterprise data warehouse, modelled with data vault methodology? Similarly to getting data in phase, automating implementation should decrease the needed effort to create ETL loads. By generating the implementation code, we reduce the possibilities for human errors and enable deployments with a reliable process. Currently, there is a lack of generated code in the getting data out phase. There is a wide variance in implementation logic in the manually written SQL queries.

1.5 Research method

Design science research focuses on developing an artefact with the explicit intention of improving the functional performance of the artefact. The purpose of information systems implementation is to improve that organisation [16]. The behavioural science paradigm seeks to develop and verify theories that explain organisational behaviour. The design science paradigm seeks to extend human boundaries and organisational capabilities by creating new and innovative artefacts [16]. Information system research may focus on defining an idea, a practice or a technical solution, which all are different types of artefacts.

The design science research process is iterative. Nunamaker et al. [42] present a systems development research methodology (SDRM). SDRM intends to include elements of both the social and engineering approaches. Five phases in SDRM are: construct a conceptual framework, develop a system architecture, analyze & design the system, build the system, and observe & evaluate the system. Vaishnavi et al. [57] present a design science research process model (DSRPM). Their model is an adaption of the Takeda et al. [55] computable design process model. The process steps are the following: awareness of problem, suggestion, development, evaluation, and conclusion. From each step it is possible to go back to the first step, awareness of problem. The process iterates until the last step, conclusion output, is suitable to the problem. Peffers et al. [43] introduce design science research methodology (DSRM) that has multiple possible entry points. The DSRM process can start with: problem centered initiation, objective centered solution, design & development centered initiation, or be client/context initiated. This research originated from problems identified in the industry. Design science research creates a suitable framework for conducting the

research process and evaluating the artefacts. Venable et al. [58] suggest choosing an objectivist, positivist methodology such as SDRM [42], DSRPM [57], or DSRM [43], when planning and organising research where scientific results have to be objective. This research used the design science research process model (DSRPM). The evaluation of the designed artefacts uses Hevner et al. [16] guidelines. For each artefact, the research followed the general design cycle [6, Figure 3.2] [57, Figure 3].

1.6 Dissertation Structure

Chapter 1 motivates why continuous delivery is beneficial in data warehousing. It presents the research problem and the research environment. Objectives and scope introduce the research questions. The used research method is design science research.

Chapter 2 is a literature review of key concepts. It introduces continuous delivery following how Humble and Farley present it. Then it draws the business intelligence and data warehousing landscape, definitions, architectures and modelling techniques. Next, it explains agile in business intelligence and data warehousing. Lastly, it presents an overview of data warehousing automation.

Chapter 3 presents research contribution. We use design science guidelines to evaluate introduced artefacts. After evaluation, each research question has separated sections. After individual research questions, it concludes the research contribution.

Chapter 4 presents theoretical and practical implications. The reliability and validity are estimated based on literature and using the design science research method.

At the end of the dissertation, conclude a summary in Chapter 5.

2 BACKGROUND AND RELATED WORK

Continuous delivery is a set of practices to enable reliable process release of new functionalities in production use. Continuous delivery uses phases of build, test, and deploy to ensure the quality of new functionality. Traditionally business intelligence projects use shared environments to develop and test the solution before deploying new functionality to a production environment. Testing the solution happens manually, and deployments are often manual processes. Business intelligence solutions implementations use agile methodology. Although, there are opinions that data-centric projects are troublesome to manage with agile methodologies. Data warehouse implementation consumes most of the work needed in business intelligence solutions. There is a place for automating different phases of data warehouse implementation.

2.1 Continuous Delivery

Jez Humble and David Farley presented continuous delivery in their book *Continuous delivery: reliable software releases through build, test, and deployment automation* [20]. Note the viewpoint "If somebody thinks of a good idea, how do we deliver it to users as quickly as possible?" [20, page 3]. The most important is the deployment pipeline, which starts after a developer has committed his code to a version control system. Yes – after every commit, which is the minimal change to your system. We simplify the deployment pipeline in Figure 2.1. In the commit, stage code is compiled and built in the deployable software artefacts. The commit stage is shortly also referred to as a build phase. The test phase validates that everything works and, it includes acceptance testing, capacity testing and optional manual testing. Testing is a seminal part of continuous delivery. Deployment update an environment with the compiled artefacts. After a successful test phase, in every needed environment, we deploy to production, which we refer to as a release. The deployment pipeline is, in essence, an automated implementation of your business intelligence solutions build, deploy, test and release processes.

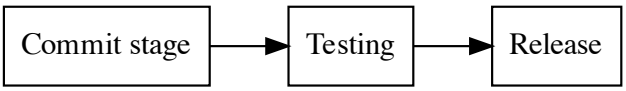

Figure 2.1 Simplified deployment pipeline.

Very often, we hear the term continuous deployment used. Continuous deployment differs from continuous delivery – deployment is fully automated and does not contain a manual phase. In the beginning, it sounds that the difference is minimal – one button press. After rethinking, you may notice that the difference is a giant leap from continuous delivery to fully automatic testing and deployment.

Continuous delivery allows manual phases. We encourage automating the manual phases, at least those that repeat often. Continuous delivery is a set of practices to enable reliable release. A desirable release duration is a short timeframe from idea to production use.

2.2 Agile Business Intelligence / Data Warehousing

Business intelligence converts data into useful information. If contains data processing and decision making. A data warehouse is the core component of business intelligence. There are several data modelling techniques for data warehousing. In a structure-oriented data warehouse there are different layers. For each layer there is a purposeful data modelling technique. Agile analytics cover agile methods as well as activities towards continuous delivery.

2.2.1 Business Intelligence

Business intelligence solutions combine operational data with analytical tools to present complex and competitive information to planners and decision-makers [41]. Business intelligence converts data into useful information and, through human analysis, into knowledge [41]. The main idea for business intelligence is identifying information needs and processing the data and information gathered, into valuable managerial knowledge and intelligence [44]. Processing data is getting data available and presenting it in a functional format.

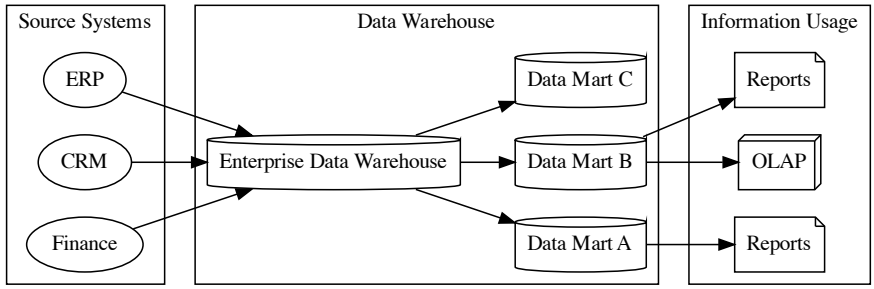

Figure 2.2 Typical business intelligence architecture

Combining data is to fetch data and integrate that to a central repository in business intelligence to the data warehouse. There is a discussion on how important it is to gain knowledge from semi-structured data. Thus the foundation of business intelligence must be relational data. Figure 2.2 presents typical business intelligence architecture. A data warehouse is the core component of business intelligence. However, business intelligence is a broader term than the data warehouse. In addition to data warehousing, business intelligence includes information usage tools and decision making.

2.2.2 Data Warehousing

A data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data to support management decisions [21]. The data warehouse contains a database, metadata about the data and data transfer and modifications, i.e. extract, transform, load process. The following data warehouse requirements follow those written by Kimball [26]. The data warehouse must make an organization's information easily accessible in an understandable form. It presents the organization's information consistently [21, 26, 56]. The data warehouse must be adaptative and resilient to change [26, 56]. The data warehouse must effectively control access to the organization's confidential information. The data warehouse must have the correct data to support decision making [21, 26, 56]. The reporting users must accept the data warehouse if it is to be deemed successful [26]. The data warehouse usage is sometimes optional. User acceptance has more to do with simplicity than anything else.

An essential part of the data warehouse is the extract, transform, load process [25]. The first step is extracting data from source systems. The second step is the data quality and cleansing operations. Third, integrating data with other data sources. Finally, delivering data for information usage purposes. ETL data processing happens in a specific server before loading to the data warehouse. Another way to handle the data transformations is to use database operations . The data is first loaded to the database and then transformed inside the database, called extract, load, transform (ELT) as the order of the loading differs from the ETL process [10].

2.2.3 Data Warehouse Architectures

There are two groups of data warehouse architectures: structure-oriented and department-oriented [14, 4]. The structure-oriented architectures have a variable number of layers. Single-layer architecture is virtual and implemented as dimensional views over operational data [14, 4]. Single-layer architecture is rare in practice. The two-layer architecture separates analytical and transactional data processing [14, 4]. Although it is called two-layer architecture, there are four data flow stages: source layer, data staging, data warehouse layer and analysis [14, 4]. Typical business intelligence architecture in Figure 2.2 follows two-layer architecture. In three-layer architecture layers are implemented physically: source layer, reconciled layer and data warehouse layer [4]. Department-oriented architectures are: independent data marts, bus architecture, hub and spoke architecture, centralized architecture and distributed architecture [4, 2] .

A data lake is a centralized data repository that can store a multitude of data ranging from structured or semi-structured data to completely unstructured data [54].

Traditionally business intelligence projects use shared environments to develop and test the solution before deploying it to production. A development environment is for development. A test environment is for technically testing that everything works as defined. A quality assurance (QA) environment cover test cases with actual data and ensures that everything is ready for production use. Each environment has costs and needs for management. In some cases, the project uses the test environment for quality assurance tests. Usually, there is limited access for reporting users to the QA environment. The last environment is the production environment, where all the actual users are.

2.2.4 Data Modelling Techniques for Data Warehousing

It is essential to understand the phases of data warehouse design as it is part of the business intelligence solution. Data modelling generally involves three stages, conceptual, logical, and physical [17, 53]. Conceptual models focus on business concepts as entities and their associations with other business concepts. Logical modelling adds attributes to the entities. Physical modelling goes even deeper, while it gives exact data type definitions to the attributes and how the data physically is stored.

At the beginning of a data modelling process, the approach may be top-down or bottom-up [51]. In a bottom-up approach, modelling starts with gathering details, mostly from existing systems. A modeller classifies the gathered information. Based on the classification, modeller identifies the things or entities. The top-down approach is the opposite. Modeller educates a group of people on what an entity is. Then this group identify entities. Regarding Sharp and McDermott [51] with the top-down, there are still needs for some sorting and synthesis. In the context of data warehouse design, the data model design can be data first or a model first approach. The model-first approach creates a data warehouse model with the top-down or the bottom-up modelling approach. After a data models exists, starts the data loadings implementation. In a data-first approach, a modeller adds entities to the model to fit the data sources when implementing the data loadings. The data sources drive the modelling of the data warehouse in the data-first approach.

Inmon and Kimball are the pioneers of data warehousing. According to Inmon, in the operational environment there is present information, and the data warehouse contains the present and the historical information [21]. Note that the records in the data warehouse do not overlap, and time is associated with each record in the data warehouse.There is no point in bringing data over several operational environments into the data warehouse without integrating it.In Figure 2.2, the enterprise data warehouse is the same as Inmon's data warehouse. Data marts are for departmental usage of the data warehouse data [21]. The data mart model uses dimensional modelling. According to Kimball, dimensional modelling is the most viable technique for delivering data to data warehouse use [26, page 10].

A dimensional model is also known as a star schema. The model looks like a star, where the fact table is the centre of a star and dimension tables forms the star claws. Kimball [26] presents a methodology to design a dimensional data warehouse.

1. Select the business process to model.
2. Declare the grain of the business process.

3. Choose the dimensions that apply to each fact table row.

4. Identify the numeric facts that will populate each fact table row.

Change in a business requirement requires far too much re-engineering and data modifications when using only star schemas [17, Page 329]. For this reason, the enterprise data warehouse (EDW) layer is added to the data warehouse before data marts, see Figure 2.2. The EDW layer generally is designed with ensemble modelling principles. Ensemble modelling principles support changing business requirements. Hyper normalization is a family of data modelling techniques that all employ ensemble data modelling [17]. According to Hughes, using hyper modelling can eliminate 30-90 per cent of the labour required to implement an EDW [17].

The most widely used ensemble modelling technique is data vault modelling. Data vault modelling uses three core entity types: hub, link and satellite[18, 19, 30, 31, 32, 33, 35, 36]. In addition to these, there are entity type reference (stand-alone) and query assistant tables [19, 33]. The natural business key is a unique identifier for a business concept [19]. A hub presents the business key for the business concept [30]. A link is an intersection of business keys [30]. Just like the hub, it contains no descriptive information [18]. A satellite is an entity that holds descriptive information for a hub or a link [18, 30]. The satellite stores the whole history of the descriptive data by storing a new data row with a timestamp every time there is a change to any of its attributes [18, 30]. A reference is a lookup table that includes valid code values and their corresponding definitions [19]. A reference table is a separate table and refers to other tables logically, but it will not have any foreign key references [30].

2.2.5 Agile Analytics

This section concentrates on the question, how to create a business intelligence solution with agile methodologies. Successful business intelligence projects must follow an agile approach [8, 46]. In a traditional sequential approach, planning took place for all steps before the project started. Agile analytics development is iterative, incremental and evolutionary, in short, iterations that are generally one to three weeks long. Each small increment adds user value functionality to the system. Each developed increment must be tested and debugged during the development iteration. A user valued functionality is "Done" when it is of production quality, it is part of the system and, developers are proud of their work [7, Chapter 1]. For everyone, the "Done" definition does not require the release to the production. Still, each agile iteration aims to provide potentially shippable products after the

iteration, even if those are not released to production [46].

Batra [3] executed a survey on which factors contribute to the success of DW/BIA (business intelligence and analytics). Research revealed two different groups: agile-plan balanced and agile-heavy. Agile-plan balanced represented 68.5% of the respondents, and agile-heavy accounted for 31.5% of the sample. Batra also mentioned that the plan-only approach would be rare if not altogether absent.

The plan-only approach is the traditional sequential way. The main difference between traditional plan-only and agile is that database design and implementation is ready before the analytical tool implementation work starts. As a worst-case, the entire data warehouse, including data marts, was implemented before the reporting work started.

Teorey et al. [56] presented the activities of a data warehouse life cycle. The process starts with project planning and business requirements definition. When the plan and the requirements are aligned, design and implementation can proceed in three different threads. One of the threads covers platform issues. The second thread covers data issues, including modelling and physical design, with data processing development. The third thread handles analytic application specification and development. These three threads join before deployment. After deployment come maintenance and growth, and detecting requirement changes. If adjustments are needed, the cycle repeats. [56, Figure 10.2]

There are opinions that data-centric projects are troublesome to manage with agile methodologies. To take care of issues in data quality, some bi teams wait until the data is ready in the EDW (placed there by a separate EDW team) before starting the report development [50]. Shamsi [50] presents an opinion that Scrum and Extreme Programming (XP) will not work when building the end-to-end business intelligence solution, including expanding the necessary EDW components. However, he continues that this does not mean that you can not go agile [50].

2.3 Data Warehouse Automation

Data warehouse automation (DWA) aims to automate data warehouse implementation. Automation supports the agile approach by offering an ability to deliver faster and with fewer resources. However, using automation does not require any agile methodology.

In this dissertation, data warehouse automation means automating the builds and deployments of a data warehouse. Build automation creates the actual implementation code by generating it based on a metadatabase. Deployment automation makes deploying to dif-

ferent environments easier. Deployment Automation can be implemented with a tool or by creating migration scripts from one version to another, where we prefer the latter. The continuous delivery build phase creates an artefact. That same artefact deploys to different environments. When the automation tool manages the migrations from one version to another, the automation tool itself may create an error, which is almost impossible to test in advance.

Data warehouse automation focuses on automating every step in data warehousing. One of the steps is to automate the modelling (design) phase. There is research on how to automatically design a dimensional star schema from an operational database [5, 13, 40] or a data vault [12, 27]. Jovanovic and Bojicic introduce the idea of converting patterns from an entity-relationship model to a data vault model [23]. Although these methods automate the design phase, they require human intervention.We have separated modelling from automation because we believe modelling brings value and needs to be done with people, not as suggested automatically by an algorithm.

It seems that products made for data warehouse automation have a short life cycle. Products have their limitations and are suitable for a particular technology or a modelling method. Ajilius competitor web page lists 24 products for data warehouse automation in Table A.1. In the year 2015, there were only 14 products listed. Growth in 4 years has been ten products (71%). There are also leavers: BIReady did not exist in 2019, and currently, Ajilius does not exist in the market. The data map tool presented in publication [P3] uses Oracle database technology. It was in use by several customers. Those customers have migrated from a Oracle database to a cloud database, and consequently there are no active users for the data map anymore. Maybe the first tool for data warehouse automation with data vault modelling was Rapidace. Rapidace has the capability to forward engineer data models from source to staging, from staging to data vault, from data vault to star schema [34]. Sixteen products support dimensional methodology, nine data vault in 2015 only three support that. There are also some products outside the Ajilius listing example, since, for instance, Vaultspeed existed in 2018 [11].

3 RESULTS

Design science research develops an artefact in the form of a construct, a model, a method, or an instantiation [16]. This research introduces five artefacts and two artefacts to mention as proof of concept artefacts. The artefacts are assessed with guidelines to evaluate: the artefact itself, problem relevance, and research process. Artefacts are validated based on research questions. Finally, the research contributes to how to shorten the time from a reporting idea until it is available for users?

3.1 Designed Artefacts

Hevner et al. present 7 guidelines in their process [16, Table 1]. Guideline 1: design as an artefact produces an artefact in the form of a construct, a model, a method, or an instantiation. Guideline 2: problem relevance: addressing unsolved and relevant business problems. Guideline 3: design evaluation: rigorously demonstrating how the utility, quality and efficacy of a design artefact is evaluated. Guideline 4: research contributions of design artefact, design foundations or design methodologies. Guideline 5: research rigour requires the application of rigorous methods in the construction and evaluation of the design artefact. Guideline 6: design as a search process to discover an effective solution to a problem. Guideline 7: communication of research: present designed artefact both to technology-oriented and management-oriented audiences. Next, we use the Hevner et al. seven guidelines to introduce and evaluate artefacts included in this dissertation.

3.1.1 Guideline 1: Design as an Artefact

The result of design science research in information science is a purposeful IT artefact [16]. Publication [P1] proposes a process change from serial to parallel working. Creating a dimensional model based on a conceptual model and an interface specification enables parallel working [P1]. Publication [P2] contains the parallel working between reporting and data warehouse implementation. In addition, publication [P2] starts with the step *plan incre-*

ment package and implementation ends in the *deployment* step [P2, Figure 2]. To present how the populate example dataset depends on source system interface specification and what sub-steps are in the data integration step, we have extended the process diagram in the Figure 3.1. Information usage development is defined mainly as creating a report. Such a report will not work if it cannot display data. The designed publish layer is populated or generated with data for report development purposes. In the best case, sample data is from the source system. Data integration starts with the step *Investigate data source*, where the data is identified at the source system table level. Based on investigation results, the data is already integrated earlier in the enterprise data warehouse, or then it is a new source system table to the data warehouse. If the source system table is new, insert table information into the ETL software. Data transformation development follows the process introduced in publication [P3]. Development happens with the model first approach, modelling layers before implementing transformations.

Once we have developed a method, we need an architecture that supports it. A simplified version of the architecture in publication [P2] is in Figure 3.2. This architecture follows a common practice, separating persistent storage from information usage [7, 19, 18, 21, 26]. Blažić et al. and Golfarelli and Rizzi use different naming, where the functionality of two layer architecture is the same than architecture presented in the Figure 3.2. In addition to the common architecture, our architecture includes the virtualization layer. The virtualization layer has two primary goals: data access and user access control. Instead of implementing user access in each information usage tool, it is recommendable to implement it in the virtualization layer. The data warehouse layers distinguish the usage and the storage. Separating storage (enterprise data warehouse) from the staging area enables automating transformations creation [P3]. The publish layer enables different interpretation and use for the same data from the enterprise data warehouse. Dimensional modeling is the de facto modeling at the publish layer, or the presentation layer as Hughes [17] call it.

For automating transformations implementation, we need an information model. In publication [P3], we introduced the data map information model for automating transformations from the staging layer to the data vault. In this dissertation, we extend this information model with the addition of SQL join. For the join addition, we added *EntityMapping* and *JoinMapping* to the information model in Figure 3.3. *EntityMapping* stores information of *Entity's* role in *Transformation*. *JoinMapping* describe the *EntityMapping* relation with the attribute level. Our extended information model enables automating implementation from the staging to the data vault and from the data vault to the publish layer. Thus,

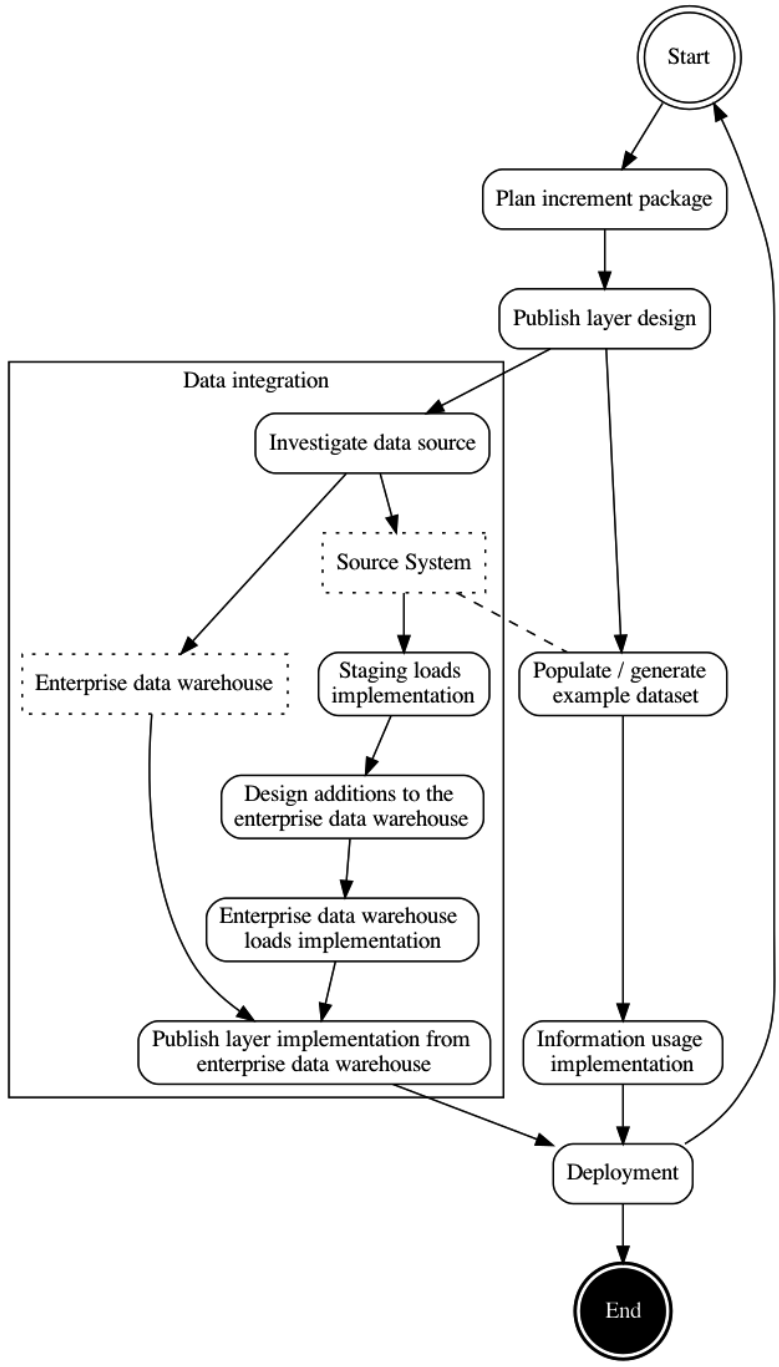

Figure 3.1 Methodology process diagram with data integration presented in detail

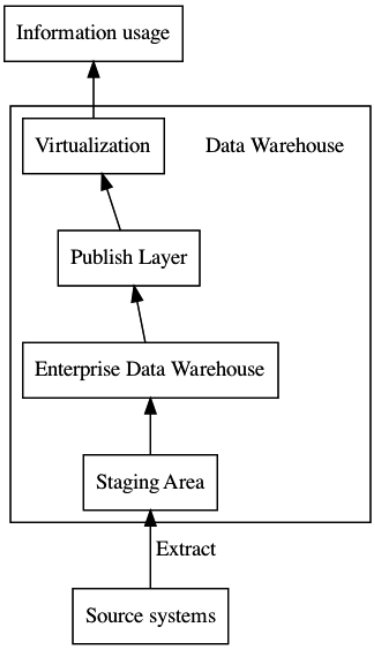

Figure 3.2 Architecture of data warehouse layers for continuous delivery

there is a requirement to implement complex transformations in the business data vault [18, 30]. Our information model is simplified and does not include aggregations such as Kuznetcov et al. [28].

When automating transformations to get out data from a data warehouse, we need an algorithm to deduce the correct database tables. Publication [P4] introduces two schema matching algorithms, the name based and the data flow based mapping. Data flow based schema matching is a novel addition to current schema matching literature. On the other hand, name based mapping is an existing technique in ontology matching. Routine design means using using existing knowledge to organisational problems [16]. The name based mapping is a crucial part of solving the problem of finding correct database tables for automating transformations.

In addition to the presented artefacts, publication [P3] describes how to develop a data warehouse with automated transformations and a proof of concept code generator. Although the code generator is not publicly available, there is a detailed description of the code generator operations. These artefacts have been used successfully in real projects.

We have presented five artefacts in this dissertation for further evaluation and two artefacts to mention as proof of concept artefacts. The process diagram for developing parallel

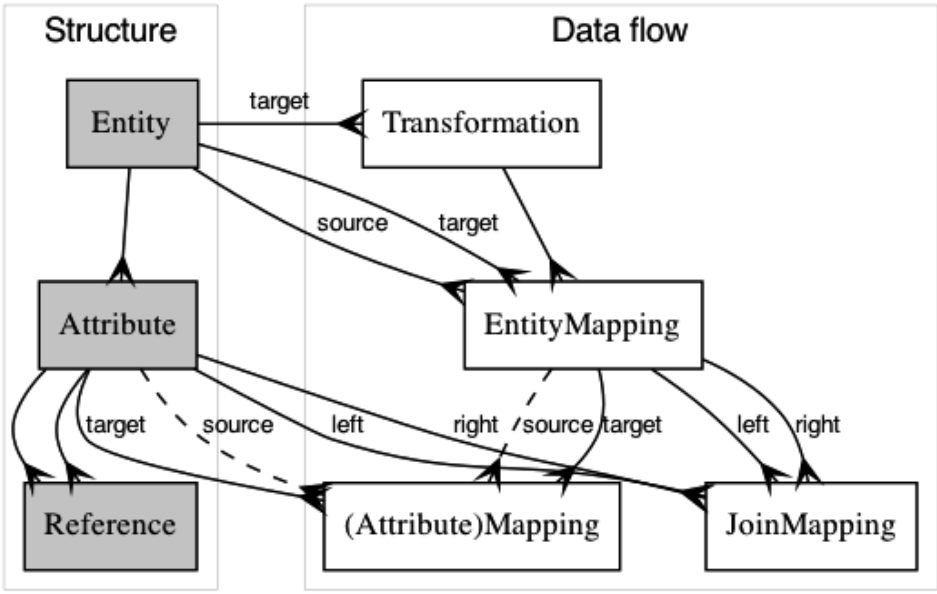

Figure 3.3 Information model for automated transformation implementation

information usage and data integration layers is in Figure 3.1. The architecture diagram of the data warehouse layers for continuous delivery is in Figure 3.2. The information model to generate the transformation implementation is in Figure 3.3. Two algorithms that we use are the attribute names based and the data flow mapping based schema matchings. Next, we evaluate these artefacts in the following sections.

3.1.2 Guideline 2: Problem Relevance

Design science research requires the creation of an innovative, purposeful artefact for a specific problem domain [16]. It is seminal that the problem is researchable and addresses a relevant issue. The problem can be defined as the differences between a goal state and the current state of a system [16]. We compare each artefact with respect to their goal state and a current state.

In publication [P1], we present the current state of report development. Typically, building a business intelligence solution is allocated to two teams. A DW team implements a data model, and loadings into that model. The reporting team takes it forward from the implemented data model with the actual report implementation. Prouza et al. presented a simi-

lar development cycle in 2020, where staging and integration are implemented in iteration *n* and then information and visualisation layers in the iteration *n+1* [46]. The presented methodology is sequential, and it is required to implement all the steps, to get a report to the production environment for end-users. A Sequential workflow extends the implementation time for a report and thus lengthens the feedback cycle from users to developers. When using an agile method like Scrum, a reporting team requires a data model with data. Thus, the implementation for a report requires two sprints: first, one with the DW team developing the data model and data loadings, and second, another sprint where actual report development is done. Methodology suggested by us that allows implementation in parallel with the data integration (DW team) and information usage (report development) is in Figure 3.1. Target state is joint working with DW team and reporting development in the same increment.

Operations involving data warehouses have traditionally been considered tedious and delicate [P2]. The current state follows these operations in the following order: build, deploy, then validate the solution. An alternative model is to apply the principles of continuous software engineering in the domain of data warehousing. There are not many tools for business intelligence that support continuous delivery [7]. Tools used for continuous delivery are less mature than those used for software development [7]. We need an architecture that supports continuous delivery. With the architecture presented in Figure 3.2, we enable continuous delivery in data warehousing and business intelligence. In the target state, development uses continuous delivery order, that is, build, test and deploy; and the architecture supports this order.

In data warehousing, creating the transformations is the most labour intensive work phase. Current tools are proprietary and store the transformations code inside the software without transparency. When using the data vault modelling technique, the number of transformations grows as the data vault uses more tables in a data warehouse [P3]. How to automate implementation when using the data vault modelling technique? Data warehouses that use data vault modelling are becoming more common. There are proprietary tools for automating transformations. Unfortunately, they restrict only specific technologies, and the automation is not transparent. With an information model to automate data vault transformations, we add transparency and enable data vault modelling to be more common.

A typical data warehouse contains hundreds of tables, even thousands of tables. When getting data out, specialists need to know the source data model, which in the case of pub-

lication [P4] is a data vault model. That is a practical problem that, the publication solves. The solution, in this case, is to present a subset of source model entities for specialists. So that the specialist can focus on the implementation of the publish layer and not spend time investigating the source model entities.

These are all practical problems. They ease up the developer's work. When the feedback is more direct, communication between teams is fluent and there is less fragmentation of work. Automating the current manual work from labour intensive work phases reduces the workload and improves quality.

3.1.3 Guideline 3: Design Evaluation

Evaluating a designed artefact for utility, quality, and efficacy that it must demonstrate, is accomplished via evaluation methods in design science [16]. One aim is to shorten the feedback cycle of business intelligence projects. Furthermore, one idea is to execute DW and reporting teams work simultaneously [P1]. As evaluation goes, shortening in the feedback cycle is evaluated with an observational field study [16]. The observation results are not reported publicly nor made publicly available. However, there is a project that used the methodology successfully in the literature [24]. In [24], the implementation was parallel with the data warehouse and the publishing data developers. In addition to this, the parallel working idea used in a project where developed the continuous delivery framework [P2]. Publication [P2] observe that, the first time to start implementation with the *publish layer design* is difficult for a project. After a few iterations, the method is in use. Publication [P2] claim that with the method, the implementation time is shorter and, the developers are more satisfied because they receive direct feedback.

Architecture (see Figure 3.2) is developed in the context of several industrial business intelligence projects [P2]. Architecture follows a common practice where where there is a separation between persistent storage and information usage [7, 19, 21, 26]. Collier presents principles and practices which are not specific to any particular architecture in his agile analytics book [7]. The presented architecture is in Figure 3.2, and practices presented in publication [P2] enable deployments to the data warehouse several times a day. Hence, we can state that the architecture is functional, high quality and efficient.

In data warehousing, creating the transformations is the most labour intensive work phase. Our information model is presented in Figure 3.3; one functional aim is to automate getting data in and out from a data vault data warehouse. Using the information model-based approach can mitigate manual work in the data transformation development [P3].

Transformations from the staging layer to the data vault are readily available in commercial production usage [P3]. However, we introduce the additional part of the information model in this dissertation, thus it has not been used in any real-life project. The additional part enables storing information for SQL joins. The information model SQL joins enable the creation of SQL queries. The created SQL queries construct the getting data out phase implementation.

The phase of getting data out from a data warehouse phase starts by finding the source tables from the data warehouse. In publication [P4], the example is the Northwind database. This descriptive scenario points out that both schema matching algorithms are needed to find correct entities. The presented example in publication [P4] is an experimental simulation that gives the expected results.

3.1.4 Guideline 4: Research Contributions

Design science research holds the potential for three different research contributions: the design artefact, foundations, and methodologies [16]. The design artefact is most often the contribution itself. The creative development of novel constructs, models, methods, or instantiations that extend and improve the existing foundation in the knowledge base, are all possible contributions. Developing design science methodologies may be the contribution too.

The process diagram is a design artefact in Figure 3.1. The architecture in Figure 3.2 is an existing data warehouse architecture that we have proved to be suitable for continuous delivery. The information model in Figure 3.2 is a novel design artefact, which adds transparency in data warehouse automation. Even though the attribute name based schema matching is an existing algorithm, together with the data flow based schema matching, they constitute a novel solution to find correct entities.

3.1.5 Guideline 5: Research Rigor

Design science research requires the application of rigorous methods in both the building and evaluation of a designed artefact [16]. In behavioural science, evaluation of research rigour means assessing data collection and analysis techniques. We have not used behavioural science research methods to collect and analyse the relevance of the artefacts in publications. Real-life projects assess the relevance of artefacts. The artefact is developed based on real-life project team feedback.

Figure 3.1 process diagram tested in the case project in publication [P2]. The architecture in Figure 3.2follows a general data warehouse architecture division where storage and information usage are separated. The information model in Figure 3.2 is used to get data into a data vault. However, we do not have evidence of how it works in real life when generating transformations to get data out from a data vault to a publish layer. Publication [P4] demonstrates schema matching algorithms with the Northwind example.

3.1.6 Guideline 6: Design as a Search Process

Hevner et al. state that design is essentially a search process to discover an effective solution to a problem [16]. Hevner et al. cite to Simon that we can see problem-solving as using means to reach an end while satisfying the laws. Simon goes through different logics to rationalise a decision [52]. A process for seeking a problem solution is a search process. More generally, it is a process for gathering information about problem structure that will be valuable when solving the problem [52].

Publication [P1] gathers information about problem structure by describing the current state. From the current state is derived a problem description. The problem is the long feedback cycle from the customer to the developers. Partly the long feedback cycle is because the implementation happens in sequential phases. As a solution to enable parallel working, the publication proposes dimensional modelling as the first task in a sprint.

A continuous delivery framework for business intelligence states problems in traditional data warehousing without describing those in more detail [P2]. As a solution, it suggests applying the principles of continuous software engineering in the domain of data warehousing.

Publication [P3] introduces that the data vault creates many structurally similar data processing modifications in the transform phase of ETL work. However, it does not describe the problem in more detail. The problem is that the transformations are structurally similar, and if done manually, they are susceptible to human error. A solution is to automate transformation creation. The developed data map tool is implemented based on feedback from developers in real projects.

Publication [P4]explains s the process driven by business requirements. The process starts with designing a dimensional model. In publications [P1] and [P2] the same process is introduced. The current situation requires a specialist who knows the source model creating transformation to the publish layer. The research problem is how to ease up the developers' work. We noted that schema matching techniques are functional. Schema match-

ing based on attribute names performs well when the data vault model has only a limited amount of tables or attribute names do not exist several times. In real-life data warehouses, this is not the case. We investigated more and found a method of using the data flow information as a schema matching technique. By combining both schema matching techniques, we noted that it proposes the correct entities all the time. The combined schema matching algorithm offers correct source entities for transformation and eases the developers work.

As a whole, publications [P1] and [P4] describe the current state and the problem. The proposed solution follows the search process. On the other hand, publications [P2] and [P3] do not follow this search process but directly solve an existing problem.

3.1.7 Guideline 7: Communication of Research

Design science research should present the outcomes to researchers and business audiences. All the publications [P1, P2, P3, P4]have been in scientific conferences. Publications [P2, P3] are presented first in the European – Japanese Conference on Information Modelling and Knowledge Bases (EJC). After the conference, the publications are published in the Series of "Frontiers in Artificial Intelligence" by IOS Press. Publications [P1] and [P3] are part of the Dimecc program Need for Speed [22]. The research work ideas are presented to companies that participated in the program.

The author works for Solita. The author presented research work in internal information-sharing meetings. In particular, the idea of publication [P1] to start with a dimensional model and do parallel work has become widespread in company projects.

This dissertation introduces how reporting and data warehouse teams can synchronously build business intelligence solutions in increments. To use continuous delivery, we need an architecture for business intelligence solution that enables continuous delivery. When we use continuous delivery practices to deploy often, we may boost the development speed by automating transformation creation in data warehousing. Data vault modelling gives us certain principles to automate transformation creation get data into a data warehouse. The same principles allow us to suggest the mappings when getting data out. A human-in-the-loop needs to accept these suggestions and add more complex transformations in a suggested code. The actual implementation code is generated based on the mapping information in the metadata. The generated code includes at least the following advantages no human-made mistakes, the code is similar and contains fewer errors.

The main research contribution from this thesis, is to combine continuous delivery with data warehousing. To make it easier to grasp, we go through each research question. After

each research question, we conclude with a summary of how these research questions into a bigger whole. The research enables deploying new reports and their code to the production several times a day.

Research for this dissertation collaborated with the industry. These ideas have been tested on real projects and thus proven to work. The scientific evaluation uses the design science research method defined by Hevner et al. [16]. Transformations generated with the data map are readily available for commercial production environments [P3]. In the project where the continuous delivery framework for business intelligence was tested, it took less than a day to add an attribute to a report – from the requirement to that it was available in the production environment.

3.2 Contribution to Research Questions

Publication [P1] presents an idea of how two different teams can together build a business intelligence solution in increments. It presents one solution to research question 1:How to implement a business intelligence solution in increments jointly with reporting and data warehousing teams? Publication [P2] introduces the architecture and a methodology for using continuous delivery in business intelligence. The architecture is a solution to research question 2: What is suitable architecture for a business intelligence solution to enable continuous delivery?

Publication [P3] presents how to automate getting data into a (data vault) data warehouse, which is research question 3. When automating getting data out from a data vault data warehouse (RQ4), we need the metadatabase from publication [P3].

How publications contribute to the research questions is presented in Table 3.1. There is one publication for each research question in questions 1-3. To research question 4, publications [P3] and [P4] contributed.

3.2.1 Shorten the Report Feedback Cycle

In the scrum, each development sprint has a planning step. In the planning phase, the team creates a sprint backlog. They list sprint items and map them to product packets. The reporting team picks reporting requirements that are available in a database, which the data warehouse team has already implemented. There is a dependency between implementation from the data warehouse team, and the reporting team. Due to this dependency affecting report creation, including database and report development, two sprints are needed to ship a

Publication	P1	P2	P3	P4
Research question				
RQ1	X			
RQ2		X		
RQ3			X	
RQ4			X	X

Table 3.1 Publications contribution to research questions.

reporting feature. Furthermore, it is possible to get customer feedback only after the report exists. All this creates the fragmentation of work and a long feedback cycle.

Report development has the requirement that data exists in the database. Report development uses a metadata layer. A metadata layer is created based on a data source definition for the reporting. Report development requires a database connection where there is a data model. The data model used in reporting follows dimensional modelling. Publication [P1] proposes to create the dimensional model based on a conceptual model that contains information on the source systems. The conceptual model presents the associations between dimensional model entities, whereas a source system specification presents the attributes and their data types. In addition to the data model, it needs sample data for helping the report visualisations. For this purpose, sample data for the data model needs to be generated.

Based on a data model created at the beginning of a sprint, both teams can work in parallel. With this action, teams can build together business intelligence solutions in increments (RQ1). For the actual report use, the data must be in a database. Both the report and data warehouse teams need to finalise their work. The parallel working makes it easier for the teams to communicate since they concentrate on the same main goal, finishing the report in one sprint. A report user can give feedback to both teams directly after the sprint. The feedback is relevant when it is received directly and without delay. A user involved in report development waits the implementation time to get the report available. Parallel work shortens implementation time from two sprints to one sprint. The user involved in report development is more likely to participate since there is no delay in the process. Working in parallel shortens the feedback cycle, improves communication, and the work is not fragmented.

3.2.2 Business Intelligence Architecture for Continuous Delivery

In a typical business intelligence solution, developers use shared environments. The shared environments create a dependency chain to manage development activities and deployments to different stages. When an implementation item is in a specific stage, that stage is reserved until the implementation is verified. Reserving stage leads to deploying broader implementation items. Broader implementation items increase the needed verification time. Thus the deployments are made less frequently. The shared environments reduce the number of deployments to production, which is contrary to continuous delivery. In publication [P2] we present a business intelligence architecture where the infrastructure as code approach creates environments for specific purposes. The specific purpose may be a feature specific environment. Using feature specific environments removes scheduling and dependency related constraints from development.

The infrastructure as code automates building a new environment. Continuous delivery practice automates build, test, and deployment processes. Dividing the architecture into different layers makes testing easier. Most importantly, the different architecture layers enable the automation of deployments. In publication [P2] and Figure 3.2 we present a layers: source systems, data warehouse, and information usage. Inside the data warehouse, there are layers: the staging area, enterprise data warehouse, publish layer, and virtualization.

Publication [P2] presents workflow where developer-specific environments are created with automation. The presented architecture supports dividing the implementation into independent increment packages. The packages can be tested and deployed separately.

Suitable architecture for continuous delivery enables automated build, test, and deployments. In the context of business intelligence, this means that architecture is divided into layers that support automation, see Figure 3.2. The available tools for business intelligence limit the architecture layers. The division with layers is to distinguish between usage and storage. The separation occurs in the architecture presented in Figure 3.2. How the storage, the data warehouse, is subdivided depends on the tools used.

Publication [P2] presents business intelligence architecture that enables continuous delivery (RQ2). Presented architecture and practices enable deployments to the data warehouse several times a day.

3.2.3 Reducing Manual Work with Automation

The most laborious phase of building a data warehouse is to create ETL loads. We follow the model first approach, where a data warehouse model is designed based on business requirements and available source data. Publication [P3] presents a data map tool for automating transformation creation. The first step is to populate the structure information to the metadata. The tool can interpret structure information from a database information schema. The next step is to populate data flow information to the metadata. The tool can assist humans by suggesting data flow mappings, but creating the actual mappings between attributes is human work. The tool can generate actual ETL loads code based on structure rules and data flow information. The generating code relies on rules on how different data vault entities are populated.

The data map tool works like a charm when following these. An enterprise data warehouse modelling follows data vault rules. Secondly, a data vault satellite table uses the same attribute names as the source interface. From each source system interface a staging area table is created. Based on these staging tables actual mappings are done. Data flow mappings are done one staging table at a time. The input data flow information starts by creating a load between two entities. The source entity is a staging table, and the target entity is an enterprise data warehouse table. The data map tool can suggest appropriate attributes based on attribute names. The only attributes that need manual mappings are between a hub table and a staging table. In the hub table there are attributes that describe the business. Usually, these differ from how the names are in an interface. The data map tool automates the implementation of getting data into a data vault (RQ3).

With this approach, the needed manual work is reduced but not completely eliminated. By reducing manual work, we also reduce the possibilities for human errors. Generated transformations for ETL loads are less error-prone than manually written code. Transformations generated with the data map are readily available for commercial production [P3].

3.2.4 Automating Getting Data Out from an EDW

Getting data out from an EDW is an inverse operation to getting it into the EDW. Data vault modelling splits data into multiple tables, from which they need to be combined into a single data set. An EDW that has been in use for a long time contains hundreds or thousands of entities. The first challenge is to find the correct entities from the big data model. Publication [P4] introduces two different schema matching algorithms. The combined al-

gorithm finds the correct entities from the data vault database.

The data vault tables can be joined automatically, based on rules and data model structure. However, there are cases when there are several possibilities and a human needs to assist automation. Suggested joins are base for adding more complex join information. For automation, joins must be stored in a metadatabase.

When the necessary data for the publish layer exists in the EDW layer, the presented schema matching algorithms find the correct entities, and the joins are trivial. However, this requires the data modelled in the EDW layer at a suitable granularity. If the needed transformations are complex, it has to be done by a human. Therefore, we recommend implementing the conversions in the data vault layer in the so-called business data vault.

By following the process driven by business requirements and starting designing a publish layer model with sample data as presented in publication [P1]. Then implement transformations from the staging layer to the EDW [P3]. Then there is data flow and structure information as described in publication [P4]. The schema matching algorithms find correct entities. In addition, based on source entities a source query is suggested. Thus, getting data from an EDW can be automated, and implementation code generated (RQ_4).

3.3 Results Summary

At the beginning of the research work, the development of a report took a long time. Each research question shortens the time used in report development. The first improvement is to organise the work of the reporting and data warehouse teams in parallel with the same increment in the scrum as a sprint. With the interface specifications and the conceptual model, it is possible to create the dimensional model used in reporting [P1]. Creating a dimensional model at the beginning of an increment, both teams can work in parallel. In addition to the dimensional model, a database needs to populate with generated data for report developers. Creating a database with generated data for report developers, both reporting and data warehousing teams can jointly build the business intelligence solution in increments. It is a solution to research question 1 (RQ_1). When teams are working together, communication is more efficient. Both teams receive feedback from the report users directly after the development work.

When teams started to work parallel in the same increment, they added several reports to the backlog. Multiple report development parallel benefits by creating feature specific environments. Also, testing is efficient when each report is available in a feature-specific

environment. The feature-specific environments creation uses the infrastructure as code approach. Automating building environments is one continuous delivery practice. Also, regarding continuous delivery, the testing and deployment should be possible to be automated. Automating build, test, and deployment processes are prerequisites for architecture in research question 2 (RQ2). The business intelligence architecture contains different layers in publication [P2]. The architecture layers separate storing data to the data warehouse, and information usage. Separation supports current business intelligence tools. Reporting tools are different from database modelling and ETL tools. In addition, the data warehouse contains layers staging area, enterprise data warehouse, publish layer, and virtualization. The layers aim to separate the transformations into their respective stages. The presented architecture enables building feature specific environments for executing testing before deployment. Hence, the architecture presented enables continuous delivery and meets the requirements of research question 2 (RQ2). In practice, the architecture works for continuous delivery.

In the architecture, the enterprise data warehouse uses data vault methodology. The data vault normalises the model using different entity types. Because of this, the data vault database has lots of tables. There are a large number of transformations between the tables. However, a single transformation is very straightforward to implement. A straightforward task that is often repeated is to be automated, at least to take advantage of using templates [19]. Publication [P3] introduces the answer to research question 3 (RQ3), how to automate getting data into an enterprise data warehouse. Data vault modelling increases the number of tables but reduces the time spent on implementation work. Furthermore, generated ETL loads are less error-prone than manually written code.

Getting data into an EDW is widely a well automated task. However, getting data out from an EDW is not usually automated. Getting data out often involves complex transformations and consists of several tables. Following [18] we suggest breaking down complex transformations as business data vault entities. Then we may use two presented schema matching algorithms to choose desired entities from the data vault. From data vault tables, the schema matching algorithm finds desired entities. The joins between desired entities are suggested based on data vault rules. A human adds more complex additions to the joins and calculations if needed. Transformations from the data vault are generated based on metadata information. Automating the implementation of getting data out from an EDW is research question 4 (RQ4). As for getting data out from an EDW, it is essential to implement this by generating respective code. The generating code includes the following advantages:

no human-made mistakes, code follows a unified style, fewer errors.

Research questions 3 and 4 focus on automating getting data into EDW and out from EDW. We have shown how to automate the implementation. The idea is to split the implementation into simple transformations that are straightforward to implement, then compile them to form a larger whole ETL load chain. In practical work, we have seen difficulty for developers to understand how automation works. The automation tool is like a black box that is doing magic stuff. When the developer creates the simple transformations manually and compiles them to a larger whole, the developer understands how automation works.

In research question 1, we created a dimensional model for enabling report development. Then with the model first approach, we modelled the enterprise data warehouse. After this, we have the data models structures in all layers, the staging, the EDW and the publish layer. Creating transformations between layers is creating data flow information between models. We introduced an information model for storing structure and data flow information in publication [P3]. We have extended this information model in Figure 3.3 to store association information for generating SQL joins and conditions used in joins. Implementation code for transformations is generated based on data model structure and data flow information. Automating data warehouse transformations creation save time from development efforts. More importantly, generated implementation code has fewer errors and fewer mistakes and is easier to maintain. The code generation allows us to split the deployment into independent deployment packages. The independent deployment packages are ready to deploy in different environments. Deployment allows us to adhere to continuous delivery. In continuous delivery in data warehousing, focus is on deliverables and customer communication and not on technical ETL loads. Of course, technical ETL loads need to be in shape, but the focus is on deliverables.

At the beginning of the research the development of a report took a long time because the developers were in different teams and had a dependency on each other. In research question 1 (RQ1) the first action is to break this dependency and allow parallel working. The simultaneous development of multiple reports is possible using feature-specific environments. The automation of the development environment was the first step of the continuous delivery approach. Research question 2 (RQ2) found an architecture that enables continuous delivery. Together, research questions 1 and 2 allow the parallel development of several reports and their deployment to use. Research question 3 (RQ3) concentrated on the automating of getting data into an EDW. Creating ETL loads is the most laborious

phase when building a data warehouse. We introduced an information model for storing structure and data flow information. Based on the information model, we generate the implementation code for ETL loads. Generating implementation code enables us to bundle several transformations as a single deployment package. Deployment packages are easy to deploy in different environments introduced in research question 2 (RQ2). Getting data out from an EDW is still often manual work. Research question 4 concentrated on how to mitigate this manual work. We introduced two different schema matching algorithms to support creating transformations. With the schema matching algorithms and complex logic implemented in the business data vault, needed data flow information is a straightforward task. Thus getting data out from an EDW can be automated. Yet in addition, getting data into the EDW and getting data out from the EDW is implemented by generating the code as deployment packages. Presented architecture and continuous delivery model enable deploying new reports and their code to production use several times a day. The change is staggering, from deployment once in an increment to deploying daily to production. Note that in the beginning, developing the report took two increments, currently only a few days. This change has happened in several customer projects, and their data warehouses are in production use.

4 DISCUSSION

In theoretical implications, we discuss how the research expands current knowledge. Practical implications viewpoints are report users and their organisations as business intelligence developers and consulting companies. Research validity focuses on the quality of scientific research and the dependability of the research findings. Research focuses on continuous delivery in data warehousing. The next step is to add analytical tools in the deployment pipeline and offer an end to end business intelligence solution with continuous delivery. When all the phases are automated, we can move from continuous delivery to continuous deployment.

4.1 Theoretical implications

Data warehousing has been traditionally considered tedious and delicate [21]. Distinct steps take place one after another in a predefined unalterable sequence. Especially the sequence begins with data integration steps followed by the report implementation steps [46]. Our approach changes this sequence, starting from the publish layer design, see Figure3.1. Teorey et al. present a similar workflow where dimensional modelling is the first step in the data issues thread, and implementation is parallel with data issues and analytical implementation [56]. It is important to note that Teorey et al.'s workflow ends with the deployment step. In the traditional plan-only approach designing and implementing the database is done before the analytical tool implementation work starts. In the worst case, the reporting (BI) team is separate from the data warehouse (EDW) team [50].

Many have voiced the opinion that data projects are hard to manage with agile methodologies [50]. Despite that, agile development is used in business intelligence projects and will be used more and more in the future. Regarding survey, DW/BIA projects use an agile plan for 68.5%, and 31.5% use an agile heavy [3]. Although development is in agile increments, deployments often do not follow the same increments as development [46]. Each development increment should create a new release, which is possible to deploy in production use!

Collier [7] says that a user valued functionality is "Done" when it is of production quality, it is integrated into the system and developers are proud of their work. Unfortunately, Collier and we are in the minority.

There are not many tools for business intelligence that support continuous delivery [7]. Tools used are less mature for continuous delivery than those used for software development [7]. When there is a lack of repeatable and reliable ways of deploying to production, it leads to scheduled service interruptions. Depending on the frequency of service interruptions, a deployment may contain several items, and the problem is how to verify that everything will go smoothly. Continuous delivery [20] is quite the opposite, releasing software with a repeatable and reliable process. Continuous delivery in business intelligence needs an architecture that supports it. One option is presented architecture in Figure 3.2. This architecture follows a common practice where persistent storage is separated from information usage [7, 19, 21, 26]. According to Collier, principles and practices are not specific to any particular architecture [7]. However, we note that current tools support architecture where storage separates from usage. The architecture in Figure 3.2 supports continuous delivery with current tools.

In data warehousing, creating the transformations is the most labour intensive work phase. We introduce an information model for structure and dataflow information in Figure 3.3. Our information model is not unique or a novelty as an information model for the metadatabase [28]. Data warehouse automation, a solution to automate transformation generation, is a novel addition in scientific literature. Modelling (design) phase automation is widely studied [5, 12, 13, 23, 27, 40]. Although these methods automate the design phase, they require human intervention. The information model (see Figure 3.3) enabless automating implementation from the staging to the data vault and from the data vault to the publish layer. Thus, there is a requirement to implement complex transformations in the so-called business data vault. [18, 30]. Our information model is simplified and does not include aggregations such as Kuznetcov et al. [28]. Our information model adds transparency to automate transformation creation to a data warehouse that uses data vault methodology compared to proprietary solutions.

Compared to other methods, the data vault technique uses more tables. A typical data vault data warehouse contains hundreds of tables or even thousands. When getting data out, specialists need to know the source data model. To solve this problem, we use a combination of two schema matching algorithms. An experimental simulation with the Northwind database gives the expected results. The combination schema matcher approach re-

duces the time needed for implementation.

4.2 Practical implication

This research focuses on shortening cycle time, from idea to a report available in production. A report user benefits from the reduced cycle time. Beginning of the research, it took a long time from an idea to a published report. Users did not want to be involved in the development of reports because of the long-lasting timeline. Our methodology starts the development from the conceptual model and reporting requirements. The user involved in the development is actively participating in the report development. The report is ready in one increment – usually within two weeks. Continuous delivery practice enables releasing reports often for production use. We have noticed that users want to use new reports on specific release dates, rather than daily.

Organisations that develop their business intelligence solutions benefit in several ways from our approach. The generated code includes the following advantages: no human-made mistakes, the code is more unified, and code contains fewer errors. Production releases are done in a reliable and repeatable way. Every release has passed several tests before its deployment to production. Tests include acceptance testing, capacity testing and optional manual testing. Reliable production deployment is key to the transition from scheduled service interruptions to frequent production deployments. Shortened cycle time allows an organisation to try things out in production use. With continuous delivery, organisations are moving to shorter release dates. Besides releases, some reports are deployed to production immediately. The biggest benefit is in maintenance. The generated code and data warehouse principles produce robust implementations. However, a robust implementation does not fix data issues. There exist techniques to raise these to data stewards. A case company that had a weekly error situation in its ETL processes, after switching to data warehouse implemented with the data vault and generated ETL code, only three error situations occurred during a one year span. The same change in reducing error situations happened based on developers' feedback in another project where a data map tool was used to create the ETL code.

Business intelligence developers benefit from joint working for the same report on data integration and reporting implementation. They receive feedback from report users directly after the development. When the feedback is relevant, developers are satisfied. Data warehouse automation reduces the number of manually laborious tasks. Developers can focus

on value-adding work, which increases job satisfaction. The generated code is more unified and contains fewer errors. Consequently data problems or maintenance tasks are rare, and developers can focus on new development tasks. Deployment to production takes executing a couple of scripts from your computer rather than a stress-filled weekend. All this does not come free for developers, of course. Developers need to commit to continuous delivery principles. They need to switch their tools to those that support version control and enable automation through scripting. They need to create testing practices and build the automated deployment pipeline. It is all difficult but worth it.

Automation reduces work and thus hourly billing. Perhaps companies offering business intelligence consultation do not want to adopt automation. Automation allows the construction of a product or service, which increases billing. A client used the data map in real project with a monthly fee. There are many proprietary products available for data warehouse automation; see Appendix A. Our research is an exception in the market, openly publishing how to automate data warehousing. There is a demand in the market for data vault data warehouses. No one implements a data vault data warehouse without any automation. Developers are satisfied when they produce high quality (generated) code and are able to do production deployments during office hours. Satisfied developers do quality work, and customers communicate it forward, increasing demand and enabling growth.

To emphasize the cumulative practical importance, we interviewed a customer about a project where the reference framework described in publication [P2] was tested. The customer has a stable business with five domains. In 2015, they invested 2.5 million euros in the development of the data warehouse and did not complete it during the year. We were invited to test our approach the following year in 2016. Using the methodology described in publication [P2], the data warehouse and reports were completed in six months. Addition to targeted five domains, we also implemented sales management reporting. Our billing was 0,5 million euros, while it was 2,5 million euros a year earlier. Although these numbers are significant, they do not tell the whole truth. We asked how long it would take to add one new attribute and it was done in fifteen minutes!

4.3 Research validity

Research validity focuses on the quality of scientific research and the dependability of the research findings. Research validity has two parts, internal and external [39]. Internal validity evaluates research credibility. External validity evaluates the transferability of research re-

sults. Understanding issues of internal and external validity in experiments, helps us understand the broader problems of causal inference, generalizability and validity, in construction and other types of research [1] . Discovery is the process of generating or proposing scientific claims. Justification includes activities of testing such claims for validity [37]. Natural or field experiments have lower internal validity but higher external validity [1]. The results are more broadly generalisable to the real world. Design science focuses on assessing the efficacy and utility of the developed artefacts [29]. We have followed Hevner et al. guidelines for evaluating the developed artefacts in Chapter 3. This section evaluates the research results in their internal and external validity. According to Larsen et al., validity has been underutilized to strengthen the claims of design science research [29].

When creating a data model used in reporting at the beginning of a sprint, reporting and data warehouse teams can work in parallel to achieve a shared goal. We followed the general design cycle [57, Figure 3]. Publication [P1] describes the current state, the problem, the suggestion, the artefact and the results. There is a lack of testing the artefact with a larger population. The result is transferable, as evidenced by Jussila et al. [24]. The result generalises to projects where there are data transfers between two different technologies.

In Figure 3.2 we present a business intelligence architecture that enables continuous delivery. This architecture follows a common practice, where persistent storage separates from information usage [7, 19, 21, 26]. Because the scientific literature presents a similar architecture, we can consider the architecture to be internally reliable. According to Collier [7], principles and practices are not specific to any particular architecture. There may be other reasons why this architecture works with continuous delivery other than the architecture itself. The architecture has proven to work in a real-life project, thus raising the external validity. Architecture is suitable for business intelligence and is not transferable. The idea of separating storage from usage is transferable to other domains.

With the data map tool, we have automated implementation getting data into a data vault. Publication [P3] explains principles for automation, as well as the information model and the workflow when using such automation. The internal validity is high quality in automating getting data into a data vault. The external validity is proven in real projects as the transformations generated with the data map are readily available for commercial production. However, there is no transferability, as the automation relies heavily upon the data vault principles.

Automating getting data out from a data vault has three steps. First, finding correct source entities for given target entities with the schema matching algorithms introduced

in publication [P4]. Secondly, creating the transformations based on suggestions, and allowing humans to add complex parts. The last step is generating the implementation code for transformations. The first step is explained in detail in publication [P4]. However, two other steps are briefly explained in this dissertation. Only an expert can implement automation, so the internal validity is weak. Publication [P4] contains a controlled experiment with the Northwind database. There is no real-life case project of the automation getting data out. Described automation for getting data out is not transferable as it needs the data vault model to the source database.

4.4 Recommendations for further research

Our research focuses on continuous delivery in data warehousing. The next step is to add analytical tools in the deployment pipeline and to offer the business intelligence solution with continuous delivery. As the data warehouse integrates the business information in one place, in future, this integrated information should be available for machine use with API interfaces.

Continuous delivery has three phases: the commit stage, the testing, and the release (see Figure 2.1). Our focus is on the commit stage, building the increment package. We also focus on how a release should be deployed to production. Testing is an important part, although we have not addressed it in this dissertation. When all the phases are automated, we can move from continuous delivery to continuous deployment.

5 SUMMARY

The unifying research question in this dissertation is: how to shorten the time from a reporting idea until it is available for users? Shortening development time has obvious business value, as it enables the business to take new reporting functionalities into use more frequently. The research focuses on shortening the development process and automating data warehouse implementation.

The main improvement in the development process is gained from applying continuous delivery to data warehousing. Collier [7] introduced continuous delivery practices to business intelligence and data warehousing. However, they are not used widely in data warehousing. Traditionally, when developing a business intelligence solution, a database is implemented before the implementation of reporting begins. The current study suggests to change this sequential process to parallel. Assuming that a database implementation is most of the work needed, parallel working shortens the duration of development by the amount of reporting implementation time. Furthermore, the reporting developers and data warehouse developers focus on the same goal, making it easier for the developers to communicate with each other. After releasing a report, reporting and data warehouse teams get direct feedback from reporting users. Another way to shorten the development time frame is automating implementation work. Data warehouse automation is divided into two parts: getting data into a data warehouse and getting data out of it. This dissertation presents an information model for automating transformations in a data warehouse. The presented information model enables automating the implementation from the staging to the data vault and from the data vault to the publish layer.

The research has clear benefits in practical solutions. A notable change is joint working with reporting and data warehousing teams. Data warehouse automation improved development efficiency. Automation removed manual tasks, improved the quality, and decreased time used in development. Continuous delivery creates a reliable process that enables frequent production releases. These together can increase the frequency of releasing new business intelligence solution functionalities from months to weeks or even to days.

REFERENCES

[1] Deborah A Abowitz and T Michael Toole. "Mixed method research: Fundamental issues of design, validity, and reliability in construction research". In: *Journal of construction engineering and management* 136.1 (2010), pp. 108–116.

[2] Moh'd Alsqour, Kamal Matouk, and Mieczysław L. Owoc. "A survey of data warehouse architectures — Preliminary results". In: *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE. 2012, pp. 1121–1126.

[3] Dinesh Batra. "Agile values or plan-driven aspects: Which factor contributes more toward the success of data warehousing, business intelligence, and analytics project development?" In: *Journal of Systems and Software* 146 (2018), pp. 249–262. DOI: 10.1016/j.jss.2018.09.081.

[4] G Blažić, P Poščić, and D Jakšić. "Data warehouse architecture classification". In: *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE. 2017, pp. 1491–1495. ISBN: 978-953-233-090-8. DOI: 10.23919/MIPRO.2017.7973657.

[5] Luca Cabibbo and Riccardo Torlone. "A logical approach to multidimensional databases". In: *International Conference on Extending Database Technology*. Springer. 1998, pp. 183–197. DOI: 10.1007/BFb0100985.

[6] Samir Chatterjee and Alan R Hevner. *Design research in information systems: theory and practice*. Springer, 2010.

[7] Ken Collier. *Agile analytics: A value-driven approach to business intelligence and data warehousing*. Addison-Wesley, 2012.

[8] J Fernández, E Mayol, and JA Pastor. "Agile approach to business intelligence as a way to success". In: *Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications*. IGI Global, 2012, pp. 132–160. DOI: 10.4018/978-1-61350-050-7.ch007.

[9] Brian Fitzgerald and Klaas-Jan Stol. "Continuous software engineering: A roadmap and agenda". In: *Journal of Systems and Software* 123 (2017), pp. 176–189. DOI: 10.1016/j.jss.2015.06.063.

[10] Tobias Freudenreich, Pedro Furtado, Christian Koncilia, Maik Thiele, Florian Waas, and Robert Wrembel. "An on-demand ELT architecture for real-time BI". In: *International Workshop on Business Intelligence for the Real-Time Enterprise*. Springer. 2012, pp. 50–59. DOI: 10.1007/978-3-642-39872-8_4.

[11] *From framework to product*. URL: https://vaultspeed.com/about-us/ (visited on 11/04/2021).

[12] Matteo Golfarelli, Simone Graziani, and Stefano Rizzi. "Starry vault: Automating multidimensional modeling from data vaults". In: *East European Conference on Advances in Databases and Information Systems*. Springer. 2016, pp. 137–151. DOI: 10.1007/978-3-319-44039-2_10.

[13] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. "Conceptual design of data warehouses from E/R schemes". In: *Proceedings of the thirty-first Hawaii international conference on system sciences*. Vol. 7. IEEE. 1998, pp. 334–343. DOI: 10.1109/HICSS.1998.649228.

[14] Matteo Golfarelli and Stefano Rizzi. *Data warehouse design: Modern principles and methodologies*. McGraw-Hill, Inc., 2009.

[15] Aymeric Hemon-Hildgen, Frantz Rowe, and Laetitia Monnier-Senicourt. "Orchestrating automation and sharing in DevOps teams: a revelatory case of job satisfaction factors, risk and work conditions". In: *European Journal of Information Systems* 29.5 (2020), pp. 474–499. DOI: 10.1080/0960085X.2020.1782276.

[16] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. "Design science in information systems research". In: *MIS quarterly* (2004), pp. 75–105.

[17] Ralph Hughes. *Agile Data Warehousing for the Enterprise*. Elsevier, 2016. ISBN: 978-0-12-396464-9.

[18] Hans Hultgren. *Data vault modeling guide*. 2019. URL: http://dvstandards.com/wp-content/uploads/2021/02/data_vault_modeling_guide_2019_v3.pdf (visited on 12/07/2021).

[19] Hans Hultgren. *Modeling the agile data warehouse with data vault*. New Hamilton, 2012. ISBN: 978-0615723082.

[20] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Fourth Printing. Pearson Education, May 2011.

[21] William H Inmon. *Building the data warehouse*. Third Edition. John Wiley & Sons, Inc., 2002. ISBN: 0-471-08130-2.

[22] Janne Järvinen, Tua Huomo, Tommi Mikkonen, and Pasi Tyrväinen. "From agile software development to mercury business". In: *International Conference of Software Business*. Springer. 2014, pp. 58–71. ISBN: 978-3-319-08738-2. DOI: 10.1007/978-3-319-08738-2_5.

[23] Vladan Jovanovic and Ivan Bojicic. "Conceptual data vault model". In: *SAIS Conference, Atlanta, Georgia*. Vol. 23. Mar. 2012, pp. 1–6. URL: https://aisel.aisnet.org/sais2012/22.

[24] Jari Jussila, Timo Lehtonen, Jari Laitinen, Markus Makkonen, and Lauri Frank. "Visualising maritime vessel open data for better situational awareness in ice conditions". In: *Proceedings of the 22nd International Academic Mindtrek Conference*. 2018, pp. 92–99. DOI: 10.1145/3275116.3275124.

[25] Ralph Kimball and Joe Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming and Delivering Data*. John Wiley & Sons, Inc., 2004.

[26] Ralph Kimball and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons, 2011.

[27] Dragoljub Krneta, Vladan Jovanovic, and Z Marjanovic. "An Approach to Data Mart Design from a Data Vault". In: *INFOTEH-Jahorina BiH* 15 (2016).

[28] Yevgeni Kuznetcov, Maxim Fomin, and Andrei Vinogradov. "Multidimensional Information Systems Metadata Repository Development with a Data Warehouse Structure Using" Data Vault" Methodology". In: *Proceedings of the XI International Scientific Conference Communicative Strategies of the Information Society*. 2019, pp. 1–5. DOI: 10.1145/3373722.3373777.

[29] Kai R. Larsen, Roman Lukyanenko, Roland M Mueller, Veda C Storey, Debra VanderMeer, Jeffrey Parsons, and Dirk S Hovorka. "Validity in Design Science Research". In: *International Conference on Design Science Research in Information Systems and*

Technology. Lecture Notes in Computer Science. Springer. 2020, pp. 272–282. DOI: 10.1007/978-3-030-64823-7_25.

[30] Dan Lindstedt and Kent Graziano. *Super Charge Your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault*. CreateSpace, 2011. ISBN: 978-1463778682.

[31] Dan Linstedt. *Data Vault Data Modeling Specification v2.0.2*. Data vault alliance. 2018. URL: https://danlinstedt.com/wp-content/uploads/2018/06/DVModelingSpecs2-0-1.pdf (visited on 12/08/2021).

[32] Dan Linstedt. "Data Vault Series 1–Data Vault Overview". In: *The Data Administration Newsletter* (2002).

[33] Dan Linstedt. *DV Modeling Specification v1.0.9*. 2010. URL: https://danlinstedt.com/allposts/datavaultcat/standards/dv-modeling-specification-v1-0-8/ (visited on 12/07/2021).

[34] Dan Linstedt. *RapidACE – Going Open Source*. July 2011. URL: https://danlinstedt.com/allposts/datavaultcat/rapidace-going-open-source/ (visited on 11/04/2021).

[35] Dan Linstedt, Kent Graziano, and Hans Hultgren. *The New Business Supermodel, The Business of Data Vault modeling*. Second Edition. Aug. 2009. ISBN: 978-1435719149.

[36] Dan Linstedt and Michael Olschimke. *Building a scalable data warehouse with data vault 2.0*. Morgan Kaufmann, 2016. ISBN: 978-0-12-802510-9.

[37] Salvatore T March and Gerald F Smith. "Design and natural science research on information technology". In: *Decision support systems* 15.4 (1995), pp. 251–266. DOI: 10.1016/0167-9236(94)00041-2.

[38] Tommi Mikkonen, Casper Lassenius, Tomi Männistö, Markku Oivo, and Janne Järvinen. "Continuous and collaborative technology transfer: Software engineering research with real-time industry impact". In: *Information and Software Technology* 95 (2018), pp. 34–45. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2017.10.013. URL: https://www.sciencedirect.com/science/article/pii/S0950584917304007.

[39] Haradhan Kumar Mohajan. "Two criteria for good measurements in research: Validity and reliability". In: vol. 17. 4. 2017, pp. 59–82. DOI: 10.26458/1746.

[40] Daniel L Moody and Mark AR Kortink. "From enterprise models to dimensional models: a methodology for data warehouse and data mart design." In: *DMDW*. 2000, p. 5.

[41] Solomon Negash. "Business Intelligence". In: *Communications of the Association for Information Systems* 13.15 (2004), pp. 175–193. DOI: 10.17705/1CAIS.01315. URL: https://aisel.aisnet.org/cais/vol13/iss1/15.

[42] Jay F Nunamaker Jr, Minder Chen, and Titus DM Purdin. "Systems Development in Information Systems Research". In: *Journal of Management Information Systems* 7.3 (1990), pp. 89–106. URL: https://www.jstor.org/stable/40397957.

[43] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. "A design science research methodology for information systems research". In: *Journal of management information systems* 24.3 (2007), pp. 45–77. DOI: 10.2753/MIS0742-1222240302.

[44] Virpi Pirttimäki. "Business intelligence as a managerial tool in large Finnish companies". In: (2007).

[45] Mary Poppendieck and Thomas David Poppendieck. *Implementing lean software development: from concept to cash*. 8th Printing, April 2010. Pearson Education, 2007.

[46] M. Prouza, S. Brodinová, and A.M. Tjoa. "Towards an Agile Framework for Business Intelligence Projects". In: *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*. IEEE. 2020, pp. 1280–1285. DOI: 10.23919/MIPRO48935.2020.9245166.

[47] Vincent Rainardi. *Building a data warehouse: with examples in SQL Server*. 1st ed. Apress, 2008. ISBN: 978-1-4302-0528-9. DOI: 10.1007/978-1-4302-0528-9.

[48] Vincent Rainardi. "Testing Your Data Warehouse". In: *Building a Data Warehouse*. 1st ed. Apress, 2008. Chap. 16, pp. 477–489. ISBN: 978-1-4302-0528-9. DOI: 10.1007/978-1-4302-0528-9_16.

[49] Mohammad Rifaie, Keivan Kianmehr, Reda Alhajj, and Mick J Ridley. "Data warehouse architecture and design". In: *2008 IEEE International Conference on Information Reuse and Integration*. IEEE. 2008, pp. 58–63. DOI: 10.1109/IRI.2008.4583005.

[50] Deedar Shamsi. "Comparative study of Agile Business Intelligence and Agile Data Warehouse". In: *Global Sci-Tech* 8.1 (2016), pp. 37–49.

[51] Alec Sharp and Patrick McDermott. *Workflow modeling: tools for process improvement and applications development*. Second Edition. Artech House, 2009. ISBN: 978-1-59693-192-3.

[52] Herbert Alexander Simon. *The Sciences of the Artificial*. Third Edition. MIT press, 1996. ISBN: Electronic: 9780585360102, Paperback: 9780262193740.

[53] Graeme Simsion and Graham Witt. *Data modeling essentials*. Third Edition. Elsevier, 2005. ISBN: 0-12-644551-6.

[54] Ajit Singh and Sultan Ahmad. "Architecture of data lake". In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* 5.2 (2019), pp. 4–4.

[55] Hideaki Takeda, Paul Veerkamp, and Hiroyuki Yoshikawa. "Modeling design process". In: *AI magazine* 11.4 (1990), pp. 37–48.

[56] Toby J Teorey, Sam S Lightstone, Tom Nadeau, and HV Jagadish. *Database modeling and design: logical design*. Fifth Edition. Morgan Kaufmann, Feb. 2011. ISBN: Electronic: 9780123820211, Paperback: 9780123820204.

[57] Vijay Vaishnavi, William Kuechler, and Stacie Petter. "(Eds.) (2004/19). "Design Science Research in Information Systems" January 20, 2004 (created in 2004 and updated until 2015 by V. Vaishnavi and W. Kuechler.); last updated (by V. Vaishnavi, and S. Petter.)" In: (June 2019). URL: http://www.desrist.org/design-research-in-information-systems/.

[58] John R Venable, Jan Pries-Heje, and Richard L Baskerville. "Choosing a design science research methodology". In: *ACIS2017 Conference Proceeding* (2017).

APPENDIX A DATA WAREHOUSE AUTOMATION PRODUCTS

Ajilius was a data warehouse automation product. That has been available between the years 2015 until 2019. Ajilius was an exception in the market as they listed their competitors at the http://ajilius.com/competitors/ web page. Unfortunately, that web page does no longer exist. Luckily there are an archive versions available at the https://web.archive.org. We found 61 snapshot for that page. We have taken snapshops February 23, 2015 [1] and April 13, 2019 [2] for presenting years 2015 and 2019 in our Table A.1.

The list is comprehensive in Table A.1. Still there are note at year 2019 web page "Matillion and Panoply, originally included on this page, have been removed as they have clearly evolved towards ETL, rather than data warehouse automation."

[1] https://web.archive.org/web/20150223102358/http://ajilius.com/competitors/
[2] https://web.archive.org/web/20190413125223/http://ajilius.com/competitors/

Product	Methodology	Open Source	2015	2019
Ajilius	Dimensional		X	X
AnalytiX DS	Data Vault, 3NF, Dimensional			X
Attunity Compose	Data Vault			X
BI Builder	Dimensional	X	X	X
BI builders	Dimensional			X
biGenius	Unknown			X
BIReady	Data Vault		X	
Birst	Dimensional		X	X
Centennium Automation Tool	Data Vault			X
Datavault Builder	Data Vault			X
DDM Studio	Data Vault, Dimensional		X	X
Dimodelo	Dimensional		X	X
Effektor	3NF, Dimensional			X
Gamma Systems	Dimensional		X	X
Halo BI	Dimensional			X
Insource Data Academy	Dimensional			X
Instant Business Intelligence (SeETL)	Unclear		X	X
Kalido	Dimensional		X	X
LeapFrogBI	Dimensional		X	X
Optimal ODE	Data Vault	X		X
Quipu	Data Vault	X	X	X
TimeXtender	Dimensional		X	X
Varigence	Dimensional		X	X
WhereScape	Dimensional, 3NF (2015), Data Vault? (2019)		X	X

Table A.1 List of products from Ajilius competitor web page

PUBLICATION I

"Towards Agile Enterprise Data Warehousing"

Mikko Puonti, Timo Lehtonen, Antti Luoto, Timo Aaltonen, and Timo Aho

ICSEA 2016, The Eleventh International Conference on Software Engineering Advances (Aug. 2016), pp. 228–232

Towards Agile Enterprise Data Warehousing

Mikko Puonti
and Timo Lehtonen
Solita, Tampere, Finland
Email: puonti@iki.fi
timo.lehtonen@solita.fi

Antti Luoto
and Timo Aaltonen
Department of Pervasive Computing,
Tampere University of Technology,
Tampere, Finland
Email: antti.l.luoto@tut.fi
timo.aaltonen@tut.fi

Timo Aho
Yle, The Finnish Broadcasting Company,
Helsinki, Finland
Email: timo.aho@iki.fi

***Abstract*—Traditional business intelligence and data warehouse projects are very much sequential in nature. The process starts with data preparation and continues with the reporting needed by business measurements. This is somewhat similar to the waterfall model of software development and also shares some of its problems: the work is done in serial manner and the reaction time for possible design changes is often long. Agile principles are not well supported by the traditional serial workflow. By making the data preparation and reporting tasks parallel, it is possible to gain several advantages, such as shorter lead time and shorter feedback cycle. The solution proposed in this paper is based on enriched conceptual model that enables the business intelligence implementation process of different teams to change from serial to parallel workflow.**

Keywords–data warehouses; business intelligence; agile software development; scrum.

I. Introduction

Business Intelligence (BI) projects are traditionally following a pattern, where the work is actually done in serial tasks, which are strongly dependent on each other. This leads to long development cycles where some tasks need to be done before the next tasks can be even started. The problems of this approach include long feedback times and inefficient working process. The working method does not support the agile process models, such as scrum [1].

Scrum is an iterative project management approach to deliver software in incremental development cycles called *Sprints* that usually last from two to four weeks. Its benefits come from the ability to respond to the unpredictable environment changes as every sprint is planned separately.

In this article, we propose a process improvement to avoid the dependency of serial BI development tasks. The core of the idea is to rearrange serial development sprints to parallel ones by using a conceptual data model as a basis for a dimensional data warehouse (DW) model. The dimensional model is, on the other hand, an agreement between different development teams with different skills and, thus, a basis for communication between them. Research literature about combining BI with agile mindset exists but to the best of our knowledge none of them concentrate on how to organize work of teams in parallel way in agile BI project.

The expected benefits of our approach include shorter sprint cycle lengths, which leads to shorter customer feedback time. Also, it helps the DW modelers and BI reporters to concentrate on their work by reducing the fragmentation of development sprints, because of easier allocation of work. As a result, more development iterations can be done in the same time frame as with a serial workflow.

The proposed process improvement can be seen as a first step towards agile practices in BI projects and it can be later on combined with other agile practices.

The rest of this paper is structured as follows. In Section II, we introduce the necessary background for the paper by addressing the related work in agile BI processes. Section III presents the current and target states of the data warehousing and reporting process while Section IV introduces the approach from the viewpoint of data modeling. Finally, we draw some concluding remarks in Section V and outline our strategy for validating the expected benefits of the proposed approach in Section VI.

II. Related Work

The chosen related work concentrates on bringing miscellaneous agile practices to DW and BI processes. In general, incremental and iterative approaches are seen as beneficial in them but to the best of our knowledge, other authors have not discussed about organizing different teams' work in parallel so that traditionally done serial work could be done simultaneously. This is a gap we are trying to fill by improving the DW modeling process.

In [2], the authors categorize different agile BI actions in their literature review. Their categorization is based on previous work presented in [3] and identifies four agile BI action categories which are *Principles* (rules and assumptions derived from extensive observation and evolved through years of experience [4]), *Process models* (guidance to coordinate and control different tasks systematically which must be performed in order to achieve a specific goal [4]), *Techniques* (a way or style of carrying out a particular task) and *Technologies* (tools). The ideas presented in this paper fit to category *Process models* as the idea is to parallelize DW design tasks. The work in [2], also noted that agile principles are often discussed in a relation to agile process models, and in *Process models* category, Scrum can be seen as the most popular research topic between the years 2007 and 2013. We go through some of this previous work in the following paragraphs.

A process model called Four-Wheel-Drive (4WD) introduced in [5] utilizes six agile DW design practices (incremental process, iteration, user involvement, continuous and automated testing, lean documentation) that are based on software en-

 ISBN: 978-1-61208-498-5

gineering methods. According to them, the impacts of an iterative and incremental process are better and faster feedback, improved change and resource management, clearer requirements and early detection of errors. They discuss incremental techniques in the light of risk analysis that balances between the value to users and the risk of releasing early. Similarly, our approach aims to enable ways of working more iteratively and incrementally while also making customer feedback easier but they don't have the viewpoint of parallelization which would also shorten the required time for DW projects.

In addition to direct process improvement, the work in [6] presents an optimization model for sprint planning in agile DW design, which is based on the team's ability to estimate a set of development constraints. In contrast to our work, we do not concentrate on the planning phases of sprints even tough the planning should be also easier in our parallel workflow where teams are working more in close collaboration. They aim to optimize the sprints by planning whereas we optimize time usage with work parallelization.

The work in [7] gives a description of a DW project that was executed in an agile manner. The lessons learned include successful usage of agile Enterprise Data Models, tools integrated to version control and continuous integration of the database. Even though their usage of Enterprise Data Model improved communication and collaboration by shortening feedback loops between different teams, they don't explicitly mention about making the workflow parallel, which is our goal. Our approach similarly improves the communication and collaboration between teams.

III. Data Warehousing and Reporting Process

According to [8], BI is a process that consists of two main activities: getting data in and getting data out. The first activity, i.e., (DW), is about collecting data from source systems to a single DW that combines the data. The data is then extracted to a useful form for decision support. Getting that data out is the part that receives the most attention as it eventually brings out the value even though the DW part is considered to be more laborious.

The skills and the tools needed for the two activities are different. Thus, the competence is diversed in DW and reporting teams. DW implementation work consists of modeling in addition to Extract, Transform, Load (ETL) loads and data integration with an ETL tool. An ETL developer needs technical knowledge of databases and data transformations while a report specialist makes visualizations and needs understanding of the data. The naming of the data items in report meta model utilized for analysis is done using business terms. Hence, a reporting specialist needs understanding of the customer's business process.

The data is the driver for the whole implementation of the reports. For analytical purposes, data is stored in a dimensional schema of a data mart [9, Chapter 1] by the DW team. Report implementation consists of two steps. In the first step, a meta model of data entities and the structure of the data is created, while in the second step, the actual report is created with a reporting tool. Testing of the reporting functionalities is commonly done by an end-user with the actual customer data. Thus, a prerequisite for the report development is an existing DW utilizing dimensional schema which is populated with the customer's data.

The diverse expertise of the different teams and the need of an existing DW before starting the report development results in lengthy workflow in current BI processes.

A. Current State

Currently, the way of working divides the design and implementation process of BI report into two teams, in which one team finishes the DW design work and another team continues by producing the specified report. Only after both the teams have finished their serial sprints, it is possible to gain feedback from the customer and start fixing the problems, starting again from DW work and continuing to reporting. This is presented in the Fig. 1.

Fig. 2 presents the current state of the workflow in a timeline. In the figure, *DW Sprint* includes actions, such as data integration, ETL and DW modeling (dimensional model) while *Reporting Sprint* consists of actions, such as creating a meta model for the report and creation of the actual report. The specification describes the business requirements and the visual guidelines for the report.

The result of the work in *DW Sprint* is a data mart that utilizes dimensional schema. The data mart and data loads in the data mart are done by an ETL developer. In the scrum process model, the DW implementation is done first in a *DW sprint* as can be seen in the Fig. 2. After the *DW sprint* deliverable (the data mart with customer's data) is available, the report implementation will be able to start. This dependency leads to a situation where there is first a *DW sprint* after which a *Reporting sprint* will follow. Implementation of a report requires at least two sprints, since in the first sprint the data comes available to the DW (DW sprint) and the actual report for the end user is implemented in the next sprint.

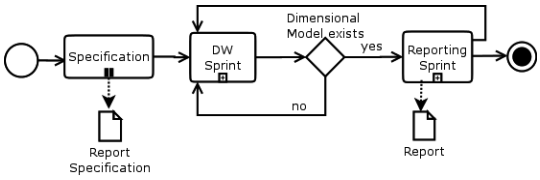

Figure 1. The current state of the process.

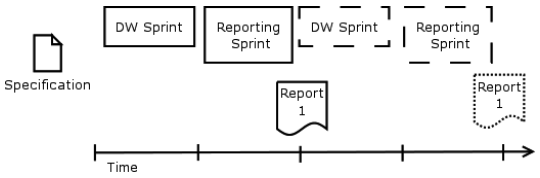

Figure 2. Workflow presented in a timeline.

B. Problem: Sequential Working

As a result of the diverse expertise in the teams and the need of an existing DW before reporting work, the full report development in *Reporting sprint* will not start before the first *DW Sprint* is finished, as it is presented in Fig. 1. The situation leads to a dependency between the DW implementation and report implementation.

The main problem of the current state is that getting feedback from the customer, which is based on the report, requires finishing both the sprints before it is possible to get feedback. After the feedback is received, the teams can start fixing the problems with new iterations of *DW sprint* and *Reporting sprint*. This also leads to fragmentation of work and excess waiting time between the sprints. Moreover, even though the workload is not as big as in the first iteration, it is still serial work and takes two sprints. If each sprint lasts for two weeks then completing both the sprints takes four weeks which multiplies to eight weeks after the feedback has been received and the corrections have been made. This is also illustrated in Fig. 2.

C. Solution: Parallel Working Enabled

As a solution to shorten the customer feedback cycle length and to defragment the DW and reporting work, we are targeting to parallelization of the serial sprints. The parallel team working is presented in Fig. 4. The parallelization is enabled by dimensional model based on conceptual model that contains information of the source systems. Based on the source system information in conceptual model, the dimensional model can designed at an attribute level with the support of interface specifications. A conceptual model presents associations between the modeled entities while the interface specification presents the attributes related to that association. The target state of the DW development process is presented in Fig. 3. The following aspects rise when comparing the current state to the target state.

1) Dimensional Model Based on Conceptual Model: Dimensional model represents *facts* which are business measures of the *dimensions*. The *dimensions* are grouping the business. Conceptual model consists of business entities and relationships between those entities. By adding information about a source system for an entity in a conceptual model, it is possible to get enough information of that entity without doing an exact logical data model. For creating the dimensional model, it is vital to know all the attributes of the *fact* and *dimension* tables. The attributes of each entity in a conceptual model can be solved out by looking at the interface of that entity. Each entity needs an interface from the source system to the DW and it the interface has to exist before the *DW Sprint* can start. The interface has the attribute information of the conceptual model entity, which makes it possible to create a dimensional model based on a combination of a conceptual model and an interface documentation.

2) Parallel Work of Different Teams: In the current state, the way of working was divided to serial sprints of different teams. The result of the completed DW sprint was a dimensional model which was utilized by reporting team. Thus, it would be beneficial, if the team could receive the dimensional model earlier to utilize it as a specification between them and the DW team. With the help of a dimensional model that is based on a conceptual model, it is possible to arrange the work so that the reporting team can to start developing the meta model for the reporting at the same time as the DW team starts the ETL work. In addition, the parallel way of working makes it easier for the teams to communicate with each other since they are concentrating on the same main goal, and further, the report can be produced in the end of the parallel sprints enabling customer feedback.

3) Shorter Feedback Cycle and Shorter Delay of Modifications: Since end-user is using the reports, getting useful feedback based on the report requires the report to include actual business data. Parallel working in *DW sprint* and *Reporting sprint* enables finishing the report in one sprint of calendar time. End-user can give feedback based on the report to both teams directly after the sprint. This is a huge difference to the DW team, which will get the feedback immediately after the sprint when compared to serial work in current state when the feedback was available only after the *Reporting sprint* was finished. This is beneficial because receiving feedback is more relevant when it is received directly and without delay. Faster feedback will also shorten the delay of starting the modification work. Therefore, making the modifications is easier since it requires less fragmented work and context switching.

Furthermore, parallel working shortens implementation time which also shortens the time that the end-user waits from giving the business needs to getting a report. In addition, the end-user is likely to be more participating in the process since the implementation time is shorter. According to [10], the end-user participation is such customer collaboration, which makes the product better. As an example of the effects in time, if a sprint lasts for two weeks, the parallel work ensures that delivering a new version of the report takes only two weeks. This is a notable improvement when compared to current state when delivering a report needed four weeks.

Data modeling is the key for communication between the teams and therefore it enables the parallelization of the work.

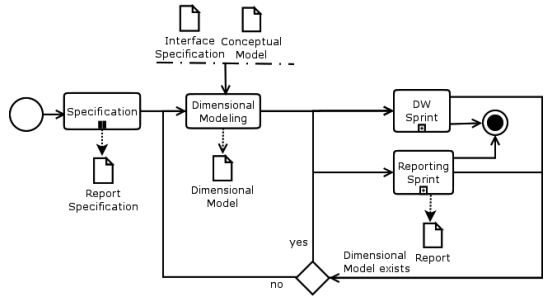

Figure 3. The target state of the process.

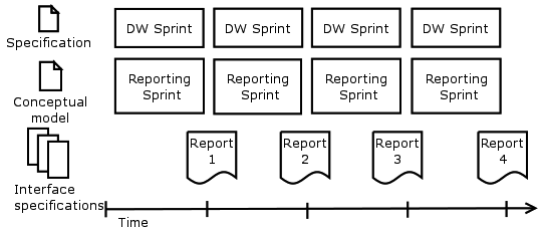

Figure 4. Sprints are parallel and feedback is faster.

IV. Data Modeling

Well managed data modeling is a crucial task for a DW project. Data modeling is about gathering the customers' data requirements and satisfying them with a DW solution.

 ISBN: 978-1-61208-498-5

According to [11], data modeling work is done on three design layers: logical, conceptual and contextual (by bottom-up order). Out of those layers, in this article, we are mostly interested in the conceptual and logical data modeling.

A. Conceptual Data Modeling

Conceptual data modeling is about modeling the user's data requirements in a conceptual manner using common concepts, such as entities and relationships. It describes the data and relationships between different data entities. Conceptual data modeling is a quick way to create a model of the problem domain with business representatives in a workshop, because the main entities come from the business domain and thus they have a business meaning. The collaboration between business stakeholders and data modelers is very important in order to tie the data intensive solution to the business processes.

Conceptual model can be utilized to ensure that all the participants share the same conceptual understanding of the modeled area [11]. In addition, it is a base that evolves to logical data model.

B. Logical Data Modeling

Logical data model presents all entities and their attributes. Each entity which has a primary key is marked in the model. Many-to-many relationships between entities are specified by creating an association entity between the entities. Creating a logical data model requires the following steps [12]:

- Specifying primary keys for all the entities.
- Finding the relationships between different entities.
- Finding all the attributes for each entity.
- Resolving many-to-many relationships.
- Normalisation.

The purpose of the logical data model is to provide a detailed specification for the physical relational database design [11]. In our context a logical data model is a tool for DW designers to produce a DW.

C. Dimensional Modeling

A dimensional model consist of fact and dimension tables in which the main items generally are *facts* and *dimensions* [9]. A *fact* represents a business measurement and is linked to several *dimensions*. A *dimension* groups and labels the measurements while it is also used to restrict the data set of measurements. Dimensional modeling is widely used modeling technique to offer data from DW to reporting tools.

D. Granularity of Data Modeling

The conceptual data model is important for communication between each participant in the project, especially for the business stakeholders, but it does not cover the detailed information needed in the implementation. The logical model, on the other hand, is more detailed but requires more work as it is relatively slow to model all the attributes and relationships of each entity.

The kind of data modeling described so far, is missing one critical piece of information as it does not tell where the data actually exists. The source system information is the most vital information in the reporting project. The needed granularity of data modeling is a mix of conceptual and logical data modeling enriched with information about the location of different entities. The combination of conceptual entities marked with the primary key attributes and information of source systems is the minimum required granularity of needed data model. A model should be enriched with the vital attributes, but the amount of attributes depend on how well the modelers know the domain. When the available information is well known and the business entity is clear, it is possible for everyone to understand the information even if it is not modeled in detail.

V. Conclusions

In this paper, we presented an idea to shorten the feedback cycle of BI projects. The proposed method consists of parallelizing DW and reporting team sprints by using a dimensional model as an agreement between the teams. Since modeling plays a crucial part in BI process, it is important to provide the dimensional model as early as possible. In this paper we claim this to be possible by developing dimensional model based on a combination of a conceptual model and the interface documentation of a source system.

Traditionally, reporting team starts working after DW team has offered a dimensional model with actual data. In our approach, reporting team can start working in parallel with DW team but initially without any actual data. The DW team implements ETL processes with small increments which gives then increasing amount of actual data to reporting team. It is worth noting that making the specifications in the new approach does not increase the overall process time. This is because interface specification is created implicitly anyway and conceptual model is very light weight to create.

As a result of the approach, the customer feedback cycle shortens which moreover makes the feedback more direct. Furthermore, because of parallel working, the communication between teams is more efficient and reaction time to feedback between teams is shorter. This is a step towards agile enterprise data warehousing where a bigger team consists of two separate teams with diverse competence.

VI. Future Work

As a future work, we are planning to conduct a case study in which we will utilize our ideas in an industrial BI project in a mid-sized Finnish software company. Moreover, we are eventually aiming at integrating the different teams (DW team and reporting team) so that the expertise of a person working in a BI project would cover both the required perspectives. That way, it is possible to reduce the amount of persons needed in a project.

The proposed idea is our first step towards agile BI projects, since it can be adopted with other agile principles, as well. To make the BI process even more agile and faster, we are studying how to shorten implementation time by generating ETL processes automatically based on modeling principles [13]. To get full advantage of these improvements, we also aim at creating release management practices to get our BI project closer to the continuous delivery.

Acknowledgment

The work was financially supported by TEKES (Finnish Funding Agency for Innovation) DIGILE Need for Speed program. We would also like to thank Solita and Yle for the possibility of doing this research.

References

[1] K. Schwaber, "Scrum development process," in the Proceedings of the Workshop on Object-Oriented Programming Systems, Languages and Applications Workshop on Business Object Design and Implementation, OOPSLA '95, Austin, Texas, pp. 117–134, October 1995.

 ISBN: 978-1-61208-498-5

[2] R. Krawatzeck, B. Dinter, and T. Duc Ang Pham, "How to make business intelligence agile: The agile bi actions catalog," in System Sciences (HICSS), 2015 48th Hawaii International Conference on, pp. 4762–4771, January 2015.

[3] R. Krawatzeck, M. Zimmer, and S. Trahasch, "Agile business intelligence - definition, maßnahmen und herausforderungen," HMD Praxis der Wirtschaftsinformatik, vol. 50, no. 2, pp. 56–63, January 2014.

[4] F. Tsui, O. Karam, and B. Bernal, Essentials of software engineering. Jones & Bartlett Publishers, 2013.

[5] M. Golfarelli, S. Rizzi, and E. Turrichia, "Modern software engineering methodologies meet data warehouse design: 4wd," in 13th International Conference, DaWaK 2011, Toulouse, France. Proceedings, pp. 66–79, August 2011.

[6] M. Golfarelli, S. Rizzi, and E. Turricchia, "Sprint planning optimization in agile data warehouse design," in Proceeding DaWaK'12 Proceedings of the 14th international conference on Data Warehousing and Knowledge Discovery, pp. 30–41, 2012.

[7] T. Bunio, "Agile data warehouse – the final frontier: How a data warehouse redevelopment is being done in an agile and pragmatic way," in Proceeding AGILE '12 Proceedings of the 2012 Agile Conference, pp. 156–164, August 2012.

[8] H. Watson and B. Wixom, "The current state of business intelligence," Computer, vol. 40, no. 9, pp. 96–99, September 2007.

[9] R. Kimball and M. Ross, The data warehouse toolkit: the complete guide to dimensional modeling. John Wiley & Sons, 2011.

[10] M. Fowler and J. Highsmith, "The agile manifesto," Software Development, vol. 9, no. 8, pp. 28–35, 2001.

[11] A. Sharp and P. McDermott, Workflow modeling: tools for process improvement and applications development. 685 Canton Street Norwood, MA 02062: Artech House, 2001.

[12] 1keydata, "Logical data model," http://www.1keydata.com/datawarehousing/logical-data-model.html, accessed: 2016-01-18.

[13] M. Puonti, T. Raitalaakso, T. Aho, and T. Mikkonen, "Automating transformations in data vault data warehouse loads," in Proceedings of the 26th International Conference on Information Modelling and Knowledge Bases, EJC 2016, pp. 219–235, June 2016.

 ISBN: 978-1-61208-498-5

PUBLICATION II

"A Continuous Delivery Framework for Business Intelligence"

Mikko Puonti, Juha Järvi, and Tommi Mikkonen

Frontiers in Artificial Intelligence and Applications 301 (2018), pp. 248–262
DOI: 10.3233/978-1-61499-834-1-248

A Continuous Delivery Framework for Business Intelligence

Mikko PUONTI [a], Juha JÄRVI [b], and Tommi MIKKONEN [c]
[a] *Solita PLC, Åkerlundinkatu 11*
FI-33100 Tampere, Finland
e-mail: puonti@iki.fi
[b] *Fennia Mutual Insurance Company, Televisiokatu 1, FI-00017 Helsinki, Finland, e-mail: juha.jarvi@welho.com*
[c] *University of Helsinki, Gustaf Hällströmin katu 2b*
FI-00014 Helsinki, Finland
e-mail: tommi.mikkonen@helsinki.fi

Abstract. In the context of business intelligence, data warehousing is often perceived as an integral component of concrete business intelligence solutions. Since the nature of a traditional data warehouse is accumulative – data from operational systems is fed in to the system when it is ready for inclusion – and as the data from different component systems is interrelated, operations involving data warehouses have been traditionally considered tedious and delicate. Distinct steps take place one after another in a predefined, next to unalterable sequence. In this paper, we present an alternative model for dealing with data warehouses, where the goal is to apply principles of continuous software engineering in the domain of business intelligence. To validate the methodology, we present a tool chain that has been used in a real-life implementation of a business intelligence solution, together with experiences from its operations.

Keywords. business intelligence, continuous delivery, data warehouse, process framework.

1. Introduction

A data warehouse system is a central component in business intelligence. Such systems are used for reporting and analyzing data that originate from various operational systems and data stores. Commonly implemented as central repositories of integrated data, data warehouses contain current and historical data that enables creating analytical reports for knowledge workers. To present some concrete examples, a report could provide annual comparisons and trends or daily sales analysis at a detailed level.

Since the nature of a traditional data warehouse is accumulative – data from operational systems is fed in to the system when it is ready for inclusion – and as the data from different component systems is interrelated, operations involving

data warehouses have been traditionally considered tedious and delicate. One may even call them waterfalls in the sense that the series of operations that it takes to create the system must typically be executed in sequence and in a fully fixed order, resulting even more rigid a creation and update process.

In this paper, we present an alternative model for dealing with data warehouses. The goal is to apply the principles of continuous software engineering [3] in the domain of data warehousing. More precisely, we present a warehouse architecture (partly building on our earlier paper [13]) that enables a methodology for data warehousing that is inspired by continuous delivery. In the field of software engineering, continuous delivery comprehends a build – test – deploy cycle of an application [5]. When the application in question is a business intelligence solution, it includes database structures, data transfer implementations, reports, and environment specific configurations. To validate the methodology, we present a tool chain that has been used in a real-life implementation of a data warehouse system, together with experiences from its operations. The main contribution of this article is to introduce pieces of business intelligence solution that enable continuous delivery.

The rest of this paper is structured as follows. In Section 2, we introduce the background of the paper. In Section 3, we present the layers of the architecture in the proposed framework. In Section 4, we describe the methodology used in connection with the framework. In Section 5, we propose techniques how to implement the methodology and create the landscape for the framework. In Section 6, we analyze the overall work and our experiences more generally. Finally, in Section 7 we draw some final conclusions.

2. Background

Within a relatively short period of time, agile development has become almost de facto as a way to implement any software project [9]. Indeed, with the promise of more satisfying outcome, shorter schedule, and lesser risks, agile development is seen as a natural choice. When pushed to the extreme, an approach results where engineering and deployment activities take place continuously [3].In contrast, business intelligence projects are still far from being agile, as they typically follow a very rigid workflow, consisting of steps that transform data from source systems into a format that is used in the business intelligence context. However, there is no fundamental reason – apart from existing mainstream tools, processes, and implementations – that business intelligence projects should be any different from software projects. In our experience, executing business intelligence projects in an agile fashion requires reconsideration of tools and methods used in the process, as well as a totally new mindset.

A business intelligence solution consists of several different tools for different purposes, including database modeling, database, ETL (extract, transform and load, a set of functions combined into one tool or solution), reporting metamodel, and eventual reporting itself. Working with these tools requires special skills, and therefore a business intelligence project consists of roles based on these skills. Typical roles include database modeler, ETL developer, reporting specialist, and

project manager. Corresponding roles in a typical software project are architect, backend developer, frontend developer, and project manager.

A typical business intelligence setup is such that project teams only use one development environment that is used by all the developers. Because this environment is shared among all the team members, individual team members cannot perform their tasks without considering others and their potentially ongoing tasks. Therefore, in any business intelligence project of even a moderate size, there are multiple dependencies that the team has to settle as a part of the process. Furthermore, as current business intelligence best practices include manual work steps when deploying implementation to a different environment, each tool specialist has to take part in the deployment. To minimize the time window when a deployment is executed, the people must schedule their daily work such that they can participate in deployment at the same time. All this synchronization and presence makes each deployment very costly. Consequently, the number of deployments are to be minimized.

Due to the close relation to the field of software engineering, many business intelligence developers are familiar with continuous delivery [6, 5]. By automating the deployment process, any team member can do a deployment, including the actual deployment as well as all the necessary quality assurance steps to validate the correctness of the system [6]. This eliminates the need for tool specific skills and enables the creation of several environments that can be targeted for different needs, such as testing different feature sets independently. Then, by using developer or even feature-specific environments, the development no longer depends on activities of other team members, thus effectively enabling parallel development and removing scheduling and dependency related constraints from development. Unfortunately, prevailing business intelligence practices are not considered compatible with continuous delivery.

The difference between traditional and continuous business intelligence has been identified as a key issue for creating a more agile approach to business intelligence [12]. Fundamentally, the problem in the traditional fashion boils down to the fact that the workflow of implementation is serial in its nature, and thus every new step in the process depends on all the previous steps and requires the presence and availability of all the necessary skills. In contrast, agile information infrastructure allows more liberal execution of the necessary operations. In the following, we present a data warehouse architecture, which builds on the traditional features of data warehouse systems, but which is extended with extra layers that supports more agile execution of operations.

3. Architecture of the Framework

Exploiting the ideas of continuous development in the context of business intelligence projects requires using an underlying architecture, where the design goals are set in preparation for change, continuous delivery, high degree of automation, and support for parallel activities, so that a single person can implement new increments. The architecture we use in this paper has been developed in the context of several industrial business intelligence projects (Figure 1). It consists

of the following logical layers: information usage, virtualization, publish layer, enterprise data warehouse, staging area and source systems. Each logical part in this architecture can be implemented in various ways, using different, readily available implementation techniques. Depending on the technology selection, the logical architecture may also be partitioned differently than proposed here, but still every logical functionality is needed in the implemented architecture.

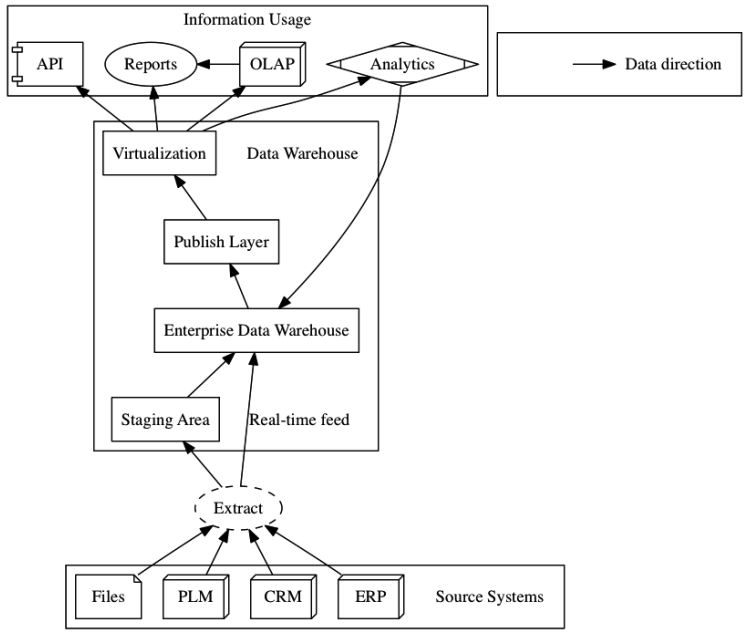

Figure 1. Different layers in data warehouse architecture

3.1. Information Usage

Information usage is a reason why data warehouses exists. A traditional business intelligence system is more or less a portal where several reports exist.Tools used in the information usage layer have to be available for each environment which will be created for developing a business intelligence solution.

Today, it is common that data warehouse content is provided as a data service via application programming interfaces. Data service gives the data warehouse an operational role, and we may start speaking of a real time data warehouse.

Analytics is creating new, typically more refined information out of information that is already acknowledged. This new information will be stored back to

the database, for other information usage applications as any other information would be.

3.2. Virtualization

In the architecture we use in this paper, virtualization – a layer between information usage and the actual information – has two primary goals. These are data access and user access control. Van der Lans lists several technical advantages related to data virtualization, and one of them is unified data access [15, Chapter 1]. Information usage layer can have various technologies if the virtualization layer offers several ways to connect to a publish layer.

It is common that data warehouses contain essential intellectual capital and other proprietary information, and therefore access control must be introduced. User access control guarantees that information is presented only to the right users. This function is most naturally implemented at a centralized place at the virtualization layer.

3.3. Publish Layer

The publish layer – also called an information mart [11, Chapter 2] – presents data using a vocabulary that is familiar to the information users. The layer is analogous to the presentation layer in OSI reference model [16].The same data may have different interpretations in different business cases and in different context as defined in the publish layer. Thus the publish layer is providing the information out of the data warehouse in a fashion that is compatible with user needs.

The traditional modeling technique for data warehouse is dimensional modeling [8]. In this context, publishing means transforming the data from enterprise data warehouse model to the dimensional model. The publish layer also contains data models for data services and data matrix for statistical analysis. Soft business rules are executed when creating the publish layer. By executing the soft business rules after storing the actual data in the data warehouse layer we do not lose any data. Furthermore, this also fosters modularity, as the dependencies of the soft business rules are only included in the publish layer.

The publish layer uses a dimensional model as an agreement that defines which data set is published for reporting purposes, and in what kind of format. Creating a dimensional model as a first step of implementation enables working with information usage and enterprise data warehouse layers in parallel [14].

3.4. Enterprise Data Warehouse

An enterprise data warehouse is the persistent storage that integrates data and stores the history of the incoming data sets. W. H. Inmon introduced term DW 2.0 where data warehouse is built a step at a time, over a long period of time [7, Chapter 8]. In addition, he claims that DW 2.0 is built by many people, not just a single person [7, Chapter 8]. These requirements – building the warehouse in steps and having several people operating with it at same time – are crucial for the data warehouse development approach. Inmon recommends using data vault as a modeling technique in "The Data Vault is the optimal choice for modeling the

EDW in the DW 2.0 framework."[1]. Flexibility of change and incremental development are the reasons why the authors recommends to use data vault modeling as the modeling technique in enterprise data warehouse layer.

3.5. Staging Area

Staging area is a data storage area to assists data loadings. Data is copied as is from source systems to the staging area. The staging area is transient [10, Chapter 2] and there is no persistence for staging area data. Instead, the data is persistent in the enterprise data warehouse layer. Therefore, the whole staging area must be automatically buildable with scripts, calling for additional information systems such as a version control system to assist. We will return to this topic later in the paper when describing the necessary tools.

Again, the implementation of the staging area can involve several different technologies. Probably the most commonly used technology is a relational database, in particular if the enterprise data warehouse is implemented in relational database, but file systems and distributed file system like Hadoop are also often used.

Strictly speaking, the staging area is actually optional, as data upload to a data warehouse may consist of a real-time feed. However, in batch based loading cycle, there are some advantages of using staging area:

- Data is extracted only once from source system interface in a batch
- When data is available in same database we may use database functionalities for comparison what data is actually changed in source system compared to the enterprise data warehouse layer

3.6. Source Systems

By definition, data warehouse's data originates from several source systems. These source systems typically offer interfaces with which the data can be fetched to the data warehouse. As many software systems offer data from database or file exports, database connection and file transfer are the most commonly used data extract technologies, although web services are becoming a more and more common way to extract data. In particular, when aiming at a real time data warehouse, using web services has benefits over the other commonly used techniques.

4. Methodology

Next, we present a methodology to create a business intelligence solution in an incremental fashion, where the goal is to release as often as possible. The methodology is based on the steps presented in Figure 2.

Each implementation increment starts with planning what to do in that particular increment. In the planning session, the specification of end-user needs is analyzed, refined, decomposed, and scheduled in collaboration with end user and

[1] https://danlinstedt.com/solutions-2/quotes/

business intelligence team. Business value is used to prioritize each possible implementation package, and packages should be implemented in order of business value importance.

Publish layer design is the starting point for implementation. The publish layer is designed to serve needs which are derived from information usage.

Data integration consist of getting data available and integrating that data in the enterprise data warehouse layer. To enable parallel working for enterprise data warehouse layer and information usage, we need DevOps [2] practices, where development and operations act as a single unit working in collaboration.

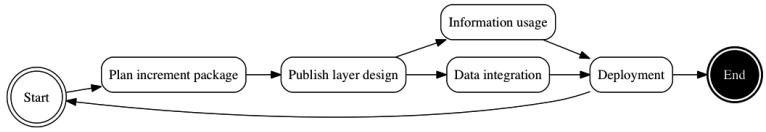

Figure 2. Steps in Continuous Delivery Framework Methodology

4.1. Collaborate with End User

Collaboration with the end user – in essence, the customer of the business intelligence team – is necessary to clarify what information is really needed. When this information is refined, it reveals what data is needed and how that data is to be presented to an end user. This constitutes detailed requirement gathering and specification of the solution.

It is advisable that the whole team participates in the collaboration with the end user, as then the whole team has a unified goal. In addition, understanding end user needs motivates the whole team to solve the end user needs as well as it can.

Business intelligence solution is not only a database or a dashboard, but the whole end user experience with logins and waiting times altogether. To this end, in addition to listening to the end user, it is also possible to deploy the solution in a development environment and test the end user look and feel. Furthermore, if in the development a deployment is quick to do, then it is possible to demonstrate the end user experience in each revision of the solution.

4.2. Focus on Business Value

The benefits of continuous delivery practices become visible when an end-to-end solution is composed such that it can be created in small pieces, each of which creates business value to the end user. To accomplish this, each implementation package that provides new functionality to a end user must be decomposed into small enough packages, so that it is possible to implement them within the available time window. This forms a sharp contrast to the traditional approach, where the creation of a data warehouse starts by integrating several source system data in one data lake, potentially requiring huge amount of integration work. Only after

the data lake is complete, information usage can begin in the form of developing reports and other information to the end user.

The proposed incremental fashion to compose data warehouses starts from end user needs. Business value is used as the driver when partitioning the data lake implementation into small packages which, when fully completed, will form the end-to-end solution. The development is initiated by creating a small dataflow pipeline from data source to a end user, we could describe these small dataflow pipelines as data creeks. As more and more small creeks are composed, the implementation is growing to a pond and further to a data lake. Furthermore, by creating business value all the way, after each implementation increment the investment will pay back more and more.

4.3. Information Needs To Drive the Development

The development should start from the information usage layer. Information usage yields specifications for the publish layer structures and their data content. The development team makes detailed analysis of data content and creates list of implementation activities to support required business use cases.

To enable parallel work, the first step is modeling a dimensional model in a publish layer [14]. Creating publish layer structures and populating the structures with real sample data enables development of reports and other information usage. Populating can be a virtual implementation, where database views from a data warehouse or a staging area are used. Information usage implementation can start immediately after an publish layer has data available. This enables prototyping and end user involvement in the development phase.

4.4. Parallel Development

As already discussed, we aim at dividing each implementation step in small tasks, which can be implemented in parallel. For instance, when implementing a dashboard that consists of several reports, these reports can be made at the same time and as they are completed, the final dashboard is created in iterative fashion.

For obvious reasons, parallel development sets requirements to tools, but the resulting improvement in flow efficiency usually exceeds these restrictions by far. Parallel working in information usage and enterprise data warehouse layers need DevOps practices to communicate the state of development and to integrate different code pieces together.

4.5. DevOps Mindset for Data Warehousing

Fundamentally DevOps is about collaboration and communication – indeed, it a way to implement continuous delivery in a fashion where developers and operators work together to accomplish a joint mission. While originating from the field of software engineering, the DevOps mindset is also a good match with data warehouse projects, where communication is needed between different stakeholders and developers and technical infrastructure must be in place to support changes in the solution in automated fashion. Automating the build process means that

the developers will get immediate feedback if problems emerge when integrating all the pieces needed for the complete solution.

A part of DevOps mindset is rapid creation of necessary infrastructure, so that a new environment can be set up for a specific purpose, to the extent that every developer can have her own environment. If so-called infrastructure-as-code (IaC) approach [4, Chapter 9] is used, a version control system can be used to create different instances of the same solution, allowing running development environments in the latest version of every developer and production environment in the version which has passed quality assurance testing. Furthermore, if there are any bugs in production, it is possible to duplicate the environment to examine and fix those in a separately created environment. Furthermore, it is also possible to validate the fixes separately, and incorporate the changes to the main solution only after that. Rapid deployment pipeline leads to a situation where a fix for production environment is done in the development environment, and the latest development version is simply deployed to the production environment. At this point we have DevOps practices to facilitate parallel development, and information usage is going forward based on publish layer structures next is to find data for information and implement historization for that data.

4.6. Get Data Available

The publish layer design is based on end user needs. The end user needs give understanding of business rules, which is used for fetching data. The data can be located in data warehouse, or, if it is not yet in data warehouse, then it is located in source systems. Usually data is located partially in data warehouse, and a part of the data is not available in data warehouse. If this is the case, source system specialists specify what source system interfaces can be used to fetch the needed dataset that is not yet in data warehouse, so that these datasets can be fetched from source system to data warehouse staging area. After data is stored in data warehouse, in enterprise data warehouse layer or in staging area, it is possible to publish the data to the end user. The first version of the publish layer is created by using database views upon data from staging area and enterprise data warehouse layer.

4.7. Integrate Data to a Enterprise Data Warehouse

In the traditional warehouse approach, this is the actual phase where all the implementation work is executed. In our approach this is clear – the specification is understandable to everyone in the business intelligence team and needed dataset is figured out and is available in staging area – but tedious phase. The Publish layer has structures where to serve the data after it is persisted in enterprise data warehouse layer. To clarify this, the input and output is known, only implementation is missing.

Implementation needs several steps, and this can require a huge amount of working time. Database modeling creates structures to the enterprise data warehouse layer. Data transfer creates data transformation from staging area to enterprise data warehouse and implements load chains of transformations to cache

data to data warehouse. Data transfer implementation can use template patterns for data vault entities. Using templates will speed up implementation work, this template thinking can go further even to automating implementation [13].

Persisted data from enterprise data warehouse will be fed to the publish layer. Queries from enterprise data warehouse structures will include business logic. These business logics are documented in the system that is available to end users. Created queries have to be tested carefully. This testing is implemented so that the tests can be executed automatically after each deployment.

4.8. Deployment Package

Each implementation is decomposed as so small pieces of implementation as is possible. The developer creates a full implementation in her own environment, which may be tricky when doing it for the first time. The goal of this piece of implementation is to create a deployment package, which can be deployed to other environments, and business intelligence tools are not designed to support this kind of working method.

All deployment steps are written as operating system scripts. While each separate tool has own script, these are put together in one main script. More about how to design the script and separating configuration out of a code is in Section 5.2.

When a deployment package is ready, it is tested and validated against new environment. A successfully validated deployment package is communicated to other team members.

5. Implementing the Methodology

The methodology described above has introduced the steps to achieve parallel working and showed how those steps lead to rapid, incremental deployment. Next, we explain the principles and tools to implement the methodology in practice. The discussion is based on the following key items:

- Documentation is created in collaboration with end users, as information exchange is two-way communication with team and stakeholders.
- Version control system usage is the key for achieving the continuous delivery.
- Business intelligence solution needs data transfer from place to place, but it is more than ETL in traditional data warehouse.
- Automatization is a powerful tool to shorten the lead time. Deployment pipeline automatization is the starting point, but there are several other areas in business intelligence projects that can be automated.
- Data warehouse core is database, that can be incrementally built by using SQL commands to migrate a database.

5.1. Document with End User

In addition to business representatives, it is important to be on the same page with the end users. Therefore, it is advisable to document everything what is useful

for a end user or is needed in communication between end users and the team or between the team and stakeholders. Documentation can take many forms. It can be a guide for a end user, report specification, interface specification, business rules documentation, or project review material, for instance.

A practical way to share such documentation is using a wiki, which provide collaborative modification to content. In addition, with a wiki it is easy to start with something small, and reorder layout and structure of documentation later, if needed. Furthermore, wiki allows involving end users in the process of documenting the business intelligence solution.

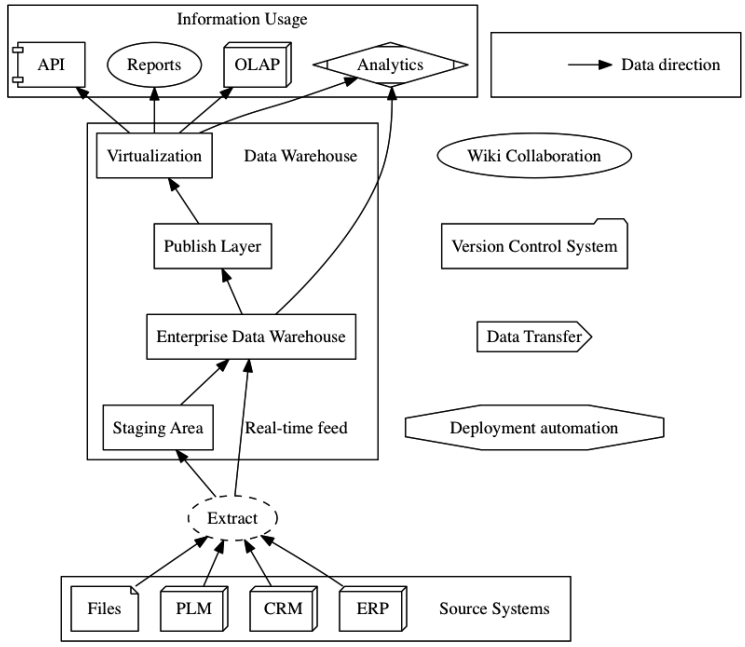

Figure 3. Data Warehouse Layers and Tools

5.2. Version Control System

While one might think that version control usage is common in business intelligence projects, based on everyday experience we claim that in industrial business intelligence projects a version control is rarely used. While some tools may offer version control systems that is in active use in the context of that particular tool, the use of a version control system is not comprehensive.

To implement the presented methodology, all the implementation done under a business intelligence project is in version control system, because otherwise business intelligence project deployment process simply cannot be automated reliably. Deploying all implementation from the version control system ensures that the same artifacts are in every environment. This ensures that what is tested in test environment is what actually ends up in production environment [6]. To support this, environment related configuration, such as database connection strings, user names and passwords, is not to be mixed with the implementation code.

A part of the automation process is to create scripts that build and deploy the whole implementation of the business intelligence solution from the version control system. This may be hard at first, because the current practices often include manual steps such as copying files and building with tools own functionality. However, it is very rewarding to figure out how the build can be done with the tool by using scripts from command line. Linking the whole solution to bigger script of each tool commands, the end result is deployment script for the whole business intelligence solution. This forms an easy way to deploy a certain version of the version control system.

5.3. Data Transfer

Data transfer can be implemented in many ways. This process is not only caching data in different places in our approach, as data has to be copied (cached) from source systems to staging area or enterprise data warehouse layer. When data is already available somewhere in data warehouse, it can be transferred using data virtualization technology (different than virtualization layer). Data virtualization technology provides intermediate layer that hide where and how data is stored [15, Chapter 1].

Traditional data warehouse is implemented by using an extract transform and load (ETL) tool. An ETL tool is valid tool for data transfer in our framework, but we have labeled moving data from place to place as data transfer. Data transfer typically includes a larger set of tools than one ETL tool.

5.4. Automate

A general rule of thumb in computing is to automate things that happens often. The first thing to automate is the deployment pipeline. Automation server is tool to create deployment pipeline automation. Requirements for automation server software in our framework is triggered by a version control system commit and the capability to reach all the servers used in a business intelligence solution. The build process can be triggered by requesting specific version of version control system.

After the deployment pipeline is automated, the focus is placed on automating other parts in business intelligence projects. As an example, we have implemented Data Map tool for creating transformations between staging area and enterprise data warehouse layer [13]: Earlier, data transfer implementation took day for one dataset to integrate it to a enterprise data warehouse layer, now it takes approximate half an hour when using Data Map tool to automate data transfer implementation.

5.5. Database Migration

Business intelligence projects differ from software projects in one important aspect: usually software projects can reload data from somewhere as initial data set, whereas in data warehouse projects this is not possible. A data warehouse stores the history of data, and this history data does not exist anywhere else. Consequently, it is of prime importance to ensure that the data is not at risk.

Database modification is done with database definition language (DDL) and data manipulation language (DML). Database definition language describes data structures, and database manipulation language can select, delete, insert or update data [1]. Both languages are included in Structured Query Language (SQL).

Database migration consists of a set of database definition language and data manipulation language commands, which are executed in certain order. Each SQL statement modifies the database structure or the data itself. SQL statements have to be executed in a certain order, and the developers have to plan and implement the order of SQL statements execution. When the implementation is decomposed in small pieces, there is an appropriate number of SQL statements to write. These SQL statements will be collected to one SQL script, which is added to version control system.

Individual SQL scripts of the implementation have to be executed in ascending order. This can be enforced with tools such as dbmaintain[2] and flywaydb[3] or simply by using a naming convention.

6. Discussion

Continuous delivery practices enable an approach where each developer uses one or more environments. Because each developer can act on her own systems, this enables parallel development – a major improvement over the conventional setting in data warehouse operations, where the development environment is shared among all the team members.

While providing an isolated environment for each developer is one tool for parallel working, dependencies between increment packages require additional attention. Decomposing the implementation into small but meaningful increment packages helps dealing with these issues. Unfortunately, decomposing the implementation into small pieces easily leads to a huge number of increment packages. Then, associated business value can be used to define the importance for each increment package.

Building a deployment pipeline based on scripting is sometimes difficult. Many of the current tools do not support such form of scripting, whereas each tool would have to support scripting to include it in the deployment pipeline. Every tool in a business intelligence solution can be changed separately, but obviously this can introduce costs. Therefore biggest advantages can usually be achieved in projects that are just starting and have not yet selected the tools to be used.

[2] http://www.dbmaintain.org
[3] https://flywaydb.org

In the project where the described framework was tested, we have successfully created a deployment pipeline for data warehouse. We are using Vagrant[4] to create new development environment, using the infrastructure-as-code approach. After creating a new environment, we need to manually log in to the created server and check the system out from version control system. The implementation is deployed by using two scripts. The first script includes configurations, and it prepares the business intelligence solution for usage. The second script deploys the actual implementation. In terms of the implementation, the latter script is actually an ordered list of SQL scripts that form the implementation (Section 5.5) in ascending order by using the agreed naming convention.

The database includes metadata tables that define how the data is transferred – in other words, from where to where data is moved. The data transfer can builds the load chain dynamically based on metadata information. This metadata can be added via implementation SQL scripts.

Current practices enable deployments to the data warehouse several times in a day. Furthermore, feedback from using the new version is immediate, as it can be deployed with one manual step only. Same goes the other way around, if the new version is not satisfactory, the previous version can be re-deployed instead.

Overall, we are very satisfied with the present situation, although there still are some directions for future work. In particular, information usage layer follows the traditional reporting delivery approach, because of licensing issues related to reporting tool that was shared between several projects. Consequently, we have not added reporting solution to the deployment pipeline, and thus we can not create developer specific environments for that reporting tool.

7. Conclusion

In this paper, we propose a data warehouse approach inspired by continuous software engineering practices. The biggest benefit of this approach is that its implementation starts with prioritized end user information usage needs, whereas a more traditional approach would extract data and execute a workflow in a serial manner, finally ending in collaboration with the end user.

The proposed architecture immediately supports continuous delivery. It also supports dividing the implementation into independent increment packages. Those packages can be implemented separately in separate environments, and, once completed, they can be composed into a bigger whole, also in accordance to the continuous delivery practices. Currently available tools support an approach where a separate environment is allocated for each developer, thus eliminating the downsides of a shared environment that creates dependencies and interruption for a single developer. We have implemented the creation of developer specific environments using the infrastructure-as-code approach, which is a good match to our needs. Based on our findings, there is no fundamental reason why business intelligence projects should be any different from software projects.

[4] www.vagrantup.com

Acknowledgements

The work was financially supported by TEKES (Finnish Funding Agency for Innovation) DIGILE Need for Speed program. We would also like to thank Solita for the possibility to do this research.

References

[1] ISO ANSI. Database language sql iso/iec 9075: 1992, 1992.

[2] Patrick Debois. Devops: A software revolution in the making. *Journal of Information Technology Management*, 24(8):3–39, 2011.

[3] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[4] Michael Httermann. *DevOps for developers*. Apress, 2012.

[5] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010.

[6] Jez Humble, Chris Read, and Dan North. The deployment production line. In *AGILE*, volume 6, pages 113–118, 2006.

[7] William H Inmon, Derek Strauss, and Genia Neushloss. Dw 2.0: The architecture for the next generation of data warehousing. 2008.

[8] Ralph Kimball and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons, 2011.

[9] Alexis Leon and Alan S Koch. *Agile software development evaluating the methods for your organization*. Artech House, Inc., 2004.

[10] Dan Lindstedt and Kent Graziano. *Super Charge Your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault*. CreateSpace, 2011.

[11] Dan Linstedt and Michael Olschimke. *Building a Scalable Data Warehouse with Data Vault 2.0*. Morgan Kaufmann, 2015.

[12] Mihaela Muntean and Traian Surcel. Agile bi-the future of bi. *Informatica Economica*, 17(3):114, 2013.

[13] M Puonti, T Raitalaakso, T Aho, and T Mikkonen. Automating transformations in data vault data warehouse loads. In *Proceedings of the 26th International Conference on Information Modelling and Knowledge Bases, EJC 2016*, pages 219–235, June 2016.

[14] Mikko Puonti, Timo Lehtonen, Antti Luoto, Timo Aaltonen, and Timo Aho. Towards agile enterprise data warehousing. *ICSEA 2016*, page 241, 2016.

[15] Rick Van der Lans. *Data Virtualization for business intelligence systems: revolutionizing data integration for data warehouses*. Elsevier, 2012.

[16] Hubert Zimmermann. Osi reference model-the iso model of architecture for open systems interconnection. *IEEE Transactions on communications*, 28 (4):425–432, 1980.

PUBLICATION III

"Automating Transformations in Data Vault Data Warehouse Loads"

Mikko Puonti, Timo Raitalaakso, Timo Aho, and Tommi Mikkonen

Frontiers in Artificial Intelligence and Applications 292 (2017), pp. 215–230
DOI: 10.3233/978-1-61499-720-7-215

Automating Transformations in Data Vault Data Warehouse Loads

Mikko PUONTI [a], Timo RAITALAAKSO [a,b], Timo AHO [c] and Tommi MIKKONEN [b]

[a] *Solita PLC, Åkerlundinkatu 11, FI-33100 Tampere, Finland, e-mail: puonti@iki.fi, timo.raitalaakso@iki.fi*

[b] *Department of Pervasive Computing, Tampere University of Technology, PO BOX 553, FI-33101 Tampere, Finland, e-mail: tommi.mikkonen@tut.fi*

[c] *Yle, The Finnish Broadcasting Company, Box 97, FI-00024 Yleisradio, Finland, e-mail: timo.aho@yle.fi*

Abstract. Data warehousing is a process of integrating multiple data sources into one for, e.g., reporting purposes. An emerging modeling technique for this is the data vault method. The use of data vault creates many structurally similar data processing modifications in the transform phase of ETL work. Is it possible to automate the creation of transformations? Based on our study, the answer is mostly affirmative. Data vault modeling creates certain constraints to data warehouse entities. These model constraints and data vault table populating principles can be used to generate transformation code. Based on the original relational database model and data flow metadata we can gather populating principles. These can then be used to create general templates for each entity. Nevertheless, we need to note that the use of data flow metadata can be only partially automated and includes the only manual work phases in the process. In the end we can generate the actual transformation code automatically. In this paper, we carefully describe the creation of automation procedure and analyze the practical problems based on our experiences on PL/SQL proof of concept implementation. To the best of our knowledge, similar has not yet been described in the scientific literature.

Keywords. data vault, database modeling, ELT, ETL, code generation

1. Introduction

Data warehousing is a technique for integrating data from several source systems to enable reporting and finding dependencies in the data. Data merging from different data sources to a data warehouse is typically done in three phases: The data is *E*xtracted from source systems, *T*ransformed to the target structure, and *L*oaded to a data warehouse. The order of the last two stages may vary and we call the variants accordingly Extract-Load-Transform (ELT) or Extract-Transform-Load (ETL). These phases are typically done in all the data warehouse modeling types.

In the case of ELT, the two first functions form a way to bring source data as-is to a staging area in a data warehouse. In this case, extract and load functions use similar data structure, and the transformation is done only at the last stage in the process. The final transformation, the final phase, is actually about loading data from the initial staging area to data vault tables. Its implementation can be (and often is) an ETL system of its own, and an ETL tool is used manually to define the transformation. This is the case in data warehouses using data vault modeling technique [5]. Data vault is a modeling technique designed to meet the needs of enterprise data warehousing. In the original work, Linstedt gives the following definition for it: "The Data Vault is a detail oriented, historical tracking and uniquely linked set of normalized tables that support one or more functional areas of business."[5].

Modeling is an integral part of the work in data warehousing and there are different ways of doing this. In particular, data vault modeling has become a popular way in designing a data warehouse because of its flexibility. When using data vault as modeling technique there are several similar data transformations. When adding a new data source to a data warehouse, the data model needs to be updated. Information about attributes in the source system interface is used when modeling the data warehouse. In addition, the transformation from a data source to a data warehouse needs to be tuned manually.

In this paper, our research concentrates on this manual work—how it is possible to minimize or even omit the manual part? The goal is to reduce the amount manual work needed by using model information more intensively. Based on the data vault modeling technique, there are rules based on the structure and rules on how the data vault entities will be populated. Based on these principal rules, it is possible to automatically generate ETL code. This code generation needs information about model and data flow. To enable this, a data model of system's metadata is created for this code generation. However, the process of creating transformations based on the metadata of models is different than in a manual ETL process.

As a practical contribution, we have implemented a PL/SQL proof of concept code generator for transformations from a data source to a data warehouse using data vault modeling. In this paper, we describe the process of automatically generating the transformations based on the metadata of data models. In addition, we present our implementation and analyze the learnings and pitfalls based on it.

The rest of this paper is structured as follows. In Section 2, we introduce the background of the paper. In Section 3, we present the principles of populating the data vault entities. In Section 4, we introduce data model to automate transformation and what to automate with it. In Section 5 we provide a small example regarding data map usage. In Section 6, we analyze the overall work and our experiences more generally. In Section 7 possible new research topics. Finally, in Section 8 we draw conclusions.

2. Background and Related Work

Jovanovic and Bojicic present a platform independent metamodel to store the data warehouse model information [3]. They give several examples of doing the conversion from a logical data model to a data vault model.

Phipps and Davis automate the data warehouse conceptual schema design [9]. They divide data warehouse creation to five steps: pre-development activities, architecture selection, schema creation, warehouse population, and data warehouse maintenance. They focus on schema creation phase and automation of that, whereas we are focusing on the next phase, warehouse population.

El Akkaoui et al, propose a model-driven development framework for ETL processes [1]. That framework aims at automatic generation of ETL code for several vendor specific platforms. The vendor-independent model is defined using a platform independent design model of ETL based on business process model notation (BPMN4ETL), ETL processes are generated based on that model.

Pankov et al share the idea of using metadata as a starting point for automating data warehouse implementation [8]. They describe metadata model which is modelled with data vault principles. Based on that metadata they show possibilities regarding how to generate ETL processes. A limitation of this approach is that the ETL tool has to expose its functionality as an application programming interface.

Several case tools for database modeling are available, such as Oracle SQL Developer Data Modeler[1] and ER Studio[2]. They use their own internal models to store the model information. The core usage of these tools is to forward and reverse engineer database structures. It is also possible to draw process diagrams with these tools. Data flow may be derived from a process model, if the structure is tied to the processes. Attributes used in process information structures may be associated with table columns with these tools. This capability is hidden behind several steps in graphical user interfaces. No code generation capabilities are available based on this information. In this case, traditional ETL tools are used to develop the transformations.

Data vault modeling consists of three major components. A hub has only the business key columns for a specific entity. A link draws a many-to-many relationship between several hubs. A satellite holds descriptive information about the context. In addition, reference tables are related to data vault modeling. Nevertheless, according to Lindstedt [4, 102], reference data should be separated from other data vault tables.

Data vault modeling introduces flexibility by adding new entities to the model. Existing entities and structures will remain and new modeling will be added to the model. As for the tool support for doing this, while data warehouse software with automation features exist, most of the software systems that are suited for data vault modeling are licensed products. However, there are two

[1] http://www.oracle.com/technetwork/developer-tools/datamodeler/overview/
[2] http://www.embarcadero.com/products/er-studio

open source distributions: Quipu[3] and Optimal Data Engine[4]Both of these use Microsoft Visual Studio as the front-end for the development.

3. Principles for Automation

Data vault modeling yields certain rules that can be used in code generation. Firstly, data vault entities hub and link have a surrogate key which is a one column primary key. Secondly, a satellite primary key is constructed with the surrogate key and load time. In addition, satellites and links have foreign key references. In this section these rules are explained for each data vault entity: hub, link, satellite, and reference.

Common attributes to all data vault tables Every data vault table has metadata. This metadata is stored in attributes as listed in Table 1. Our naming convention is to start every data vault metadata attribute with prefix `DV_`.

Table 1. Data vault common attributes

`DV_ID`	Surrogate key
`DV_LOAD_TIME`	Timestamp when the row is inserted
`DV_SOURCE`	Metadata of the source system
`DV_LOAD_NAME`	Name of the transformation which inserted the row
`DV_RUN_ID`	Every load batch has a unique id

3.1. Principles for a Hub

Business keys are vital to locate the business data [7]. Natural business key is a unique identifier for business concept [2, Chapter 4]. A hub present the business key for the business concept. If the business uses the same business key in all steps of the business process, then all data would be linked via the hub. Unfortunately, this is not usually the case, since different systems use quite often sequence numbers for their business keys. A hub is a mapping from a business key to a surrogate key. Surrogate key is one column primary key.

A business key may consist of several columns. A hub stores all unique business key values and creates a unique key constraint for business key attributes.

3.2. Principles for a Link

A link is an associative entity between hubs. When two or more business keys interact, a link is created to represent this interaction. This link is a many-to-many relationship table between hub tables, storing hub surrogate key values. Link table attributes are the common attributes listed in Table 1, together with all associative hub surrogate key values, which are related to that particular link. A link can also be used to store hierarchy information of a hub. In this case,

[3] http://www.datawarehousemanagement.org
[4] https://github.com/OptimalBI/optimal-data-engine-mssql

the link has relations to only one hub table. Every link hub surrogate key has a foreign key reference to the related hub surrogate key attribute.

Data vault modeling allows creating more than one link between two or more hub tables. The link structure introduces flexibility in the modeling process. When there are changes in the real world, the corresponding modifications can be done in the data vault model simply by creating new links.

A link table referring to another link table should not be modeled. Such a link to link structure can be resolved by creating a link to have all hub surrogate keys from both links. Database foreign key information is used in code generation, which is the reason why all link to link relations is forbidden.

3.3. Principles for a Satellite Entity

A satellite is an entity which holds descriptive information of a hub or a link. It includes other attributes than those that comform a business key for that entity. The referring attribute to a hub or a link is a surrogate key.

We list all common satellite attributes in Table 2. Satellite tables have two additional columns compared to a hub or a link table. Datahash is special attribute, which is used for capturing changes in a satellite [6, Chapter 11.2.5]. The Datahash value is calculated from the concatenation of business key and all satellite data attributes. There might be a need in future to have multiple active values in a satellite, we added in every satellite record order for that purpose.

Table 2. Satellite table common attributes

`DV_ID`	Surrogate key
`DV_LOAD_TIME`	Timestamp when the row is inserted
`DV_SOURCE`	Metadata of the source system
`DV_LOAD_NAME`	Name of the transformation which inserted the row
`DV_RUN_ID`	Every load batch has a unique id
`DV_DATAHASH`	Datahash is computed of business keys and data attributes
`DV_RECORD_ORDER`	This is for multiple active satellite rows

Satellite information is subject to change over time [5]. A satellite will store the whole history of the descriptive data. To help querying the satellite we have created current view, which show the latest known data row for each surrogate key. When populating the satellite table the current view can be used. New row is added from staging table to satellite when the data in staging table differs from data in current view, the comparison is done by comparing the datahash attribute values in staging table and satellite current view.

There are modeling principles regarding how to split the satellite attributes to several satellite tables [2, Chapter 21], but how the split is done is not relevant for generating transformations to satellite tables.

Status satellite has a special meaning A status satellite tells the time intervals when the referenced row has been active. If the staging table is a full dump from a source system, the deleted rows can be recognized. If it is an incremental dump for a link, information about a driving hub should be available. The driving hub information is used to identify changed links.

3.4. Principles for a Reference

Reference data holds a list of code values with their descriptions. Typical items are units of measure, postal codes, currencies, and so on. The code can be in several places in a data vault, and it is not reasonable to copy the same descriptions to several places. A reference – a different kind of entity from other major data vault components – refers to other tables in the logical sense, but it will not have any foreign key references in the physical model. To mitigate new transformation for reference the reference implementation can be changed to view which is based on a hub and a satellite table. Then the reference transformation is splitted to hub and satellite transformations, there is no need to have different transformation rules for the reference.

3.5. Transformation

A transformation phase in ELT is in itself a kind of a transformation of its own, this time of type ETL. One transformation between staging and data vault table is extracting rows from the source table, transforming them, and then inserting the chosen columns of the rows to the target table. All of these steps may be done in a single insert into select statement.

A transformation is a data flow element at entity level. When a data flow is coming from a staging table to a data vault structure, each source entity - target entity constructs a transformation.

Attributes in a Transformation A transformation at the attribute level is constructed using mappings. For a normal attribute, such mapping is simply a relation from a source attribute to a target attribute for raw data vault transformations. There should not be any attribute level transformations in the mappings. No data cleansing is to be done in this phase of data warehouse population. Consequently, both source and target attributes in a mapping should have matching data types. The business keys in hubs are used for consolidation. This is why it is suggested to trim business key columns [6, Chapter 11]. This could lead to a need for technical hubs that store different presentations of the business keys. The original presentation might be stored in a satellite as a normal attribute. This way, dirty data in source systems could lead to a need for misusing multi-valued satellite implementation. For the hub transformations, the whole business key should be mapped. In other words, if the hub has several columns in its business key, there should be a mapping for all of its key columns in each transformation that will be populating the hub. The attributes are mapped from a source staging table columns to target data vault entity columns. It is equally important to map the access paths for columns used in foreign keys. Surrogate key columns are used in foreign keys in the data vault. To define a transformation for a satellite surrogate key, a data transformation from the same staging table defines which business key columns are used to look up or construct the surrogate key. The same rule applies to link referential hub surrogate key columns.

Defining Transformations for a Hierarchy Parent-child relationships in the model introduce a hierarchy link in the data vault. Examples of such are product and account hierarchies. In these cases, several transformations from the same staging table to the same hub exist, and at least one to a link that defines the hierarchy. Therefore, transformation naming needs more information than just source entity and target entity names.

4. Automate Transformation

An agile approach to develop transformations is by slicing the model. One way to divide work to get smaller deliverable is to develop by source system dataset.

Current ETL software have lack of parallel development. The development design happens in the same server and developers have to communicate that they do not change the same ETL object at the same time. Development with automating transformations can be separated. Each developer has their own design environment. A developer creates the implementation of transformations and the implementation is shared among other developers via version control system.

4.1. Information Model for Automated Transformation – Data Map

For automating transformations, we need information of source structure and target structure. The power of the data model in Figure 1 is that it represents model information and mapping between different models. Automating the transformations is based on the structure (model) and data flow information.

Structure information is in Figure 1 with green color. An entity is in a relational database table or view, but it can be something else if the data warehouse is not in a relational database. An attribute is column information, each entity has several attributes. Reference is foreign key relations at attribute level.

Data flow information between entities is a transformation. In the model, a transformation has a source entity and a target entity. A mapping is transformation information at attribute level. A mapping information is entered for each attribute for a target entity. Figure 1 shows the data flow information at violet color.

Populating the Structure The data model may be populated from a existing relational database metadata. While populating an entity and querying the table information from a database metadata also the reference and attribute level information may be queried. Based on the found information the model may be populated. An entity may become from a table or a view. A staging entity may be implemented as a view. The rules of the data vault structure may be used to populate the structure part. The entity type may be decided based on the rules. Also if there is a satellite in the model there has to be a hub or a link that the satellite belongs to. Similarly if there is a link there should be at least two hubs in the model that the link is referring to. So if the model is populated from a database metadata the model population may be started from satellites. Traverse through foreign key definitions and populate the entity, attribute and reference information on the way.

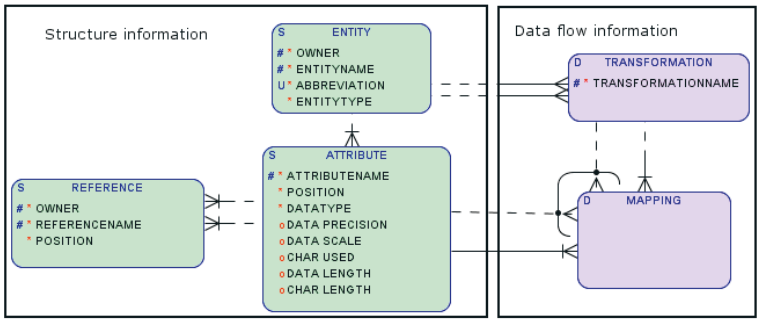

Figure 1. The data map model

Populating the Data Flow The transformation level is chosen first. This means choosing from which staging entity to a data vault entity there will be a transformation. Similar rules like populating the structure part need to be used. If there is a transformation from a staging table to a satellite, there has to be a transformation from the same staging table to a dependent hub or link. The same rule applies to a link transformations. All dependent hubs should have a transformation from the same staging table that is transforming the link. Mapping at an attribute level may be suggested based on the similar naming of the attributes. Another way to suggest the mapping information is using the attribute order of a source entity. Also similar data types should be checked to avoid implicit data type conversions. After the model is enriched with several mappings the mapping table constructs a dictionary, which is used to suggests the mappings in other transformations.

4.2. Generating Transformations

We have created a data map data model capable of automating transformation generation. Data map enables more than just transformation generation.

Data map model can be used as a source to generate create table clauses. It can be used as a primary definition place to define a data warehouse data model. This would support data first development strategy. We have currently chosen the model first development strategy where the data warehouse model is drawn first with a suitable tool for the job. Those tools support the features of the environments that the data vault model will be installed to.

As the data vault is the place to store a history of data changes the latest knowledge of the data may be published through views that publish a latest known rows, we call these views current view. Current views is possible to generate based on structure information and rule that the latest known row will be displayed. The history is stored mainly in satellites. For a satellite the current view publishes the latest rows for each surrogate key. These satellite current views are used in data map insert views generated in the next steps. Also a relationship may be changed

in source systems. So there is a need to make links disapear in certain points in time. This is done with a status satellite for a link. In link current view a link rows that are not marked as deleted in the latest status satellite are published. A link without a status satellite may be published as is as a link current view. The view is created for future changes in the model. A link status satellite may be created afterwards for the link. The view may be changed to take the status satellite into account. By creating such a link current view the interface to the data vault for user stays similar even when future changes appear. Point in time views are somewhat similar to current views. A point in time table is a query assistant table [4, Chapter 7] . They publish the transaction time aspect history of the tables. A point in time view is showing the hub surrogate key and satellite transformation times related to that hub for every snapshot date chosen to point in time view. This generation needs only the structure part of the data map.

Also basic publishing views may be generated based on the structure. A link publishing view could join the used hubs and publish the used business keys in a view. A current satellite may be published together with the business keys in the related hub. Also supernova views may be generated [10]. Supernova modeling technique generates views on top data vault modeled structure and reveals data to the reporting layer.

A ETL transformation could be done by simply with a single insert into clause. That could be generated from data map. We have chosen to split the single SQL statement into a view and a function. These insert views and functions are generated for each ETL transformation. The split is done for better testability.

The insert view publishes the new rows from the staging table compared with the latest rows in the data vault. The insert view is using earlier generated current views for satellites. For hubs and links the comparison is made straight between the staging entity and target table. The insert view publishes distinct rows from the source.

The insert functions include the insert into select clauses. Insert is into the target table and select part is using the generated insert views. The function is returning the number of inserted rows. Inside the generated function it is possible to do error handling code. Unique key constraint violation can be handled in the functions if there happens to become several runs of the ETL transformations in parallel.

How a data vault 2.0 style surrogate key hash is generated may be interpreted from the data map. Several staging tables may be used to populate the same hub. The used order of the columns in hash generation should be taken from the order of the target hub attribute ordering. The access path to the ordering of the staging table columns goes through the mapping table. Data hash generation is based similarly to the target table column ordering than the surrogate key hash generation.

4.3. Developing with Automated Transformation

Here is a list of steps when developing with automated transformation.

1. Model and create the new data warehouse entities with case tool
2. Populate structure information from database metadata

3. Populate data flow information
4. Generate objects based structure and data flow information
5. Testing in development environment
6. Create delivery scripts from development to other environments like test and production

First step is to make the data warehouse data model changes with a case tool. When the data model is modeled, it will be created with ddl-statements to a relational database. The end result of the modeling will be ddl-statements and model information in the relational database.

Structure is populated from database metadata of all entities which structure information is needed in the implementation. Structure population use structure rules and populate all dependant entities. When populating link or satellite structure information check is the dependent objects structure information populated and populate those if needed. In Listing 1 the structure population is on row 1.

Listing 1 Pseudocode commands for automating transformations

```
1  populate_entity('<insert entity here>');
2  populate_transformation('<source table>','<target table>');
3  create_current_views();
4  create_insert_views_and_functions();
```

Data flow information population needs manual acceptance. Data flow information is populated at entity and attribute level. Entity level data flow information is a transformation and it can be given with command like in Listing 1 at row 2. transformation information need information on source table and target table. Mapping information population can be suggested based on either attribute names or position of the attributes in a source table.

Automating work is generating the deliverable without human work. After the structure and data flow information is in the data map model all transformations can be generated. In generation one command generates all objects, compared with manual work where each object has to done manually. Generation can also be done one by one by specifying what an object should generate.

In parallel development testing has to be part of the development. Test cases are stored in version control and after each development cycle a developer will execute the test cases. We stress the importance of local testing in parallel development.

After testing in the local development is passed then the implementation is locally ready. In parallel development there has to be version control in use and release management practices. Create the delivery script from local environment by following the release management practices.

5. Data Map Usage Example

Next, we provide a small example, where data from several source systems is stored in a data vault data warehouse. There are three different source systems –

customer relations management (CRM), supply chain management (SCM), and product life cycle management (PLM) systems. The data model of this example is given in Figure 2. Tables are classified by using colors; white tables are staging tables, and blue, red, and yellow links represent data vault tables, links, and satellites, respectively.

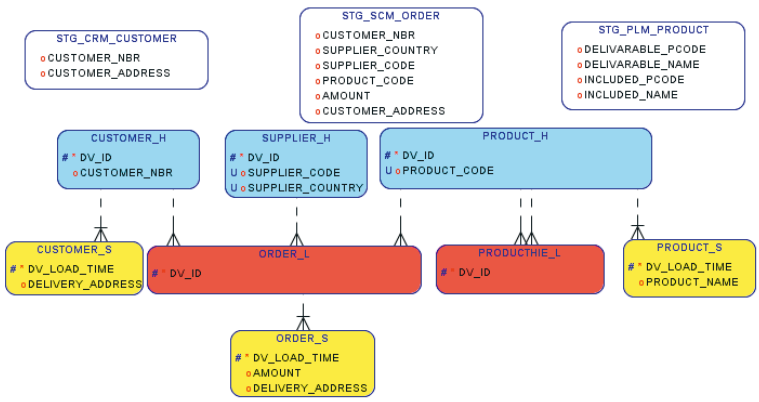

Figure 2. Customer - Supplier - Product example

CRM Populate Structure The source is `STG_CRM_CUSTOMER` and targets are `CUSTOMER_H` and `CUSTOMER_S`. We populate the structure information with the following commands. After the structure information population we create the current view for the satellite.

```
1 /* populate structure */
2 populate_entity('STG_CRM_CUSTOMER');
3 populate_entity('CUSTOMER_S');
4 /* create current views for satellites */
5 create_curr_views;
```

CRM Populate Data Flow Populating data flow information for CRM can be done with one command.

```
1 populate_transformation('STG_CRM_CUSTOMER', 'CUSTOMER_S');
```

Based on the model rules, transformations from `STG_CRM_CUSTOMER` to `CUSTOMER_S` and `CUSTOMER_H` are created. In addition, the mapping for surrogate key attribute is populated for these transformations.

For transformation attribute level population, data map suggests population code. Based on similar naming, the `CUSTOMER_H.CUSTOMER_NBR` is sug-

gested correctly. CUSTOMER_ADDRESS needs to be chosen to be mapped to CUSTOMER_S.DELIVERY_ADDRESS. Missing attribute mappings for populated entity level transformations may be queried from the data map.

Code generation for insert views and functions is possible after the data flow is populated into the data map. Insert views and functions are generated for both transformations from STG_CRM_CUSTOMER to CUSTOMER_S and CUSTOMER_H.

PLM Populate Structure To populate PLM entities use the following commands.

```
1 populate_entity('STG_PLM_PRODUCT');
2 populate_entity('PRODUCT_S');
3 populate_entity('PRODUCTHIE_L');
```

The dependant PRODUCT_H is populated on row 2 based on model rules when PRODUCT_S is populated. The link PRODUCTHIE_L population is satisfied with the earlier population of the hub because the row 2 populate the hub. If the order in populating script would be different between PRODUCT_S and PRODUCTHIE_L the hub PRODUCT_H would be populated in that entity which is executed first.

Based on the populated data map structure part, a PRODUCT_SC current view for the satellite may be generated.

PLM Populate Data Flow Transformations from STG_PLM_PRODUCT to PRODUCT_H, PRODUCT_S and PRODUCTHIERARCHY_L will be generated. There will be two transformations to both PRODUCT_H and PRODUCT_S. One pair for staging table included and deliverable columns.

```
1 populate_transformation('STG_PLM_PRODUCT','PRODUCTHIE_L');
2 populate_transformation('STG_PLM_PRODUCT','PRODUCT_S');
3 populate_transformation('STG_PLM_PRODUCT','PRODUCT_S');
```

Row 1 generates two transformations to PRODUCT_H and links the surrogate key mapping for PRODUCTHIE_L attribute to foreign key reference.

In row 2 there already is a hub transformation for PRODUCT_H from the same STG_PLM_PRODUCT table. The surrogate key reference mapping cannot be done automatically. From data map, it is possible to create suggestions that there are two candidate hub transformations available for this satellite transformation. The user may choose the hub transformation that is used in deliverable part of the transformation. Based on that the user maps the attribute level mapping from DELIVARABLE_NAME to PRODUCT_NAME.

Row 3 will generate a satellite transformation for the included_name mapping. Also here the surrogate key mapping needs to be chosen and the INCLUDED_NAME to PRODUCT_NAME need to be mapped at attribute level.

The mapping of data map table is now populated for STG_PLM_PRODUCT source entity transformations, which is visualized in Table 3.

SCM Populate Structure SCM entities are populated with commands.

```
1 populate_entity('STG_SCM_ORDER');
2 populate_entity('ORDER_S');
```

Table 3. Mapping table contents for STG_PLM_PRODUCT source entity transformations

TR	TARGETENTITY	TARGETATTRIB	SOURCEATTRIB	DVIDTR
PPD4	PRODUCT_H	PRODUCT_CODE	DELIVARABLE_PCODE	
PPD5	PRODUCT_H	PRODUCT_CODE	INCLUDED_PCODE	
PPL3	PRODUCTHIE_L	DV_ID_PROD_DELI		PPD4
PPL3	PRODUCTHIE_L	DV_ID_PROD_INCL		PPD5
PPS6	PRODUCT_S	DV_ID		PPD4
PPS6	PRODUCT_S	PRODUCT_NAME	DELIVARABLE_NAME	
PPS7	PRODUCT_S	DV_ID		PPD5
PPS7	PRODUCT_S	PRODUCT_NAME	INCLUDED_NAME	

Entity `ORDER_L` is populated based on the model rules. If the previous CRM and SCM parts are already in the data map model, the `CUSTOMER_H` and `PRODUCT_H` entities are populated already into the model. If the SCM part is developed privately without first populating other parts, also `CUSTOMER_H` and `PRODUCT_H` entities are populated. This way it is possible to develop transformations in parallel. Now, current views for the link and the satellite are ready to be generated.

SCM Populate Data Flow Transformations from `STG_SCM_ORDER` to `CUSTOMER_H, SUPPLIER_H, PRODUCT_H, ORDER_L` and `ORDER_S` will be created with one command.

```
1  populate_transformation('STG_SCM_ORDER','ORDER_S');
```

All other attributes except `CUSTOMER_ADDRESS` match with the naming so data map suggest the mappings. If the CRM part is already stored in the mapping table, also the mapping from `STG_SCM_ORDER.CUSTOMER_ADDRESS` to `ORDER_S.DELIVERY_ADDRESS` may be suggested. This is based on the previous usage of similar naming in `STG_CRM_CUSTOMER` to `CUSTOMER_S` transformation. Artifacts for all five transformations are now ready to be generated.
`SUPPLIER_H` has two column business key. If the hash primary keys are used, the column ordering is taken from the order of the hub column order. Even thou the ordering of the columns is different in the staging table.

6. Discussion

In this paper we show how to generate code for ELT transformations from a staging table to a data vault table. This is enabled by the presented data map, resulting from the introduced model for storing structure and dataflow information. The power of the data map is in using both structure and data flow information instead of relying to only one of them. It is possible to make also changes by using data map, although this is not its initial purpose. Instead, there are various case tools available for managing the structural information. The tools also enable structural changes in a flexible fashion. Moreover, such tools can also store data flow information. Nevertheless, the data flow information is only presented graphically, and it is not possible to extract out of the tool easily in a similar way as with our solution data model.

In addition to the above tools, there are ETL tools that are intended for transformations. Such ETL tools include information similar to our data map.

These ETL tools often introduce a vendor lock-in situation, and work done using one tool is not modifiable with other tools. Consequently, internal models cannot be exported from currently available tools. Nevertheless, there are some ways to tackle the vendor lock-in. For example, [1] provides BPMN4ETL design model for ETL processes which allows transporting between several technologies. In BPMN4ETL a ETL process is designed manually, we are creating transformations based on data vault principles. In theory, BPMN4ETL could be generated based on data map model information.

Even if data map could be developed in data first fashion, we chose to use model first approach. Model first approach create data warehouse structure which is more timeless than data model created based on source systems models. These strategies have different ways regarding how the data map is populated. The structure information in data maps enables generating statements that create tables for a data warehouse database model, which is very convenient when develop in data first fashion.

It is worth noting that the staging table columns may include null values, which are then populated to hubs and satellites. Initially, all values are unknown as there are no rows in the target model. In case of satellite tables, null values override possible previously known values, whereas in hubs null values should actually be stored in a data vault. There is also a common usage pattern to replace nulls with some fixed characters, like -1 for an example. However, this character is then not available for any other use. We have chosen to allow null values as a business key and tie that to a hash binary that is not produced from actual data. In this case, the hash of null value is defined to be also null.

As described earlier, it is possible to generate the transformations by using both the existing data warehouse data and data flow information. At the moment, we generate the following items based on data map data:

- data warehouse create table statements,
- current views for satellites,
- current views for links,
- insert views for all data vault tables and
- transformation functions which use insert view and have error handling for parallelizing the ETL loads.

In addition, we can also generate other interesting structures from on the data map. Most of the ETL tools can read XML formatted transformation specifications. We can generate the specifications with data map information in XML format. This way we can actually execute the transformations with ready ETL tools.

Data flow information includes data lineage information which can be presented graphically with different tools, like DOT[5] language.

ETL has been historically based on batch load operations. Nevertheless, there is an ongoing trend towards online processing, which requires streaming the data. In this case, we do not have separate extract or loading phases in ETL, but the whole process is about transformation. Consequently, the data map approach is readily available for streaming transformation generation as well.

[5] http://www.graphviz.org/doc/info/lang.html

Finally, it is important to consider the tradeoff between manual work and work needed for automating the tasks. Obviously, the automation effort should be placed on development steps that require a lot of manual work. Nevertheless, automation also introduces other benefits. The resulting code systematically similar for all generated objects. The coded objects are therefore often easier to maintain, even if manual work is required.

7. Future Work

Generating data transformations out of a data vault will be the next long-term goal of this work. The first step is to implement an inverse transformation of a staging interface, so the data vault model may be published with a similar interface as is used when storing the data.

As described and discussed in this article, transforming data to a data vault structure splits the entities to smaller groups of columns. This way a single executable transformation takes place between a staging table and a data vault table. Mappings between source and target columns are stored in data map data model, with information of access paths to resolve surrogate key values in foreign keys. The introduced data map data model in this article supports the table splitting transformation. When getting data out of the data vault, the entities are put together. Such join transformations would consist of several data vault tables as the input. The data map model described in this paper cannot store such information. The model needs some minor changes to accomplish these requirements.

8. Conclusion

In data warehousing, data vault modeling is widely used methodology. The data is divided into four different entities, which are hub, link, satellite, and reference. A data warehouse that is modeled using data vault modeling technique can be flexibly modified if the source systems change. As a cost of the flexibility, we have significantly increased the amount of data load operations and transformations, which requires reducing the amount of manual work involved in these phases.

Fortunately, the manual work for the data transformation development can be mitigated. We note that data vault methodology gives certain principles regarding how the different entities should be populated. By using these principles, the metadata of data models, and data flow information it is possible to semi-automatically generate the actual data transformations. To support this, in this paper we have introduced a metadata model which allows the transformations code generation.

In our approach, the amount of needed manual work is reduced but not completely eliminated. Nevertheless, even for the manual work phase, with data flow information we can give good proposals based on the data map metadata. As there are several exceptions which are easy for human to recognize to instruct to a machine, the proposed data flow information needs at times to be accepted by a human.

Using data model information in generating data transformations introduces several advantages. By reducing manual work, we also reduce the possibilities for human errors. Transformations can also be generated as large sets, whereas manually implemented transformations have to be done one by one. Furthermore, transformations generated from data map are readily available for commercial production environments.

Acknowledgements

The authors would like to express deep gratitude to Janne Pirkkanen for his encourage to think differently, he was the person who have inaugurate the systematic way of doing data warehousing.

The work was financially supported by TEKES (Finnish Funding Agency for Innovation) DIGILE Need for Speed program. We would also like to thank Solita for the possibility to do this research.

References

[1] Zineb El Akkaoui, Esteban Zimànyi, Jose-Norberto Mazón, and Juan Trujillo. A model-driven framework for etl process development. In *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP*, pages 45–52. ACM, 2011.

[2] Hans Hultgren. *Modeling the Agile Data Warehouse with Data Vault*. New Hamilton, 2012.

[3] Vladan Jovanovic and Ivan Bojicic. Conceptual data vault model. In *SAIS Conference, Atlanta, Georgia: March*, volume 23, pages 1–6, 2012.

[4] Dan Lindstedt and Kent Graziano. *Super Charge Your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault*. CreateSpace, 2011.

[5] Dan Linstedt. Data vault series 1–data vault overview. *The Data Administration Newsletter*, 2002.

[6] Dan Linstedt and Michael Olschimke. *Building a Scalable Data Warehouse with Data Vault 2.0: Implementation Guide for Microsoft SQL Server 2014*. Morgan Kaufmann, 2015.

[7] Dan Linstedt, Kent Graziano, and Hans Hultgren. The new business supermodel, the business of data vault modeling. *Lulu. com*, 2008.

[8] Ivan Pankov, Pavel Saratchev, Filip Filchev, and Paul Fatacean. Towards a generic metadata warehouse.

[9] Cassandra Phipps and Karen C Davis. Automating data warehouse conceptual schema design and evaluation. In *DMDW*, volume 2, pages 2–2. Citeseer, 2002.

[10] Rick F. van der Lans. *Data Vault and Data Virtualization: Double Agility*, March 2015 (accessed 4 january 2016). URL `https://www.cisco.com/web/services/enterprise-it-services/data-virtualization/documents/whitepaper-cisco-datavaul.pdf`.

PUBLICATION IV

"Data Vault Mappings to Dimensional Model Using Schema Matching"

Mikko Puonti and Timo Raitalaakso

Research and Practical Issues of Enterprise Information Systems 375 (Dec. 2019), pp. 55–64
DOI: 10.1007/978-3-030-37632-1_5

Data Vault Mappings to Dimensional Model Using Schema Matching

Mikko Puonti[1,2][0000-0001-6579-0224] and Timo Raitalaakso[1,2][0000-0002-7094-7203]

[1] Solita Ltd, Åkerlundinkatu 11, 33100 Tampere, Finland
[2] Tampere University, Kalevantie 4, 33100 Tampere, Finland
puonti@iki.fi, timo.raitalaakso@iki.fi
https://www.solita.fi/en/

Abstract. In data warehousing, business driven development defines data requirements to fulfill reporting needs. A data warehouse stores current and historical data in one single place. Data warehouse architecture consists of several layers and each has its own purpose. A staging layer is a data storage area to assists data loadings, a data vault modelled layer is the persistent storage that integrates data and stores the history, whereas publish layer presents data using a vocabulary that is familiar to the information users. By following the process which is driven by business requirements and starts with publish layer structure, this creates a situation where manual work requires a specialist, who knows the data vault model. Our goal is to reduce the number of entities that can be selected in a transformation so that the individual developer does not need to know the whole solution, but can focus on a subset of entities (partial schema). In this paper, we present two different schema matchers, one based on attribute names, and another based on data flow mapping information. Schema matching based on data flow mappings is a novel addition to current schema matching literature. Through the example of Northwind, we show how these two different matchers affect the formation of a partial schema for transformation source entities. Based on our experiment with Northwind we conclude that combining schema matching algorithms produces correct entities in the partial schema.

Keywords: Schema matching · data flow · data warehouse · data vault · dimensional model.

1 Introduction

In a data warehouse, whereas several data sources are integrated as one data set, mapping information is crucial. Most commonly used in data warehouse implementation is Extract- Transform and Load (ETL) process [6].

Business driven development defines data requirements to fulfill reporting needs. These reporting needs are typically modelled with a dimensional modeling [7] technique. To enable parallel work with data transformation (ETL) creation and reporting tools we have created a dimensional model as a prerequisite for actual implementation [11]. Reporting tools need a dimensional model populated with a sample data set.

Populating a sample data set to a dimensional model creates data flow mapping information at attribute level, whereas one or many attributes are mapped to one target attribute. As a new interface is introduced to a data warehouse, it is created first to staging layer. A sample data set is mapped from staging layer to the publish layer (Figure 1 B). Mapping may be implemented as a database view or a ETL-transformation that populates tables. In this paper, we are referring to these views and tables as entities.

A data warehouse stores current and historical data in one single place. A data vault model [8] is used for storing history. When the data vault model exists, the transformations implementation between staging layer and data vault is automated by using the process presented in our earlier research [12].

By following the process which is driven by business requirements and start with designing a dimensional model with data, continuing with transformation implementation from a staging layer to a data vault model. The transformation graph forms a data flow. This creates a situation where we have in a place publish layer structure, data vault model and transformation mapping between a staging layer and a publish layer together with a data vault (Figure 1 C). Transformation mapping between a data vault and a publish layer is missing. When producing a durable implementation, a target is to create transformation to the publish layer based on the data vault model. Currently, this is manual work and it requires a specialist who knows the data vault model. A large data warehouse may consist of several hundred entities. Our research is focused on how to help this transformation mapping creation. How may we use the data flow mapping information, which is generated in earlier phases? Is there any particular schema mapping technique useful to solve this manual work and at least partially automate tasks in the current situation? Our goal is easing up the developers work. This is accomplished with finding candidates to be used in schema mappings. We are also finding ways to prune parts of a big data model to be used.

2 Related Work

Ontology matching is a wider research area than our schema matching. We are using ontology matching a name based technique where strings are identical [4]. Our ontologies are database schemas, even when the actual implementation not include a database schemas there are structures where tables (relation) contain attributes.

We are using existing data flow mapping information to generate new replacing transformation mappings together with more commonly researched schema matching methods. The data flow forms an directed acyclic graph. It may be considered form a computer program. It describes the dependencies between all entities in a system. Frank Tip is writing about using program slicing in program integration [14, chapter 5.2]. We are using such slicing to generate subgraphs assisting transformation generation.

Villányi describes schema matching techniques in service-oriented enterprise application integration in his dissertation [15]. In a hybrid matcher Villányi combine a vocabulary matcher and a structural matcher, where structural matcher uses a neighborhood level structural similarity.

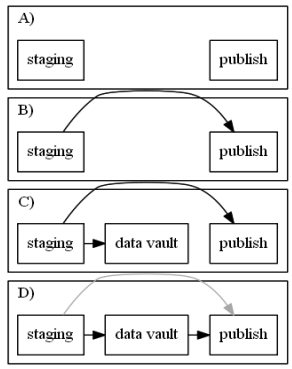

Fig. 1. Transformation stack evolution

Paolo et al. introduce meta-mappings as a formalism that describes transformations between generic data structures [2]. This enables mapping reuse, when similar information is located in several schemas, whereas our reuse is use data flow mapping for creating new transformation between schemas where transformation is not yet defined.

Golfarelli et al. [5] introduce a starry vault approach to generate a dimensional model automatically from a data vault model. In their paper there are formal definitions of data vault and multi dimensional schemas introduced. Their paper is aiming to find a multi dimensional model from data vault structures. Our approach is to match and generate data flow between two predefined models.

Human effort is needed in schema matching scenarios as existing matching algorithm results are not perfect. Nguyen et al. concentrate on minimizing human effort in reconciling match networks [10]. They stress that after matching there is still a need for a post-matching phase, which is manual correction. Their reconciliation process is an iterative process, whereas our solution is to offer a partition schema for transformation as a selection. Many authors agree that mapping can not totally automate, there is a need for manual corrections [1-2, 5, 10, 15].

3 Schema Matching and Experiments

In a data warehouse data is in relational form [3], even when NoSQL techniques are used in implementation. A relational database consists of tables and attributes. A set of tables is grouped together with a schema. Schema is used as an implementation of data warehouse layers, each layer in Figure 1 is a separate schema. Now we can refine our research question ”how to help transformation mapping generation” to form a match schema between a data vault and a publish layer schema.

3.1 Matching Workflow

In a matching workflow, we are using phases introduced E. Rahm [13]. First phase is preprocessing, where metadata information is extracted from a relational database (Figure 2 a). Relational schema offer metadata: a table name, an attribute name, the attribute data type, additional information for data type and a description field for attribute.

Matching is an execution of a matching algorithm, whereas it can contain several matching steps. These matching can be sequential, parallel or mixing both of those principals.

A combination of matcher results is combining different matcher algorithm values and possibly calculation aggregated value of those values. In a sequential matching a matcher can use earlier matcher values in an algorithm.

A selection of correspondences is in our case a human work phase, which we aim to help with offering matching results in use. If there is a tool built based on our article, it could suggest good matches and human work would be only accept suggestions and creating more complex mappings.

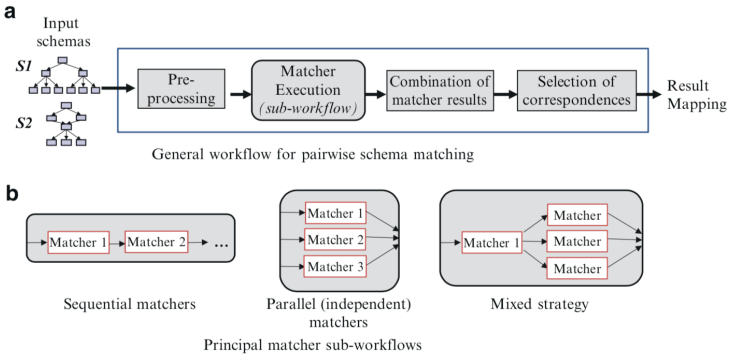

Fig. 2. General match workflow (copied from [13])

3.2 Schema Matching Based on Attribute Names

A matcher compares every attribute name of a data vault with every attribute in a publish layer target attribute. This cross join operation can be easily quite large and this is reason why we suggest to do this few publish layer entities at a time. The preprocess phase in Figure 3 target subset is chosen. In matcher first pruning is to feed only a partial entity set from the publish layer entities, this may be interpreted as a partition of a second schema [13].

A result of the attribute name matcher is a set of data vault entities, which has common attribute naming compared between data vault and target entities.

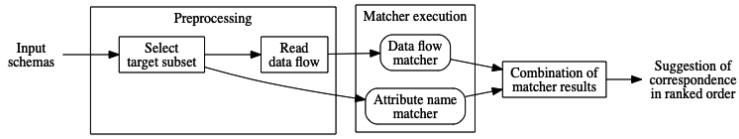

Fig. 3. Match workflow

3.3 Schema Matching Based on Data Flow Mapping

Publish layer entities have data flow mapping for a sample data set. This mapping can be implemented from the data vault or the staging area, nevertheless there is data flow mapping information as presented in Figure 1 B.

Data flow mappings are expressed at attribute level between source and target schemas. Depending on technology used it may be a challenging process to extract that attribute level mapping information, or even worse manually create this mapping information. We suggest expressing data flow information between source and target schema entities, this information is useful and it is easier to extract from ETL-tool or database view metadata information.

The target publish layer entities are chosen as an end point of the subgraph slice. A result of data flow mapping is data vault (source schema) entities, which have corresponding data flow to a publish layer (target schema) entities.

Inside data vault there might exist layers. A raw data vault that is populated straight from staging area. Business data vault [9] is a layer that enriches the data vault model and uses other data vault entities as a source. This piles up the transformation stack and makes the data flow graph deeper. We are using this depth as an indicator of enriched information. Giving a better ranking for business vault entities to be used in the suggested mappings.

3.4 Schema Matching Combination of Attribute Names and Data Flow Mapping

Last phase of our algorithm is a combination of earlier matchers results. Noteworth, these matchers have result sets at different level of granularity. This is presented in Figure 3 combination of matcher results.

As we are aiming to match schemas between the data vault and the publish layer, this algorithm is for helping creating transformations between these schemas. For human decision, we are presenting potential entities for transformation creation. The earlier matchers enable us to use the following strategy:

- Present all potential entities in order where first is the most prominent candidate
- Present only potential entities, which are common in both matcher result set

As the data flow information is not always available, our suggested strategy is to present all potential entities in relevant order.

The algorithm to create this ordering for candidate entities.

1. Count at entity level how many attributes is in an attribute name matcher result set.
2. Add a depth value for each entity, which is in a data flow matcher result set.
3. Summarise these result sets.
4. Order the result set according to the value of each entity.

3.5 Northwind Example

As a demonstration of our approach, we use Northwind[1] source data model. We modeled a publish layer schema of order fact and dimensions. The ORDER F references to CUSTOMER D, EMPLOYEE D, ORDER D and SHIPPER D. After data vault modeling and transformation population at the phase (Figure 1 C) we get suggestions to new (Figure 1 D) phase transformations as described in (Table 1) and (Table 2).

CUSTOMER D gets side different false positives from both suggestions but CUSTOMER H and CUSTOMER S are found in both sets to be considered as source for mappings. The number of common columns in naming suggestion is higher for these and gets prioritized based on the naming match. Adding the second suggestion set from data flows the results become more convincing. SHIPPER D gets assurance that CUSTOMER H and CUSTOMER S should not be considered as source for mapping. ORDER F gets suggestions from either ORDER L or ORDER BV L. Data vault model is layered. ORDER L represent a raw data vault layer and ORDER BV L is an entity of business data vault layer. ORDER BV L uses another data vault entities as a source for it data. This dependency graph depth is visible in (Table 2) ORDER BV L - ORDER F suggestion row. It is used together with higher score from naming suggestion to choose correct mappings to be implemented.

3.6 Observations from Northwind Example

Both our matchers return side hits - false positives. Attribute naming return overlapping from irrelevant similarity matches. The data flow matcher raises other potential mapping candidates. As the data flow subgraph from used staging entities contains transformations to other data vault entities that are not needed in the desired resulting mapping for a specific target publish layer entity.

With combing results from both approaches we get more precise suggestion for the new data vault publish layer transformations. With our experiments, the false positive groups are some what differing. So exclusion of false positive mapping candidates becomes more convincing. This minimizes the needed human effort while creating the end results.

[1] https://github.com/dshifflet/NorthwindOracle DDL

Table 1. Transformation suggestions based on naming

MAINENTITY	SOURCEENTITY	TARGETENTITY	C
CUSTOMER_H	CUSTOMER_H	CUSTOMER_D	2
CUSTOMER_H	CUSTOMER_S	CUSTOMER_D	2
SHIPPER_H	SHIPPER_H	CUSTOMER_D	1
SHIPPER_H	SHIPPER_S	CUSTOMER_D	1
EMPLOYEE_H	EMPLOYEE_H	EMPLOYEE_D	2
EMPLOYEE_H	EMPLOYEE_S	EMPLOYEE_D	2
ORDER_H	ORDER_H	EMPLOYEE_D	1
ORDER_H	ORDER_S	EMPLOYEE_D	1
ORDER_H	ORDER_H	ORDER_D	2
ORDER_H	ORDER_S	ORDER_D	2
CUSTOMER_ID_CUSTOMER_L	CUSTOMER_H	ORDER_F	1
CUSTOMER_ID_CUSTOMER_L	CUSTOMER_ID_CUSTOMER_L	ORDER_F	1
CUSTOMER_ID_CUSTOMER_L	CUSTOMER_ID_H	ORDER_F	1
ORDER_BV_L	CUSTOMER_H	ORDER_F	4
ORDER_BV_L	EMPLOYEE_H	ORDER_F	4
ORDER_BV_L	ORDER_BV_L	ORDER_F	4
ORDER_BV_L	ORDER_H	ORDER_F	4
ORDER_BV_L	SHIPPER_H	ORDER_F	4
ORDER_L	CUSTOMER_ID_H	ORDER_F	3
ORDER_L	EMPLOYEE_H	ORDER_F	3
ORDER_L	ORDER_H	ORDER_F	3
ORDER_L	ORDER_L	ORDER_F	3
ORDER_L	SHIPPER_H	ORDER_F	3
CUSTOMER_H	CUSTOMER_H	SHIPPER_D	1
CUSTOMER_H	CUSTOMER_S	SHIPPER_D	1
SHIPPER_H	SHIPPER_H	SHIPPER_D	2
SHIPPER_H	SHIPPER_S	SHIPPER_D	2

4 Discussion and Future Work

In our experimentation, we created a target schema as a database views from the staging layer. The data flow based matcher used this information at an entity level. There are possibilities to extract this data flow mapping information at an attribute level, one option is to use a tool like Queryscope[2]. This would open possibility to create more fine tuned result of the combined matcher described in this paper.

In this paper, we introduce two schema matcher. By adding more schema matchers, it is possible to improve the suggestions. A structural matcher might be beneficial. Link and fact granularities might be used. Link granularity, calculated based on the number of hubs it references, and fact cardinality, how many dimensions it references, could be compared.

Future work would be to suggest transformations between source schema (data vault) and target schema (publish layer) entities. At least the result set from the attribute name matcher is re-usable for creating transformation where is one-to-one mapping between source and target attribute. Our target is to reducing manual work by offering a subset of source schema entities for creating transformations, not actually create that data flow.

[2] https://app.sqldep.com/demo/

Table 2. Transformation suggestions based on data flows

SOURCEENTITY	TARGETENTITY	SCORE
CUSTOMER_H	CUSTOMER_D	1
CUSTOMER_ID_CUSTOMER_L	CUSTOMER_D	1
CUSTOMER_ID_H	CUSTOMER_D	1
CUSTOMER_S	CUSTOMER_D	1
ORDER_BV_L	CUSTOMER_D	2
EMPLOYEE_H	EMPLOYEE_D	1
EMPLOYEE_S	EMPLOYEE_D	1
CUSTOMER_ID_H	ORDER_D	1
EMPLOYEE_H	ORDER_D	1
ORDER_BV_L	ORDER_D	2
ORDER_H	ORDER_D	1
ORDER_L	ORDER_D	1
ORDER_S	ORDER_D	1
SHIPPER_H	ORDER_D	1
CUSTOMER_ID_H	ORDER_F	1
EMPLOYEE_H	ORDER_F	1
ORDER_BV_L	ORDER_F	2
ORDER_H	ORDER_F	1
ORDER_L	ORDER_F	1
ORDER_S	ORDER_F	1
SHIPPER_H	ORDER_F	1
SHIPPER_H	SHIPPER_D	1
SHIPPER_S	SHIPPER_D	1

This paper is talking about a process of creating new or extending an existing data warehouse. A similar approach may be used when replacing an existing direct star schema publish layer data flows by adding data vault modeled enterprise data warehouse layer between a staging and a publish layer. Replacing old ETL tool implementation. Benefits of data vault methodology such as history in satellites and better agile development enablement. The process described in (Figure 1) fits as is also on such replacement process. Phase A) Use existing staging and star schema model. B) Reverse engineer data flow transformation dependencies from old ETL implementation. C) and D) phases as described in this paper.

It is inevitable that the data vault model does not have a perfect match for source mapping at some point in data warehouse evolution. These pruned partial matcher results could be used to be a base for new business vault entities. The knowledge of a developer and an accespath suggestor[3] could be used to generate links and bridges that satisfy publish layer source information needs.

[3] http://rafudb.blogspot.com/2019/01/access-path-suggestor.html

5 Conclusion

Our research is focused on helping transformation creation between a data vault and a publish layer. For each publish layer entities we create a set of source entity candidates from a data vault schema entities.

As we are following the process which is driven by business requirements, there is a publish layer entity populated with a sample data. This sample data population construct a data flow to the target schema entity.

We examine whether schema matching is based on attribute names enough to suggest correct entities from a data vault schema. Schema matching based on attribute names finds correct entities from a source schema, but still there is room for improvement.

By adding a schema matcher based on data flow mapping we get a result set to enrich an attribute name based matching. Combining the results from these two matchers we may present potential transformation source entity candidates in order where the most prominent one is at first.

This combined algorithm is suitable for building a tool to help transformation mapping creation between a data vault and a publish layer. The result set from algorithm offer correct source entities for transformation mapping sources.

After choosing source entities for transformation mapping, we could additionally suggest mappings from schema matching based on attribute names as a base for that transformation where a specialist could continue with additional mappings and with the complex mappings.

References

1. Alexe, B., Ten Cate, B., Kolaitis, P.G., Tan, W.C.: Designing and refining schema mappings via data examples. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. pp. 133–144. ACM (2011)
2. Atzeni, P., Bellomarini, L., Papotti, P., Torlone, R.: Meta-mappings for schema mapping reuse. Proceedings of the VLDB Endowment 12(5), 557–569 (2019)
3. Codd, E.F.: A relational model of data for large shared data banks. Communications of the ACM 13(6), 377–387 (1970)
4. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer Science & Business Media (2013)
5. Golfarelli, M., Graziani, S., Rizzi, S.: Starry vault: Automating multidimensional modeling from data vaults. In: East European Conference on Advances in Databases and Information Systems. pp. 137–151. Springer (2016)
6. Kimball, R., Caserta, J.: The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming and Delivering Data. John Wiley & Sons, Inc. (2004)
7. Kimball, R., Ross, M.: The data warehouse toolkit: the complete guide to dimensional modeling. John Wiley & Sons (2011)
8. Linstedt, D.: Data vault series 1–data vault overview. The Data Administration Newsletter (2002)
9. Linstedt, D., Graziano, K., Hultgren, H.: The new business supermodel, the business of data vault modeling. Lulu.com (2008)

10. Nguyen, H.Q.V., Wijaya, T.K., Miklós, Z., Aberer, K., Levy, E., Shafran, V., Gal, A., Weidlich, M.: Minimizing human effort in reconciling match networks. In: International Conference on Conceptual Modeling. pp. 212–226. Springer (2013)
11. Puonti, M., Lehtonen, T., Luoto, A., Aaltonen, T., Aho, T.: Towards agile enterprise data warehousing. ICSEA 2016 p. 241 (2016)
12. Puonti, M., Raitalaakso, T., Aho, T., Mikkonen, T.: Automating transformations in data vault data warehouse loads. In: EJC. pp. 215–230 (2016)
13. Rahm, E.: Towards large-scale schema and ontology matching. In: Schema matching and mapping, pp. 3–27. Springer (2011)
14. Tip, F.: A survey of program slicing techniques. Journal of programming languages (3), 121–189 (1995)
15. Villányi, B.J.: Schema matching techniques in service-oriented enterprise application integration. Informatikai Tudományok Doktori Iskola (2016)