TU WIEN Informatics

Generating Automatic As-Built BIM Models In Conventional Tunnel Construction Lifecycle

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Bsc Dzan Operta
Registration Number 11935976

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Christian Huemer
Assistance: Univ.Ass. Dipl.-Ing. Marco Huymajer
Dipl.-Ing Robert Wenighofer

Vienna, 1st September, 2022

Dzan Operta	Christian Huemer

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Bsc Dzan Operta

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. September 2022

Dzan Operta

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I would like to thank my colleague Marco for helping me understand the topic idea and for his support in the thesis evaluation, mister Robert Wenighofer for providing me with the as-designed BIM model and explaining its elements, professor Christian for his wise feedback and time, and all the interviewers who invested their time to evaluate my thesis. Last but not the least, I would like to thank my family for their unconditional understanding and support.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Building Information Modeling (BIM)-Modelle gelten als ideales Werkzeug für die Darstellung und Verwaltung digitaler Informationen über eine Anlage in einem Bauprojekt. Im Bereich des Tunnelbaus werden BIM-Modelle verwendet, um einen Tunnel digital darzustellen. Ein Tunnelbauprojekt gliedert sich in drei große Lebenszyklusphasen, die Planungs-, die Bau- und die Betriebsphase, und BIM-Modelle werden in diesen verschiedenen Phasen als Planungs-, Bau- und Nutzungsmodelle charakterisiert. Derzeit werden bei konventionellen Tunnelbauprojekten sowohl As-designed- als auch As-built-Modelle manuell erstellt, da es keine automatische Möglichkeit gibt, As-designed-Modelle aus den Daten aus der Bauphase zu erstellen und anzureichern. Durch die Anwendung der Design Science Forschungsmethodik entwickeln wir einen Artefakt, der die automatische Erfassung und Übertragung von BIM-Modellwissen von der As-Designed- zur As-Built-Phase ermöglicht. Durch die Erkennung der Merkmale eines As-Designed-Modells und dessen Anreicherung mit Echtzeitdaten aus der Bauphase ermöglicht unser Prototyp die automatische Generierung eines As-Built-Modells.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Building Information Modeling (BIM) models are regarded as the ideal tool for representing and managing digital information about an asset in a construction project. Respectively, in the tunnel construction domain, BIM models are used to digitally represent an actual tunnel. A tunnel construction project has three major life cycle phases, the design, construction, and operation phase, and BIM models are characterized at these different stages, as as-designed, as-built, and as-used models. Currently, in conventional tunnel construction projects, as-designed and as-built models are both created manually, as there exists no automatic way to generate and enrich as-designed models with the data from the construction phase. By employing the Design Science research methodology we build an artifact that enables the automatic acquisition and transfer of BIM Model knowledge from the as-designed to the as-built stage. By detecting the characteristics of an as-designed model and enriching it with real-time data from the construction phase, our prototype enables the automatic generation of an as-built model.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

1.1 Problem definition

Building Information Modeling (BIM) is a broad term that encompasses creating and managing digital information about an asset in a construction project. BIM enhances the information management process by the collaborative use of semantically rich three dimensional (3D) digital models during the complete life cycle of a construction project [23]. One of the main aspects that BIM enables is the digital representation of assets through 3D digital models. Specifically, in the tunnel construction domain, an actual tunnel is represented digitally through a 3D model. This 3D model can be enriched with additional information from the construction project. One perspective is that each additional layer of information adds a dimension to the 3D model. For example, if we also store time information on top of the 3D model, or we enrich the model with additional information about the material used for the tunnel construction, we can refer to these models as 4D (time), or respectively 5D (material cost) models. Theoretically BIM models can be n-dimensional (n-D) [1].

A tunnel construction project has three major life cycle phases, the design, construction, and operation phase, and BIM models are characterized at these different stages, as as-designed, as-built, and as-used models [2]. Users may use BIM in each phase of a tunnel construction project, for a certain task, such as collision detection and settlement risk predictions in the design phase, productivity and decision-making improvements in the construction phase, and safety monitory and visualization in the operation phase [23, 26, 18].

However, at the moment, there is a BIM model development and usage interruption between as-designed and as-built models in tunnel construction projects. While a lot of time is invested in the creation of as-designed models in the design phase, these as-designed models are not used in the construction phase and as-built models are created

only after the construction phase finishes. Figure 1.1 shows the current standpoint of the BIM life cycle in conventional tunnel construction. The as-designed, as-built, and as-used models need to be manually created at each phase of the tunnel construction life cycle. This leads to many undesirable consequences [23, 2]. Currently, the as-built models are created manually with all additional information from the construction phase, which is a time-consuming activity [2]. Hence, due to a lack of timely feedback, project stakeholders are unable to make adequate, timed reactions to unexpected on-site events. The problem is that there exists no automatic way to enrich as-designed models with data from the construction phase. This approach would eliminate the need to create as-built models manually, which would save time, as well as reduce possible human errors in capturing important model-based information. Moreover, due to the interruption in the BIM model life cycle, a great deal of information is not preserved. Sequential transformations (e.g., translations, reductions, simplifications) from source to target models can lead to loss of information and possible integration issues in the source-target model exchange [2]. The loss of information reduces the possibility of making predictions and reusing the acquired information for future tunnel construction projects.

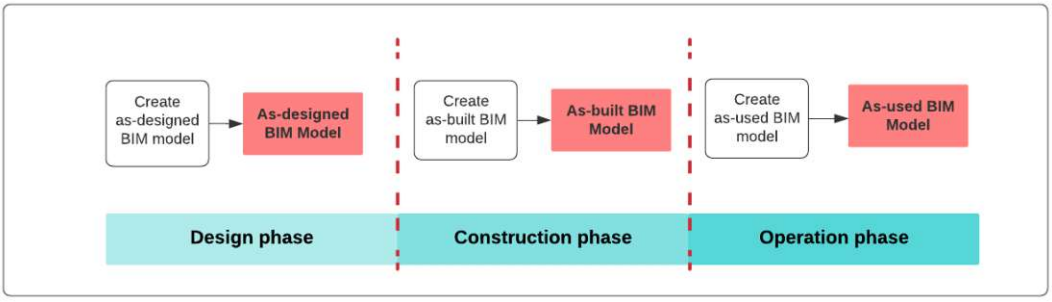

Figure 1.1: The current status of the BIM life cycle in conventional tunnel construction

1.2 Expected goals

This work aims to support the transfer of BIM Model information between the design and construction phase of the conventional tunnel construction life cycle and the automatic acquisition of construction data for the as-built BIM model. Figure 1.2 presents our solution to overcome the problem of the interrupted BIM life cycle. We develop a tool that automatically detects the 3D properties of provided as-designed 3D models, constructs as-built 3D models based on the detected properties, and finally enriches the newly created as-built models with real-time data from the construction phase of a tunnel construction project. As a result, by automating the as-built BIM model generation, we enable continuity and eliminate interruptions in the BIM life cycle.

Consequently, without interruptions and by enriching the BIM models with real-time data from the construction phase, project stakeholders are capable of overseeing and

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

coordinating the construction process at any point in time. This enables them to make timely reactions to unexpected on-site events. Next, this approach should persist model-based information between different phases of the construction project by making automatic model transformations. Fostering rich model-based information enables better prediction of future events, such as detecting possible unforeseen events in forthcoming tunnel construction projects.

In addition, we build a Tunneling Information Management System (TIMS), a tool for documenting the tunneling process in NATM[1] projects. TIMS enables continuous digitalization of tunnel construction information. We integrate real-time construction data from TIMS into our tool for the automatic generation of as-built BIM models. As a result, we enrich the created as-built 3D models, with additional information dimensions from TIMS and enable the availability and update of the evolving BIM model at any point in time.

We consider our goals successful, first, if we can demonstrate that our prototype can automatically create and enrich as-built BIM models, as currently as-built BIM models are created manually in conventional tunnel construction. Secondly, we evaluate the utility of the prototype with concerning the stakeholder's feedback acquired in a series of interview rounds.

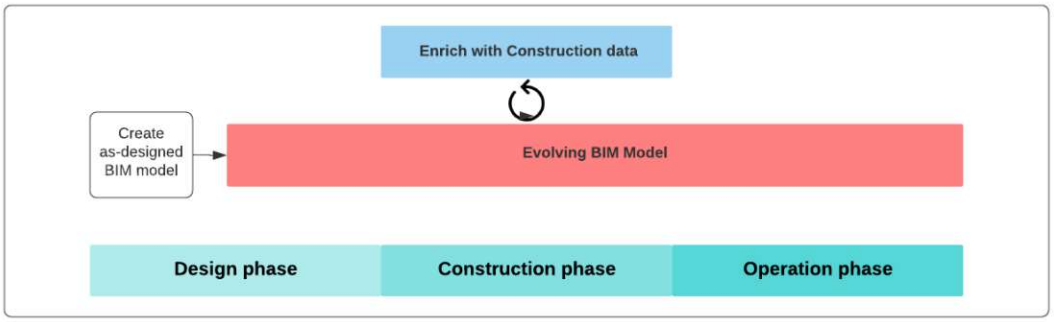

Figure 1.2: The proposed solution to overcome the interrupted BIM life cycle in conventional tunnel construction

[1]New Austrian Tunnelling Method

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Approach

2.1 Methodology

The methodological approach of this master thesis is based on Design Science [7] through the application of the relevance, rigor, and design cycle [6].

The relevance cycle - The contextual environment of this master thesis is digitalization in the tunnel construction domain and BIM, as the central aspect of this ongoing digitalization process. As part of the relevance cycle, we present the problem, the BIM model development, and the usage barrier between as-designed and as-built BIM models in conventional tunnel construction. More precisely, the problem is that the models are created manually, which is an error-prone and time-consuming activity and leads to many undesirable consequences. We hypothesize that, if the models were created automatically, this would not only speed up the model creation process, and improve the model information quality, but also expand the utility of as-built BIM models in the overall construction lifecycle.

Thus, our contribution to the tunnel construction domain is a prototype that enables automatic as-built model generation, relative to the current absence of an artefact of this type. To evaluate the utility of our artefact, we conduct interviews with experts from the tunnel construction domain.

The rigor cycle - To explore the application domain and define the relevant state-of-the-art expertise and existing artefacts we conduct a Systematic Literature Review [10] by answering the following research questions:

1. What is the current state of using BIM in the conventional tunnel construction life cycle?
2. What are the differences between as-designed, as-built, and as-used BIM Models?

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

3. How are BIM models constructed today and to what extent can this process be automatized?

The design cycle - Based on the requirements from the relevance cycle and the existing knowledge base collected from the rigor cycle, we develop a digital prototype. To develop this digital tool we employ an agile software development methodology [13], which proposes a rapid, iterative process, where we can reevaluate our prototype and readjust the product requirements in increments. Applying the agile development methodology not only enables rapid prototype production but also the constant revision process. The revision process is especially important in settings where there is an initial lack of knowledge in the field and many unknown parameters. We choose to maintain short development cycles where we define tasks for the prototype construction, and stepwise revise them, as we acquire more experience and knowledge of the presented problem.

We develop a product artefact that people can use to generate automatic BIM models. More specifically, our artefact is a digital prototype tool that enriches as-designed BIM models with data from the construction phase, and as a result, generates as-built BIM models. The artefact is socio-technical, as it still requires human interaction, where the user needs to specify the as-designed BIM model as input to generate the as-built model. This interaction is reduced to the minimum as the artefact automatically detects all characteristics of the as-designed model and automatically enriches the model with real-time data from the construction site. We chose to construct a prototype instantiation as it provides strong evidence when used to show that a design works as intended and is useful for its intended purpose [16].

To demonstrate our prototype efficacy and effectiveness we qualitatively answer several evaluation points, as defined by Prat et al. We compare our prototype instantiation against manual instantiation (absence of artefact) for the creation of as-built BIM models on the following KPIs (Key Performance Indicators), where activity presents the task of creating as-built BIM models, and model refers to the created as-built BIM model:

- Activity prerequisites - What are the required inputs for the model creation?
- Activity performance - How much time does model creation take?
- Activity simplicity - How much user interaction does the activity need?
- Model completeness - How complete is the model information?
- Model level of detail - How much information does the model contain?
- Model accuracy - How accurate is the model information?
- Robustness - To what extent can the instantiation respond to fluctuations of the environment?

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Since automatic as-built BIM models do not exist in the domain, we want to evaluate the usability potential of these models. To evaluate the overall utility of our prototype in the tunnel construction domain, we conduct a series of expert interviews. We select a minimum of 5 examiners with experience in BIM modeling, as well as stakeholders with experience in the complete tunnel construction life cycle. The interviews will be led by the following set of qualitative questions:

- Would you use the prototype?
- Why/Why not would you use the prototype?
- What would you use the prototype for?
- How would you improve/extend the prototype?
- Additional feedback

To summarize, the prototype efficacy and effectiveness, as well as the prototype utility and usability potential represent the two key factors for the ultimate evaluation of our prototype.

2.2 Structure of the Work

We divided the structure of the work in the following chapters:

- **The Context of the Work** chapter covers the background knowledge necessary for the implementation of the prototype. Here we describe aspects of the tunnel construction domain, specifically, the tunnel construction lifecycle, BIM models, and BIM tools used for tunnel construction. We also present the current state-of-the-art in regards to our problem formation.
- **The Tunnel Information Management System (TIMS)** chapter describes our work on the tunnel information management system which is the basis for acquiring, managing, and distributing real-time construction data to the BIM models.
- **The Prototype for Generating As-built BIM Models** chapter covers the implementation process of our solution to the stated problem. We present the software tools used, the initial requirement analysis, and the detailed description of each step in the process that lead to the complete solution.
- **The Evaluation Results** chapter presents our evaluation process with the results and feedback acquired.
- **The Conclusion** chapter summarises our work results and indicates future development possibilities.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Context of the Work

3.1 Tunnel Construction Lifecycle and Information

A tunnel construction project can be divided into three major phases, the design phase, the construction phase, and the operation phase. According to Succar[21] each of the phases can be subdivided into sub-phases based on this phase's activities and tasks.

The design phase is the starting phase, where the project is carefully planned and specified. It includes conceptualisation and cost planning activities, followed by, architectural, structural, and systems design and 3D modeling activity where the BIM models are designed. The design phase also includes analysis, detailing, and coordination activities. One of the outputs of the tunnel construction design phase is the as-designed BIM model. This model represents how the tunnel should look when tunnel is constructed. The information gathered in the design phase includes more categories [23]:

- Public information about the project - national and local policies, laws and regulations, specifications and procedures, environmental policy, government services, and limitations.
- Location information - geology, hydrology, and topology, surrounding building information, access points for water, electricity, and gas.
- Design information - hydrological investigation data, design specification and schedule, design drawings, preliminary and technical design.
- Other information includes economic information, such as the construction budget, contract information, and information related to similar projects

The construction phase starts with construction planning and construction detailing. It is followed by the actual tunnel construction activity, which also includes manufacturing

and procurement tasks. Lastly, the tunnel construction is commissioned, an as-built BIM model is constructed and the project handover is conducted. Besides the information from the design phase, the following project-specific information is included in the construction phase [23]:

- General situational information - engineering situation, bidding documents, and contracts.
- Construction management information - construction plan and site layout, meeting summaries, construction schedule, actual costs.
- Resource information - allocation schedule for employment, supply schedule for material and equipment, material usage.
- Other information in regards to the environment, construction standards, and technology

The operational phase starts at the end of the construction phase. This phase includes operational activities such as asset management and facility maintenance, as well as other long-term activities, such as decommissioning and major re-programming [21]. Additional information provided in the operational phase mainly stems from the facility management, including user, maintenance, and operational equipment information.

Figure 3.1 presents the amount of information acquired and the incremental data usage in different tunnel construction phases on the y-axis and the x-axis, respectively.

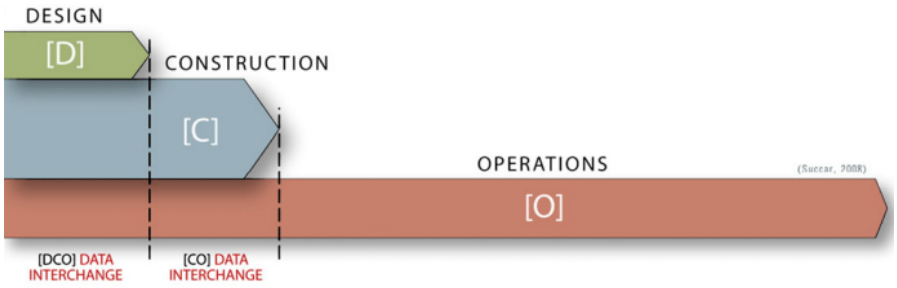

Figure 3.1: Data usage in the tunnel construction process[21]

3.2 BIM Models in Tunnel Construction

Building Information Modelling (BIM) is an established technique for the digital representation of actual information, through tools such as 3D geographic figures and non-geographic information which include elements such as materials, weight, price, procedures, scale, and size [21]. As we previously mentioned, different BIM models are constructed at different stages of tunnel construction projects. The usage potential of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

BIM models is manifold. Figure 3.2 abstractly presents the BIM definition and BIM usage possibilities [21].

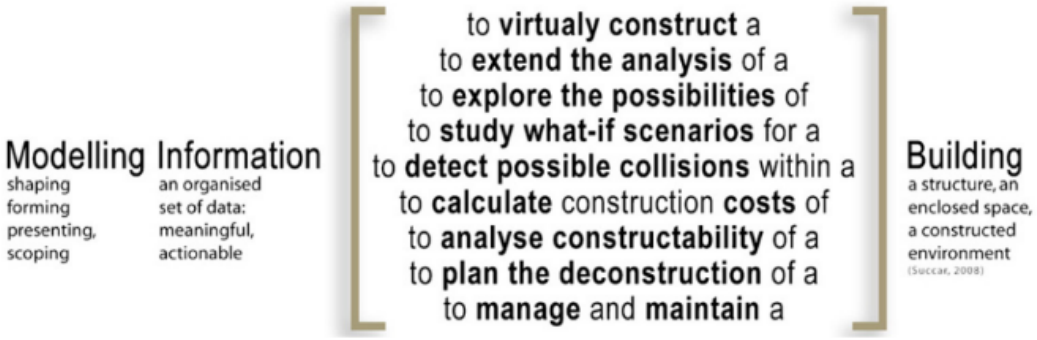

Figure 3.2: Illustrated BIM definition and usage scenarios

For instance, **as-designed BIM models** can be used for collision detection, by combining spatial and geometrical data into a single subsystem. Appropriate detection indicators and actual data entered can form a collision detection system. Thus, conflicts between system components can be intuitively presented [23]. Another use case is building performance analysis at the design stage, an effective technique to reduce unforeseeable costs in the construction and usage phases of the process. **As-built BIM models** enable the overview of the day-to-day progress of a given project. Then, the project stakeholders are able to oversee the planned construction process, monitor the real-time situation on the site and construction safety [23]. And lastly, the usage potential of **as-used BIM models** is mainly connected to the presentation of information relevant to facility and emergency management.

BIM does not only visually present construction projects, it is also an information management tool [21]. The integrated information of the project can also provide construction experience for further use, which includes the ability to do forecasting activities for future tunnel construction projects. According to Succar[21] *"An nD model is an extension of the building information model by incorporating all the design information required at each stage of the lifecycle of a building facility"*. Thus, on top of the basic 3D model, we can also add other information dimensions by enriching the model with additional data. For instance, one significant dimension is time, by adding time-related information to the model we can track the model changes through the tunnel construction lifespan.

3.3 BIM software

BIM software is a 3D design and modeling software that is used in different domains, such as architecture, construction, and engineering. It is an essential component in the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

BIM lifecycle. The United Kingdom's National BIM Report[1] gathers data from nearly 1,000 construction stakeholders and lists the most used BIM software products on the market. We list the most popular BIM software products below:

- **Revit** is a well-known 4D BIM construction software developed by Autodesk[2]. It aims to solve different architectural and design problems, by offering an intelligent approach to different stages of a construction process via BIM models. According to the aforementioned report, it is the most widely used BIM software, used by 46% of respondents. It runs on Microsoft Windows and is a paid software solution. However, a free licence is available for students.
- **AutoCAD** is another Autodesk solution with broad drawing capabilities but it also has support for BIM models. While Revit is based on 3D parametric modeling, AutoCAD creates 2D geometries with 3D modeling capabilities. According to the report it is used by 24% of respondents. It runs on Microsoft Windows and Mac OS X environments. It is a paid software solution, with a free licence available for students.
- **ArchiCAD**[3] is in the third place of the National BIM Report, used by 15% of respondents. According to the authors, it offers a powerful set of built-in tools and an easy-to-use interface that makes it the most efficient and intuitive BIM software on the market. While Revit has more features for building engineering ArchiCAD offers more possibilities for architectural design. It runs on Microsoft Windows and Mac OS X environments. It is a paid software solution, with a free licence available for students.

The full list of BIM software solutions on the market is vast; it includes software tools such as Navisworks[4], Tekla BIMsight[5], Kreo[6], Revizto[7], ArCADia BIM[8], and midas Gen[9]. In conclusion, it can be seen that there exist a lot of software products on the market that offer the ability to work with BIM models.

3.4 State-of-the-Art

Based on our Design Science methodology, as part of the rigor cycle, we conducted a systematic literature review to answer the research questions stated in the Methodology

[1] https://www.thenbs.com/knowledge/national-bim-report-2020
[2] https://www.autodesk.com/
[3] https://graphisoft.com/solutions/archicad
[4] https://www.autodesk.com/products/navisworks/overview
[5] https://www.tekla.com/products/tekla-bimsight
[6] https://www.kreo.net/
[7] https://revizto.com/en/
[8] https://arcadiabimsystem.com/
[9] https://www.midasstructure.com/en/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

section of the master thesis. We defined different sets of keywords *"BIM AND (model OR modeling OR modelling) AND (tunnel OR tunel)"*, *"conventional construction AND (tunnel OR tunel)"*, *"BIM AND (as-built OR built OR as-designed OR designed OR as-used OR used OR automatic)|* and used Google Scholar[10] and Scopus[11] databases to search for relevant literature. Additionally, we used the snowballing technique[20], to further explore the initial systematic literature results. The results of the state-of-the-art analysis are presented below.

Recent literature shows widespread adoption of BIM methods in tunnel construction, as well as benefits of using BIM in tunnel construction projects [26]. Moreover, the usage of BIM methods is supported across all lifecycle phases in construction projects [23], and also more specifically in tunnel construction projects [26]. BIM is mostly used in the design and construction phases of the tunnel construction lifecycle, by constructing as-designed and as-built BIM models [26].

In addition, most of the literature differentiates the general tunnel excavation method of tunnel projects which are constructed with tunnel boring machines (TBM), and tunnels constructed through drill-and-blast cyclic advances, such as with NATM[12][3]. Accordingly, we address these two different excavation methods as TBM tunnel construction and conventional tunnel construction. Sharafat et al.[19] present an overview of existing tunnel BIM literature. It can be seen that there has been a lot of research on employing BIM technology with the TBM tunnel construction method. Koch et al. present a BIM framework for tunnels which consists of BIM models for TBM, tunnel lining, build environment, and ground. Later, based on the aforementioned framework, Ninić et al.[14] propose the semi-automatic generation of as-designed BIM models and as-built BIM models [15] for the TBM tunnel construction method. On the other hand, Lee et al.[12] and recently Sharafat et al.[19] propose theoretical BIM frameworks for the conventional tunneling approach which do not cover automatic BIM model generation.

To conclude, there exist several articles directed toward systematizing and improving the as-designed BIM models in both TBM and conventional tunneling methods. We especially see advances in the literature related to TBM tunneling methods, where Ninić et al.[15] recently proposed automatic as-built BIM model generation. Nevertheless, there is also scientific work that proposes a prototype for the automatic creation of as-used BIM models [24]. Yet, none of the literature covers the topic of automatic creation of as-built BIM models for tunnel projects constructed with the conventional tunneling method.

[10] https://scholar.google.com/
[11] https://www.scopus.com/home.uri
[12] The New Austrian Tunneling Method

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

The Tunnel Information Management System

The Tunnel Information Management System (TIMS) is a prototypical software tool that we developed for replacing the still common paper-based documentation process of tunneling projects employing the New Austrian Tunneling Method (NATM)[9]. While Building Information Modeling (BIM) has been proposed as a means to improve the current situation [19]. Most proposals, however, are at a conceptual level or are inappropriate for directly capturing data at the tunnel face and subsequently utilizing it for the invoicing process [22]. We used and extended the data model from the master thesis of Zach [25], defining the data structures to capture the most essential data directly at the tunnel face and to realize a seamless digital data flow to the company's ERP (Enterprise Resource Planning) system. Based on this, we present the software architecture and the implementation of TIMS.

4.1 Software architecture and Implementation

We implemented TIMS by utilizing a two-tier, multi-layer architecture which is depicted in Figure 4.1. The software architecture has a physical separation between two components of the system [5] - in this case, the front-end and the back-end. In our architecture, the terms front-end and back-end can be used interchangeably with the terms client and server, respectively. Client and server communicate with each other over an application programming interface (API). Via an API the back-end exposes a well-defined set of functionality to the front-end. The API provides two different mechanisms of client-server communication: REST[1], and JSON-RPC[2]. The term "multilayer architecture" refers

[1] Representational State Transfer
[2] JavaScript Object Notation Remote Procedure Call

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

to the fact that there are separate components for different responsibilities, which are typical components for the presentation of data, the business logic, and the data storage [5]. Our implementation is based on Tryton[3], which is an open-source enterprise resource planning (ERP) system and framework with an emphasis on modularity. Tryton provides the core functionality found in modern business software like access control, a workflow engine, a view engine, a report engine, and functionality for internationalization (i18n). Object-relational mapping (ORM) is the process of converting class instances to instances in a relational database, and vice versa. The core of our implementation is the business logic which consists of the tunneling models based on the underlying data model, data constraints, access rules, and definitions of workflows [8].

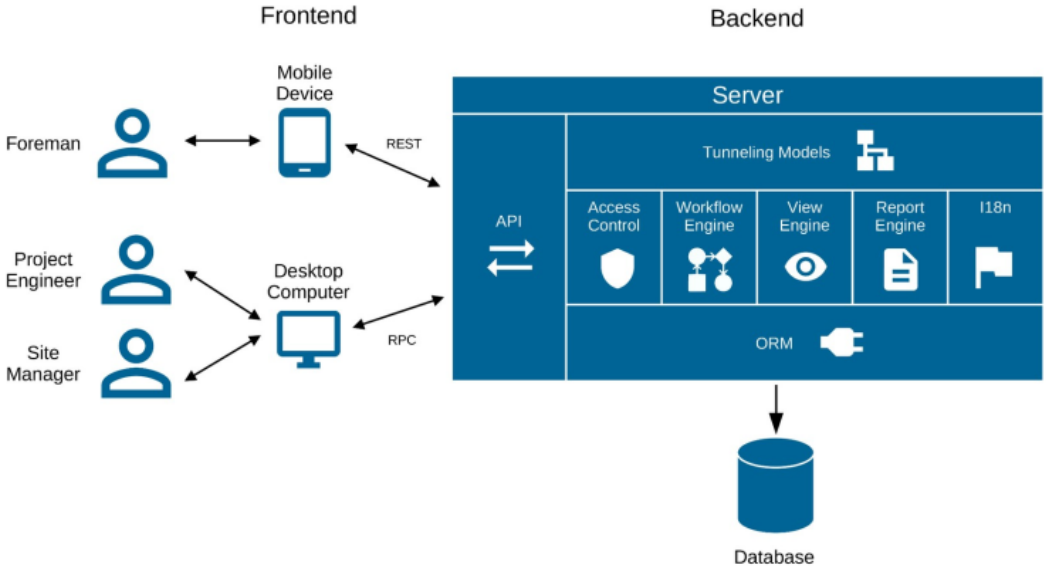

Figure 4.1: TIMS software architecture

We developed two different front-ends that enable digital documentation of the tunneling process. The interface for desktop computers presents a variety of functionalities. First, it enables the user to have complete control over the data model based on CRUD functionality (Create, Read, Update and Delete operations) through master views. The desktop interface enables the initial setup for any tunneling project, where the user can predefine and populate data of *Sections, Cross-Sections, Activity Types*, etc., as well as create, modify and delete *Personnel, Shifts, Rounds*, etc. Besides the data control functionality, the desktop interface provides an overview of the whole tunneling process and tunneling diagrams with the ability to add, modify, and delete activities and material usage in tunnel rounds seen in Figure 4.2. Lastly, based on the acquired tunneling process data, the desktop interface enables the automatic creation of digital documents, such as

[3]https://docs.tryton.org

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

excavation and support sheets, daily construction reports, etc., as well as computation of tunneling and support statistics [8].

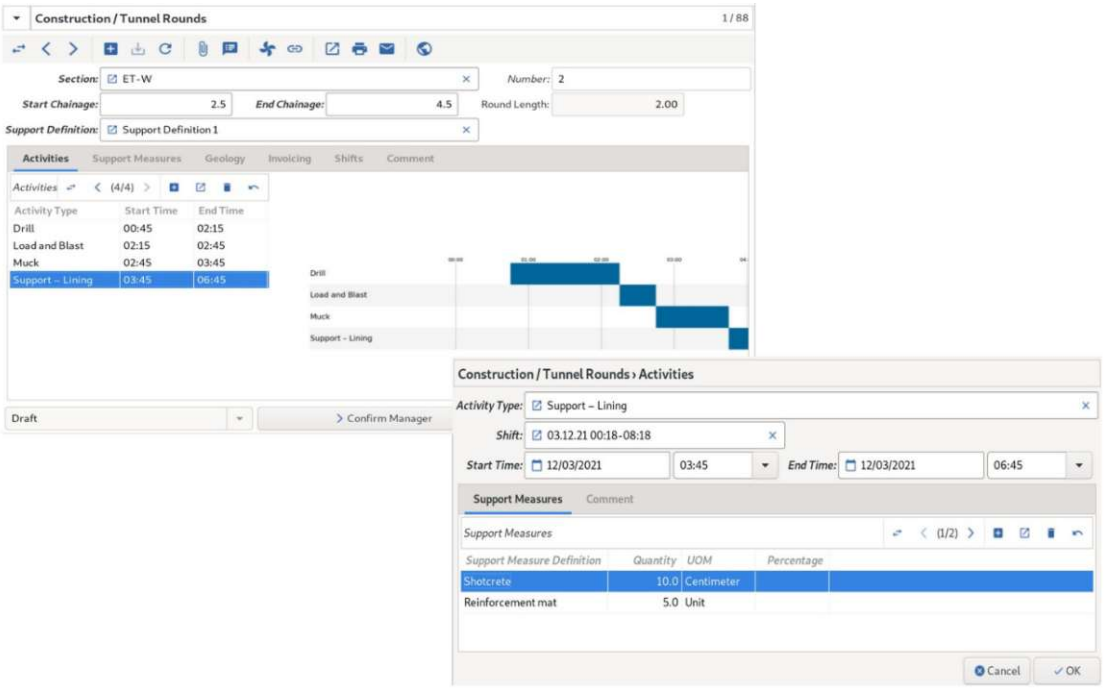

Figure 4.2: TIMS desktop interface

4.1.1 TIMS foreman interface

While the desktop interface acts as a basis for setting up and administering a tunneling project, the second interface for the foreman provides a fast and intuitive approach for data acquisition. It is a web application that can capture the data directly at the tunnel face with the focus on activities and material usage in a tunnel round, intended to be used by the foreman on a mobile device. The web application was created with the Angular framework[4]. The application exchanges data with the Backend server through REST API calls. The design was optimized for tablet devices and simple usage by the foreman at the construction site. Intuitively, the foreman selects the current tunnel section, and the shift and adds the shift personnel. In the same view (s)he is presented with a list of tunnel rounds, where (s)he can select the current tunnel round or create a new one, as shown in Figure 4.3. After selection, (s)he is offered a timetable diagram where (s)he can add, modify or delete activities, as depicted in Figure 4.4. By clicking on the "Add an activity" button, a new view opens. The foreman selects the activity type from a predefined list, while the start time and duration of the activity are automatically prefilled based on past activities, but can be modified. Moreover, the foreman can add

[4]https://angular.io/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the materials used in this activity, where (s)he selects the material type from a predefined list and enters the quantity, and the unit of measure, prefilled based on the material type depicted in Figure 4.5.

Figure 4.3: TIMS foreman interface - Shift report

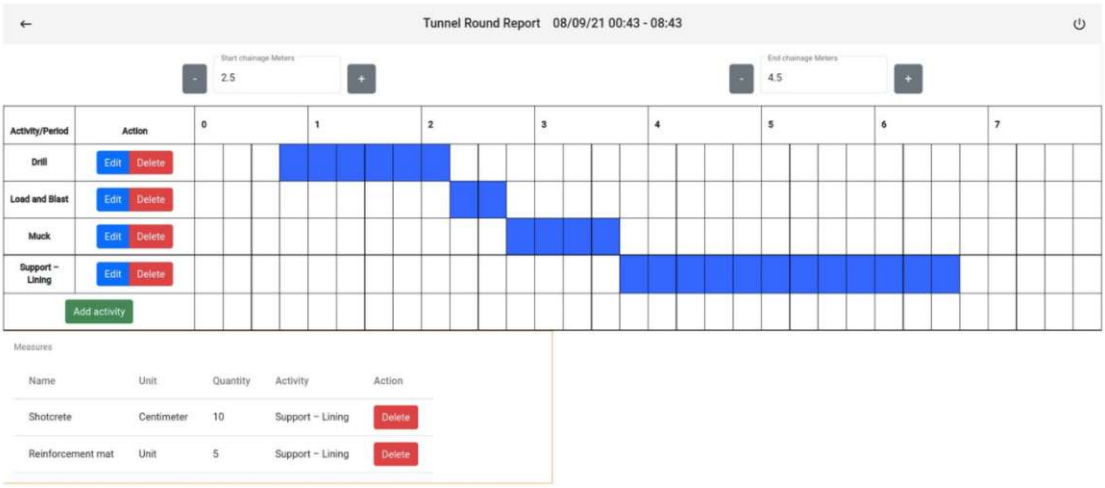

Figure 4.4: TIMS foreman interface - Tunnel round report

4.2 TIMS API

The Tunnel Information Management system helps to realize a central vision of digitalization: to capture each information only once and provide information flows to use the information in all phases of a project. Accordingly, TIMS allows for capturing all relevant information of the tunneling process in a structured data format [8]. The future work on TIMS concentrates on seamless data flows to and from other software tools used in a tunneling project [8]. Hence, the data exchange between TIMS and the prototype for generating the as-built 3D BIM models is one extension of this kind. One of the ways

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

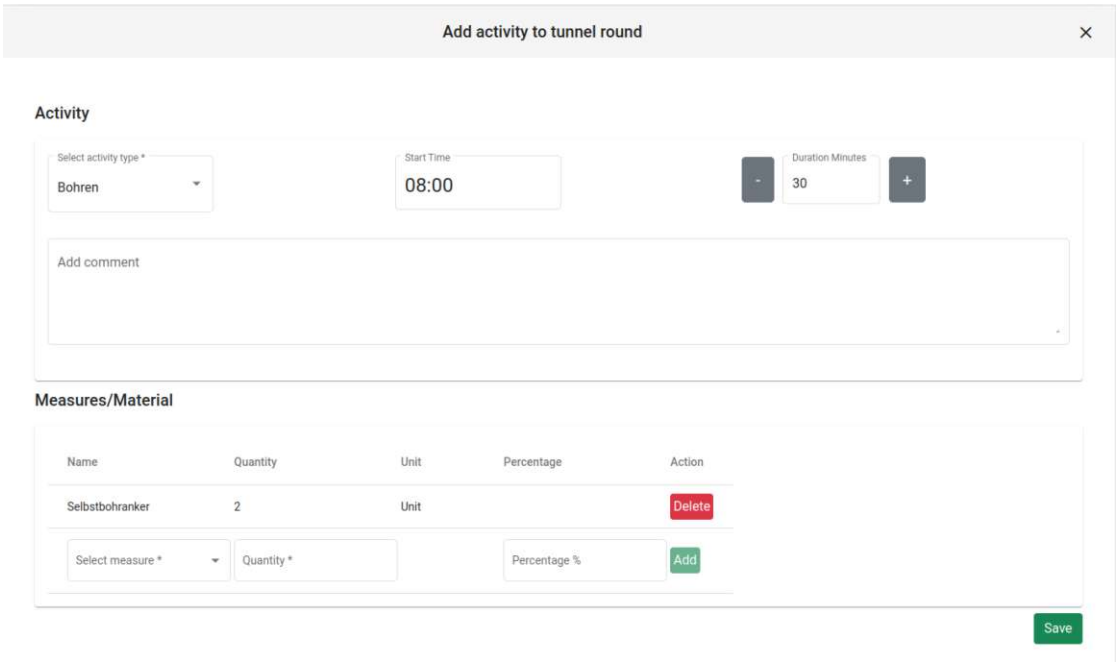

Figure 4.5: TIMS foreman interface - Add activity view

to expose the construction data from TIMS is through the TIMS REST API. Thus, in this master thesis, we use TIMS REST API as the source of real-time construction data. In this section, we present the API interface implementation details.

The API follows the REST architectural style [4]. It is written in Flask[5], a microframework for writing lightweight web applications. Because our underlying back-end structure is based on the Tryton framework we use flask-tryton package[6] to add Tryton support to the REST API.

We create a `BaseView` class that extends the Tryton Views with GET, POST, PUT and DELETE methods according to the REST architectural style. The property `model` refers to the underlying Tryton model definition name and the property `fields` refers to the list of exposed Trtyon model properties.

```
class BaseView(MethodView, metaclass=ViewMeta):

    model = None
    fields = []

    ...

    def get(self, id=None):  # @ReservedAssignment
```

[5]https://flask.palletsprojects.com/en/2.0.x/
[6]https://pypi.org/project/flask-tryton/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
            Model = tryton.pool.get(self.model)

            if id:
                # return a single object
                data = Model.read([id], fields_names=self.fields)[0]
            else:
                # return a list of objects
                query = request.args.get('q')
                if query:
                    query = json.loads(query)
                else:
                    query = []
                ids = [r.id for r in Model.search(query)]
                data = Model.read(ids, fields_names=self.fields)
            return self._make_response(data)

        def post(self):
            Model = tryton.pool.get(self.model)
            data = json.loads(request.data, object_hook=JSONDecoder())
            ret = Model.create([data])[0]
            return self._make_response(ret.id)

        def put(self, id):  # @ReservedAssignment
            Model = tryton.pool.get(self.model)
            obj = Model(id)
            data = json.loads(request.data, object_hook=JSONDecoder())
            if not all(name in self.fields for name in data.keys()):
                raise ApiException('Field does not exist.', 400)
            Model.write([obj], data)
            return self._make_response()

        def delete(self, id):  # @ReservedAssignment
            Model = tryton.pool.get(self.model)
            Model.delete([id])
            return self._make_response()

        ...
```

Next, we expose a selected number of previously defined Tryton construction models by creating their corresponding Views which extend the `BaseView`. Thereby we enable the GET, PUT, POST, DELETE methods on the following construction models in a generic manner.

```
class ActivityView(BaseView):
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
    model = 'construction.activity'
    fields = ['id', 'type', 'type.name', 'start_time', 'end_time', 'comment',
        'round', 'shift', 'measures']


class RoundView(BaseView):
    model = 'construction.tunnel.round'
    fields = ['id', 'section', 'start_chainage', 'end_chainage', 'comment',
        'state', 'support_definition', 'shifts']


class SectionView(BaseView):
    model = 'construction.section'
    fields = ['name', 'state', 'project']


class ShiftView(BaseView):
    model = 'construction.shift'
    fields = ['id', 'section', 'start_time', 'end_time', 'participation',
        'comment', 'activities.start_time', 'activities.end_time']


class ShiftParticipationView(BaseView):
    model = 'construction.shift.participation'
    fields = ['employee.party.name', 'employee', 'role', 'shift']


class MeasureView(BaseView):
    model = 'construction.tunnel.measure'
    fields = ['measure_definition', 'measure_definition.name',
        'measure_definition.type', 'measure_definition.type.class_uom',
        'measure_definition.type.class_uom.name', 'quantity', 'activity',
        'activity.type.name', 'uom', 'uom.name', 'percentage']


class SupportDefinitionView(BaseView):
    model = 'construction.tunnel.support.definition'
    fields = ['name', 'section', 'partial_drift.name', 'cross_section.name']
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

4.3 TIMS Data Model

The TIMS API endpoints and the TIMS Architecture are based on an underlying data model presented in Figure 4.6. The data model is presented as a Unified Modeling Language (UML) class diagram. The UML class diagram covers all the classes necessary to store the data from a NATM construction process.

First, we describe the underlying classes necessary to set up a tunneling project. The *Project* class represents a tunneling project. Each *Project* consists of one or more *Sections* and each *Section* is part of exactly one *Project*. Each *Section* has one or more *SupportDefinitions*. Additionally, each *SupportDefinition* represents one *CrossSection* type. For each *CrossSection* there are multiple *PartialDrifts*, each referring to one *CrossSection*.

Rounds are an essential concept when the tunneling process starts. Each *Round* is related to one *SupportDefinition*, and each *SupportDefinition* is used by one or many *Rounds*. A *Round* consists of one or more tunneling *Activities* and each *Activity* is part of one *Round*. Additionally, each *Activity* can have *Equipment* and *SupportMeasures*. With classes *MaterialUsage* and *EquipmentUsage* we track the *SupportMeasure* and *Equipment* usages in an *Activity*.Multiple *SupportMeasures* can be used in a *SupportDefinition*. With class *MaterialUsageDefault* we define the quantity of a *SUpportMeasure* in a *SupportDefinition*. Each *SupportMeasure* is of a certain *SupportMeasureType* and can be related to more *Products*.

Each *Round* is done in one or many *Shifts*, and each *Shift* is related to one or many *Rounds*. The regular *Activities* are part of a *Round*, but there might be some special *Activities* that are assigned only to a particular *Shift*. Each *Shift* refers to exactly one *Section* and, in addition, it relates to multiple *Personnel*. A *Shift* also has a group leader, which is presented by an additional relationship to *Personnel* class. We differentiate between a *ForemanShift* and regular *Shift* as they can have different working hours.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

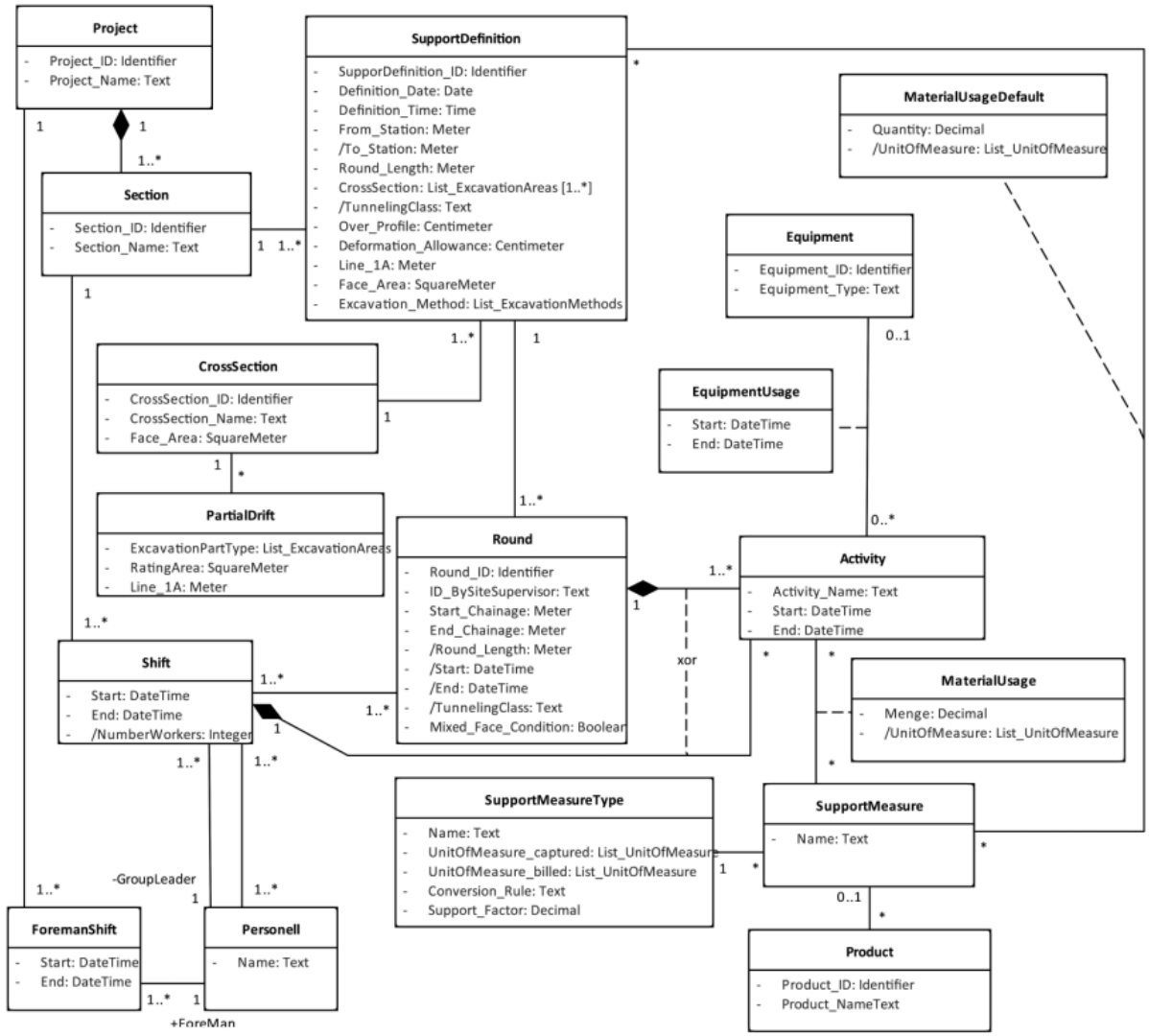

Figure 4.6: Class diagram of TIMS

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 5

Prototype for generating as-built 3D BIM models

5.1 Software tools

To construct the prototype we made several software tool choices. We chose Revit as the BIM software for multiple reasons. First, Revit is the most popular choice on the market, which means that many engineers are familiar with it. Nevertheless, being the most popular choice, many supporting software tools were developed that can be integrated with Revit.

To create automatic 3D BIM models, we need to be able to programmatically manage the BIM software. For this means, Revit exposes a powerful .NET API[1] that can be used to automate repetitive tasks but may also be used to extend the core functionalities of Revit. There are multiple versions of the API 2017.1, 2018, 2018.1, 2018.2, 2019, 2020, 2020.1, 2021.1, and 2022 which follow the current development of Revit. Revit changes over times and there are some breaking changes, that do not make all functions and features backward compatible.

The Revit API enables the user to write plug-ins for Revit, mainly in C#. However, to reload and debug the plug-in, Revit needs to be restarted and the entire BIM model needs to be reloaded each time a change is made to the source code, which we want to avoid, especially when developing a prototype. To overcome this bottleneck for rapid development, there exist a couple of options. As C# is a compiled programming language, code still needs to be rebuilt on every code change, which slows down the development process. In our case, this is especially evident, as we are developing a prototype artifact, where the focus is on development speed and MVP showcase, rather than on performance optimization of the final product.

[1] https://www.revitapidocs.com/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Based on the aim to develop a prototype we chose to use pyRevit, a Rapid Application Development (RAD) environment for Autodesk Revit. As stated by its authors *"pyRevit helps you quickly sketch out your automation and add-on ideas, in whichever language that you are most comfortable with, inside the Revit environment and using its APIs."*[2]. As our final choice, we decided to use Python, an interpreted programming language with the pyRevit (RAD) environment, as Python code is not required to be rebuilt and enables faster development and prototyping.

Additional tools used are PyCharm as an integrated development environment (IDE)[3] for Python (free student licence) and Git for version control.

5.2 Analysis of provided as-designed 3D BIM model

For the scope of the master thesis, we were provided with an as-designed model from Zentrum am Berg[4]. The as-designed 3D BIM model is constructed generically by using a predefined Revit Family. A Revit Family contains the blueprints for Revit elements which describe the basic geometry of the elements, the behaviour of the geometry, and define the element's parameters[5]. Hence, as part of the as-designed 3D BIM model, a Revit Family is provided that essentially describes three different base elements listed below, that correspond to the different excavation types of the tunnel cross-section, as depicted in Figure 5.1.

- EBO_K - the base model for Top Heading (Kalotte) excavation class
- EBO_ST - the base model for Bench (Strosse) excavation class
- EBO_So - the base model for Invert (Sohle) excavation class

All base elements are based on the Adaptive Component Type[6]. Adaptive Components can flexibly adapt to contextual conditions. In the case of the provided base elements, this means that the tunnel round element's length can be adjusted. The length depends on a start point and an endpoint in the geometrical space. Based on the selected start and end points, the base elements get adjusted to fit the contextual conditions.

[2] https://www.notion.so/pyrevitlabs/pyRevit-bd907d6292ed4ce997c46e84b6ef67a0
[3] https://www.jetbrains.com/pycharm/
[4] https://www.zab.at/
[5] https://blogs.autodesk.com/revit/2018/08/27/understanding-revit-families/
[6] https://knowledge.autodesk.com/support/revit/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Revit-Model/files/GUID-6E0ECA27-AF40-4B1D-9E0B-1DE5FBBD45F2-htm.html

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

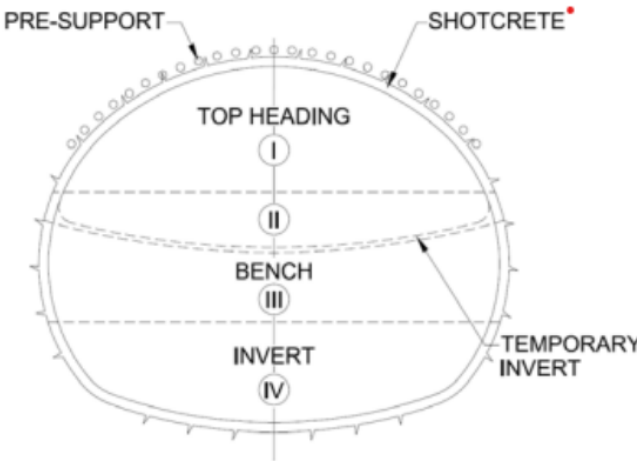

Figure 5.1: Excavation types based on the tunnel cross section

The base elements also contain the following parametric properties, important for element positional and rotational control:

- **rotXY_A, rotXY_B (Rotation)** set the rotation of the base element.
- **Querneigung (Cross slope)** sets to the cross slope of the base element.
- **Gradienthöhe_A, Gradienthöhe_B (Gradients)** set the base element gradients.

Additionally, the provided 3D BIM model contains coded material properties for holding the material values, where the codes are later on used inside another software tool to calculate the costs of the tunnel construction.

5.3 Essential requirements for generating an as-built 3D BIM model in Revit

Having the goal to automatically generate an as-built 3D BIM model, we analyzed the elements of the provided as-designed BIM model, explored the capabilities of the Revit software, and read the Revit API documentation in parallel, using the aforementioned agile methodology. After many revisions, we provide the final list of steps required to generate an as-built 3D BIM model from an as-designed 3D BIM model in Revit:

- To generate the as-built BIM model we need to identify the tunnel axis by accessing the provided tunnel reference curve which represents the tunnel axis.
- To generate the as-built BIM model we need to have access to the provided Revit Family blueprints for base elements (EBO_K, EBO_ST, EBO_So), as these

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

elements can represent different types of tunnel round excavations of desirable length.

- To correctly enrich the models with the material used and other tunnel round properties in the construction phase, we need to create our own "Construction" Family with the necessary properties.
- To approximate the exact positional and rotational parameters of each base element, we need to extract and interpolate the rotational and positional parameters of as-designed base elements located on the tunnel meters corresponding to the as-built base elements.
- To get real-time construction data we need to establish a connection to the TIMS REST API and get the relevant construction data.

5.4 Software Architecture

To conceptually understand our solution for generating automatic as-built 3D BIM models, we create a software architecture overview, depicted in Figure 5.2. Colored in green we define our artifact, **As-built 3D BIM Model Generator**, and the **TIMS API**, the interface from which we get the real-time construction data.

The enabler of the model generation process is the **AB-BIM Plugin**, a plugin that needs to be installed in the Revit environment. The plugin enables the configuration of buttons inside the Revit environment. Every defined button holds logic that performs a certain action in the Revit environment by accessing the Revit API.

In the **AB-BIM Plugin** we define two buttons, that act as interfaces for user-interaction and trigger the underlying services; the **Load Data from TIMS** and **Generate Model** buttons as depicted in Figure 5.2.

The **Load Data from TIMS** button triggers the service for loading construction data from TIMS. First, the service function gets the construction data from TIMS REST API interface. Next, the JSON structure is formatted, so it can be easily used for model generation. Lastly, we timestamp the formatted construction data and store it on the file system. As the output of this service, we obtain a JSON file containing the construction data from the TIMS API at a certain point in time.

The **Generate model** button triggers the process to generate the as-built 3D BIM model. The output of the process is the final as-built 3D BIM model. This process consists of several steps:

- In the step **Load construction data** the user is prompted to choose a file that contains the construction data used to create the as-built BIM model. The data file can be either the one loaded from TIMS or a custom data file created by the user. In this way, we decouple the solution from the TIMS system and enable the user to also use other arbitrary sources, if necessary.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

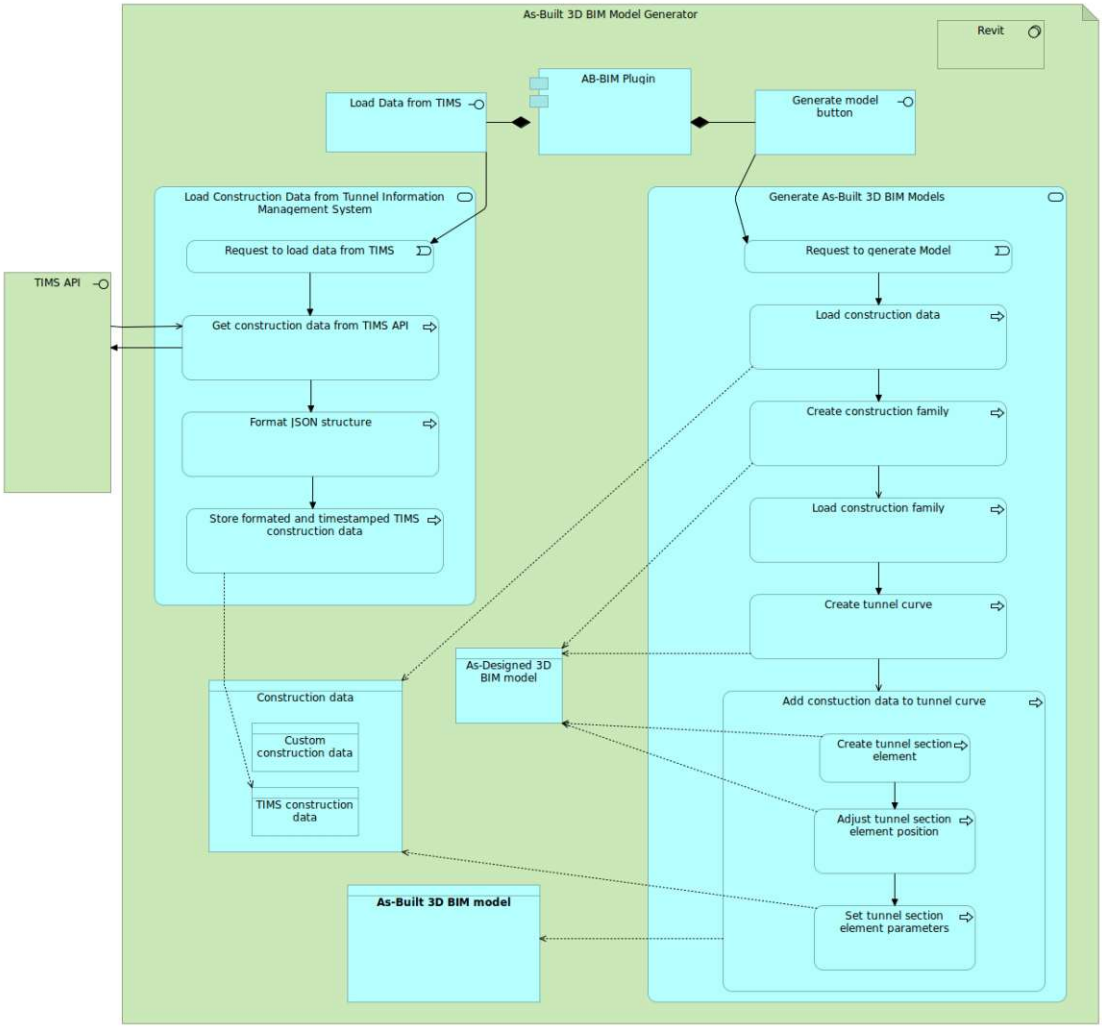

Figure 5.2: Software Architecture of the Prototype for Automatic Generation of As-Built 3D BIM Models

- In the step **Create construction family** we locate the provided as-designed Revit Family with user assistance, by prompting the user to enter the Revit Family's name. After locating the provided Family, we make a copy of the blueprints and add additional properties related to the construction phase of the tunnel construction lifecycle. Lastly, we store the new "Construction" Revit Family blueprints.

- In the step **Load construction family** we load the previously created "Construction" Family in the Revit interface to use its blueprints for the creation of the as-built 3D BIM model elements.

- The function **Create tunnel curve** locates the as-designed 3D model tunnel curve

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

and replicates it as the basis for the as-built 3D model axis. The tunnel curve is a prerequisite for adding the tunnel and construction elements.

- The final step is to **Add the construction data to the tunnel curve**. Based on the loaded construction data, we iterate over each tunnel round and execute the following functions:
  - We **Create a tunnel section element** with the corresponding starting and ending meter, and the corresponding cross-section type (e.g Top Heading, Bench, Invert). This enables the visual creation of the as-built element in the Revit interface.
  - In the step **Adjust tunnel section element position** we refine the element position and rotation by detecting the positional and rotational properties of the corresponding as-designed BIM model elements.
  - Lastly, in step **Set tunnel section element parameters** we enrich the tunnel section element with parameters from the construction data, such as the construction material used in this tunnel round, or the duration and additional comments in this tunnel round.

5.5 Implementation

After we defined the overview of our software architecture, in this section we provide detailed documentation of every underlying function of the **AB-BIM Plugin**, which we annotate based on the elements presented in the software architecture section.

5.5.1 Load Data from TIMS

We write a pyRevit script that connects to the exposed TIMS API to extract the current state at the tunnel construction site. We send the data to our Revit Plugin to automatically create 3D as-built models enriched with construction data. This seamless data exchange enables time-saving due to avoidance of multiple entries, reduction of transmission errors, and higher data quality, consistency, and traceability.

The script execution consists of three high-level functions; first, we authenticate to the TIMS Rest API to obtain a JWT token[7] that authorizes our requests to the necessary TIMS API endpoints. With the `authenticate()` function based on our provided credentials we obtain a JWT access token.

```
def authenticate():
    response = requests.post("https://tunnel.big.tuwien.ac.at:8000/api/login/",
                             json={"user": credentials.username,
                                   "password": credentials.password})
    access_token = "Bearer " + response.text
```

[7] https://jwt.io/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
    headers = {"Authorization": access_token}
    return headers
```

Next, we get the necessary construction data and format it. The JSON format of the construction data can be seen in Figure 5.3. The structure follows a simple form, on the top level we have an array of `sections` with the property `name`. Each tunnel section contains several tunneling `rounds`. For each round we provide information about the `start_meter`, `end_meter`, `start_datetime`, `end_datetime`, and the `duration` of the tunneling round process. Further, we provide information about the `cross_section_type`, optional foreman `comments` and lastly an array of `material` used in the tunnel round. Each material instance contains the material `name`, the material unit in the property `value_type`, and the number of material units in the property `value`.

The flat structure was chosen as it enables a simple decomposition of the construction data iteratively, effectively usable in the process of creating as-built 3D BIM models. As we are building a new, non-existing prototype, we decided to include all available construction information that could be useful for the stakeholders. Nevertheless, this format can be extended based on future stakeholder utility use cases.

The function `get_formatted_data()` loads all the necessary data and formats it to our JSON serialized format.

```
def get_formatted_data():
    sections = get_sections()
    for section in sections:
        rounds = get_rounds(section.id)
        rounds = sorted(rounds, key=lambda x: x.start_meter)
        for round in rounds:
            round_material = get_material(round.id)
            round.material = serialize_data(round_material)
        section.rounds = (serialize_data(rounds))
    return {"sections": serialize_data(sections)}
```

Lastly, we store the JSON formatted data on the file system by calling the `store_data(data)` function. Additionally, we also create the file path and add the current timestamp to annotate the file with the current tunnel construction state.

```
def store_data(data):
    file_path = create_file_path()
    with open(file_path, 'w') as f:
        json.dump(data, f, ensure_ascii=True)
        print("Data was successfully stored!")


def create_file_path():
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
    absolute_path_of_script = os.path.dirname(__file__)
    absolute_file_path = get_parent_dir(get_parent_dir(
        get_parent_dir(absolute_path_of_script)))
    return absolute_file_path + '\\data\\tims' + get_current_timestamp() + '.json'
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
{
    "sections": [
        {
            "id": 1,
            "name": "ET-W",
            "rounds": [
                {
                    "id": 1,
                    "start_meter": 0,
                    "end_meter": 1.2,
                    "cross_section_type": "Kalotte",
                    "comment": "",
                    "start_datetime": "13.3.2022 3:26",
                    "end_datetime": "13.3.2022 9:26",
                    "duration": "6.0h",
                    "material": [
                        {
                            "name": "Spritzbeton Kalotte und Strosse",
                            "value_type": "centimeter",
                            "value": 10.0
                        }
                    ]
                },
                {
                    "id": 2,
                    "start_meter": 1.2,
                    "end_meter": 2.3,
                    "cross_section_type": "Kalotte",
                    "comment": "",
                    "start_datetime": "14.3.2022 7:0",
                    "end_datetime": "14.3.2022 8:0",
                    "duration": "1.0h",
                    "material": [
                        {
                            "name": "Selbstbohranker",
                            "value_type": "Unit",
                            "value": 2.0
                        },
                        {
                            "name": "Ortsbrustanker",
                            "value_type": "Unit",
                            "value": 5.0
                        }
                    ]
                }
            ]
        }
    ]
}
```


Figure 5.3: Construction data JSON formatting

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

5.5.2 Generate Model

We write a pyRevit script that handles the whole model generation process. Figure 5.4 shows the overview of the as-built model generation process. The diagram shows external artifacts (red color), artifacts created during the generation process (grey color), the resulting artifact, the generated as-built Revit model (green color), and the generation process steps (yellow color). The first prerequisite is to load the existing construction data from a file (generated from TIMS API or manually). The next prerequisite is to create an as-built Revit family by locating and extending the provided as-designed Revit family with additional construction parameters. Next, we create the tunnel axis as the basis for the tunnel element's placement. Then we iterate over each item of the provided construction data. Based on the provided as-designed model, item type, and location information, we choose the appropriate Revit element to represent the new tunnel section. Subsequently, we place the element on the tunnel axis based on the item location information and the provided as-designed model. Finally, we enrich the as-build model with data presented in 3.1 which is contained within the currently iterated construction data item. In the next part, we give a detailed description of all functions necessary to complete this process.

Loading construction data

The function `load_construction_data()` prompts the user to choose a file that contains the construction data. As previously mentioned, this data can be from TIMS API or other sources but needs to conform to the JSON format presented in Figure 5.3.

```
def load_construction_data():
    print('Loading construction data')
    Alert("Click button \'Load data from TIMS\' to generate current data "
          "snapshot from TIMS. You are also able to add your own construction "
          "data",
          header="Adding Construction Data",
          title="Information")
    file_path = forms.pick_file(
        title='Please select a file containing construction information',
        file_ext='json')
    data = open(file_path, 'r').read()
    return json.loads(data)
```

Creating and loading the as-built construction family

Previously, we defined the base elements of the as-designed BIM model family, EBO_K, EBO_ST, and EBO_S. We create the as-built construction family based on the provided as-designed family. Hence, the first step is to locate the as-designed family and use it as a basis for constructing our as-built construction family. We implement

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

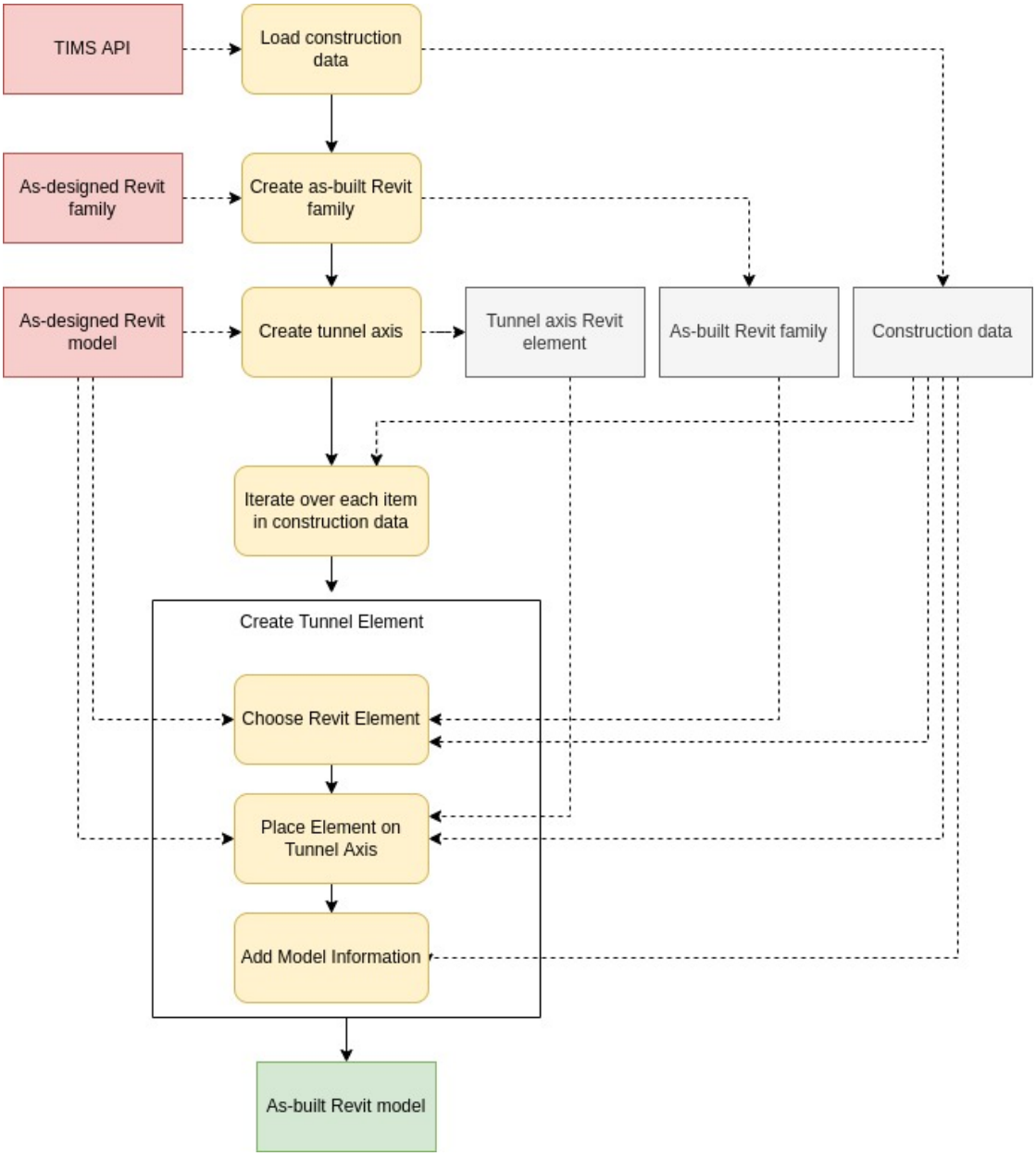

Figure 5.4: Overview of the As-built Model Generation Process

this feature in the function `locate_as_designed_family()`. To make the function more generic, we prompt the user to enter the name of an as-designed family element. The `get_element(revit_document, name)` function selects any FamilySymbol[8] element by a given name. By locating the as-designed element by the entered name, we are also able

[8]https://www.revitapidocs.com/2022/a1acaed0-6a62-4c1d-94f5-4e27ce0923d3.htm

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

to get access to the element's Family[9] with our utility helper functions[10].

```
def locate_as_designed_family():
    as_designed_element_name = TextInput(
        'Loading As-designed Family',
        default='EBO_K',
        description='Please enter the name of a used as-designed model.')
    child_family_element = Utils.get_element(doc, as_designed_element_name)
    return Utils.get_element_family(child_family_element)
```

Next, by having access to the Family element of the child base element we can construct our own as-built construction family, with our base elements for different excavation types; Top Heading (German: "Kalotte"), Bench (German: "Strosse"), and Invert (German: "Sohle"). The function to create our construction family is presented below. We make changes in Revit inside transactions, where we also provide appropriate exception handling logic.

```
def create_construction_family(new_family_name):
    print('Creating construction family')
    existing_family = locate_as_designed_family()
    family_doc = doc.EditFamily(existing_family)
    add_construction_parameters(family_doc)
    options = DB.SaveAsOptions()
    options.OverwriteExistingFile = True
    try:
        family_doc.SaveAs(new_family_name, options)
    except Exception as e:
        print('Overriding Revit file permissions')
        loadFamilyCommandId = UI.RevitCommandId.LookupCommandId('ID_FAMILY_LOAD')
        UI.UIApplication(uiapp).PostCommand(loadFamilyCommandId)
        raise Exception("Couldn't create family due to Revit file permissions,"
                        " please close the dialog and try again.")
```

Additionally, based on our analysis of the provided as-designed BIM model, we need to add additional construction parameters, to our newly created construction base elements in function `add_construction_parameters(family_doc)`. Each Revit element has a set of parameters. These parameters are separated into different parameter groups. We choose to save our construction parameters, inside the default built-in parameter group of identity parameters[11] in the function `add_identity_parameter`. The construction parameters refer to the construction material and additional tunnel round information acquired from the Foreman interface application. In the function `load_construction_parameters()` we

[9] https://www.revitapidocs.com/2022/f51d019d-6ff3-692b-d1d2-b497cab564de.htm
[10] The whole utility class is available in Appendix C
[11] https://www.revitapidocs.com/2022/9942b791-2892-0658-303e-abf99675c5a6.htm

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

provide all possible construction material definitions from TIMS, as well as the place to store the Foreman comment, and time dimension attributes. The construction parameters in the snippet are in German language.

```
def add_construction_parameters(family_doc):
    parameters_tuples = load_construction_parameters()
    for p in parameters_tuples:
        parameter_name = p[0]
        parameter_type = p[1]
        add_identity_parameter(family_doc, parameter_name, parameter_type)


def load_construction_parameters():
    return [
        ('Selbstbohranker', DB.ParameterType.Text),
        ('SN Mörtelanker', DB.ParameterType.Text),
        ('Ortsbrustanker', DB.ParameterType.Text),
        ('Baustahlgitter 1. Lage, ohne Bogen', DB.ParameterType.Text),
        ('Baustahlgitter 1. Lage, mit Bogen', DB.ParameterType.Text),
        ('Baustahlgitter 2. Lage, mit Bogen', DB.ParameterType.Text),
        ('Rammspieß', DB.ParameterType.Text),
        ('Selbstbohrspieß', DB.ParameterType.Text),
        ('Spritzbeton Kalotte und Strosse', DB.ParameterType.Text),
        ('Spritzbeton Ortsbrust', DB.ParameterType.Text),
        ('Spritzbeton Teilflächen', DB.ParameterType.Text),
        ('Bogen', DB.ParameterType.Text),
        ('Verpressung', DB.ParameterType.Text),
        ('Teilflächen', DB.ParameterType.Text),
        ('Sprengstoff', DB.ParameterType.Text),
        ('Kommentar', DB.ParameterType.Text),
        ('Zeit Anfang', DB.ParameterType.Text),
        ('Zeit Ende', DB.ParameterType.Text),
        ('Dauer', DB.ParameterType.Text),
    ]

def add_identity_parameter(family_doc, parameter_name, parameter_type):
    family_manager = family_doc.FamilyManager
    family_doc_transaction = DB.Transaction(family_doc)
    try:
        family_doc_transaction.Start("ADD PARAMETER")
        family_manager.AddParameter(parameter_name,
                                    DB.BuiltInParameterGroup.PG_IDENTITY_DATA,
                                    parameter_type, True)
        family_doc_transaction.Commit()
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
    except Exception as e:
        print(e)
        family_doc_transaction.RollBack()
```

Next, we load the created construction family with function `load_construction_family()`.

```
def load_construction_family(family_name):
    print('Loading construction family')
    try:
        transaction.Start('LOAD CONSTRUCTION FAMILY')
        result = doc.LoadFamily(family_name)
        if not result:
            print('Family already loaded, using loaded family')
        transaction.Commit()
    except Exception as e:
        transaction.RollBack()
        raise Exception("Could not load family", e)
```

Identifying and creating the tunnel axis

To identify the tunneling axis, we need to get the existing as-designed tunnel axis. In the function `get_exiting_tunnel_curve()` we search the open document for a tunnel curve with the function `search_for_tunnel_curve(document)`. If the tunnel curve is not found, we implemented the function `search_families_having_tunnel_curve()` which searches all available Revit project families for a tunnel axis and gives the user appropriate feedback with all families containing a tunnel axis. The tunnel axis element type in Revit corresponds to 'Autodesk.Revit.DB.CurveByPoints'[12].

```
def get_existing_tunnel_curve():
    result = search_for_tunnel_curve(doc)
    if result:
        return result
    else:
        search_families_having_tunnel_curve()


def search_for_tunnel_curve(document):
    elements_collector = DB.FilteredElementCollector(document)\
        .WhereElementIsNotElementType()\
        .ToElements()
    for element in elements_collector:
        if element.GetType() and \
```

[12] https://www.revitapidocs.com/2022/2df7ab39-1ac0-5803-79fa-b23a959bf8f2.htm

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
                str(element.GetType()) in TUNNEL_AXIS_ELEMENT_TYPES:
            return element
    return None


def search_families_having_tunnel_curve():
    available_families = []
    for family in Utils.get_families():
        family = doc.GetElement(family.Id)
        if family.IsEditable:
            fam_doc = doc.EditFamily(family)
            result = search_for_tunnel_curve(fam_doc)
            if result:
                available_families.append(family.Name)
    content = Utils.format_list_to_string(available_families)
    Alert(title='Error',
          header='Could not locate tunnel curve, '
                 'please open one of the family documents'
                 ' with the tunnel curve',
          content=content)
```

Finally, if the tunnel axis is found, we can create a new tunnel axis as the basis for our as-built model by copying the found as-designed tunnel axis, for clarity and easy comparison we create the tunnel axis in the same reference plane. All creation logic in Revit needs to be included in transactions, which are context-like objects that guard any changes made to a Revit model. Additionally, we implement transaction exception handling logic which we finally handle in a user-friendly manner in one place, in the top-level code environment[13].

```
def create_tunnel_curve():
    print('Creating tunnel curve')
    as_designed_tunnel_curve = get_existing_tunnel_curve()
    new_xyz = DB.XYZ(200, -200, 0)
    try:
        transaction.Start('CREATE TUNNEL CURVE')
        new_tunnel_curve_ids = DB.ElementTransformUtils.CopyElement(
            doc,
            as_designed_tunnel_curve.Id,
            new_xyz
        )
        new_tunnel_curve = doc.GetElement(new_tunnel_curve_ids[0])
        transaction.Commit()
```

[13] https://docs.python.org/3/library/__main__.html

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
    except Exception as e:
        transaction.RollBack()
        raise Exception(e)
    return new_tunnel_curve

doc = __revit__.ActiveUIDocument.Document
transaction = DB.Transaction(doc)

try:
    as_built_tunnel_curve = create_tunnel_curve()
except Exception as error:
    Alert(str(error), header="User error occurred", title="Message")
```

Adding construction data to the tunnel curve

Based on the previously loaded construction data, we iterate over the tunnel round objects in function `add_construction_data(construction_data)`. A tunnel round object is based on the specification in Figure 5.3.

```
def add_construction_data(construction_data):
    print('Adding construction data')
    cross_section_type = SelectFromList(
        'Select cross section type of tunnel rounds you want to generate',
        ["Kalotte", "Strosse", "Sohle"])
    for item in construction_data['sections']:
        for round in item['rounds']:
                add_tunnel_element(
                    round['start_meter'],
                    round['end_meter'],
                    round['material'],
                    round['comment'],
                    round['start_datetime'],
                    round['end_datetime'],
                    round['duration']
                )
```

For each tunnel round element we call the function `add_tunnel_element`. We first detect the corresponding as-designed elements based on the start and end meter of the tunnel round in function `find_as_designed_element_name(start_meter,end_meter)`. The fallback is that the user manually enters the corresponding as-designed element.

```
def add_tunnel_element(start_meter, end_meter, material, comment,
                       start_time, end_time, duration):
    print('Adding tunnel element')
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
    as_designed_element_name = find_as_designed_element_name(start_meter,
                                                             end_meter)
    if as_designed_element_name is None:
        as_designed_element_name = TextInput(
            'Could not find element at position ('
            + str(start_meter) +' — ' + str(end_meter) + ')',
            description='Please enter the model type name for this tunnel round',
            default='EBO_K')
    section_element = create_section_block(
        as_designed_element_name, as_built_tunnel_curve, start_meter, end_meter)
    set_element_parameter(section_element, 'Kommentar', comment)
    set_element_parameter(
        section_element, 'Station Anfang', str(start_meter) + 'm')
    set_element_parameter(section_element, 'Station Ende', str(end_meter) + 'm')
    set_element_parameter(section_element, 'Zeit Anfang', start_time)
    set_element_parameter(section_element, 'Zeit Ende', end_time)
    set_element_parameter(section_element, 'Dauer', duration)
    add_section_material(material, section_element)
    set_section_position(start_meter, end_meter, section_element)


def find_as_designed_element_name(start_meter, end_meter):
    collector = db.Collector(of_class='FamilyInstance')
    elements = collector.get_elements()
    for e in elements:
        try:
            if e.Symbol.Family.Name != 'as—built' and has_blocknummer(e):
                element_start_meter, element_end_meter = \
                    find_as_designed_model_position(e)
                if element_overlap(start_meter, end_meter, element_start_meter,
                                   element_end_meter):
                    return e.name
        except Exception as e:
            continue
    return None
```

Next, in the function `create_section_block` we create the tunnel element by setting the element placement points on the tunnel curve, with the functions `get_element_placement_points()` and `create_new_point_on_edge(edge, position_meter)`. To set our placement points with the provided start and end meter, we create new points on the tunnel curve and replace the initial placement points with the newly created ones. Additionally, we also need to convert the meter distances into feet distances, because Revit calculates system

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

units in Imperial units[14].

```
def create_section_block(
        section_element_type_name,
        tunnel_curve,
        beginning_meter,
        ending_meter
    ):
    section_family_element_type = Utils.get_as_built_element(
        doc, section_element_type_name)

    try:
        transaction.Start("CREATE SECTION BLOCK")
        section_family_element_type.Activate()
        new_section_block = DB.AdaptiveComponentInstanceUtils.\
            CreateAdaptiveComponentInstance(
            doc,
            section_family_element_type
        )
        placement_point_a, placement_point_b = get_element_placement_points(
            new_section_block
        )
        placement_point_a.SetPointElementReference(
            create_new_point_on_edge(tunnel_curve, beginning_meter)
        )
        placement_point_b.SetPointElementReference(
            create_new_point_on_edge(tunnel_curve, ending_meter)
        )
        transaction.Commit()
    except Exception as e:
        transaction.RollBack()
        raise Exception("Couldn't create section block", e)

    return new_section_block


def get_element_placement_points(element):
    try:
        placement_points = DB.AdaptiveComponentInstanceUtils.\
            GetInstancePlacementPointElementRefIds(element)
        return doc.GetElement(placement_points[0]), \
               doc.GetElement(placement_points[1])
```

[14]https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/3DSMax/files/010D97F3-A50E-42D2-BD55-6EB0F239EBFE-htm.html

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
    except Exception as e:
        raise Exception("Couldn't get placement points", e)


def create_new_point_on_edge(edge, position_meter):
    return doc.Application.Create.NewPointOnEdge(
        edge.GeometryCurve.Reference,
        DB.PointLocationOnCurve(
            DB.PointOnCurveMeasurementType.SegmentLength,
            Utils.meter_to_feet(position_meter),
            DB.PointOnCurveMeasureFrom.Beginning
        )
    )
```

Next, we set the previously defined construction parameters with the function `set_element_parameter` and we add the construction material with the function `add_section_material(material, section_element)`.

```
def set_element_parameter(element, parameter_name, parameter_value):
    try:
        transaction.Start('SET PARAMETER')
        parameter = get_element_parameter(element, parameter_name)
        parameter.Set(parameter_value)
        transaction.Commit()
    except Exception as e:
        transaction.RollBack()
        raise Exception("Couldn't set section parameter", e)

def add_section_material(material, section_element):
    print('Adding section material')
    for item in material:
        set_element_parameter(
            section_element,
            item['name'],
            str(item['value']) + ' ' + item['value_type']
        )
```

Last but not the least, we adjust the tunnel element position, by interpolating the positional and rotational parameters of corresponding as-designed elements detected based on the start and end meter.

```
def set_section_position(start_meter, end_meter, section_element):
    print('Setting section position')
    position_parameters = approximate_section_position_parameters(
        start_meter, end_meter, section_element.Name)
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
        for parameter in position_parameters:
            parameter_name = parameter[0]
            parameter_value = parameter[1]
            set_element_parameter(section_element, parameter_name, parameter_value)


def approximate_section_position_parameters(
        start_meter, end_meter, element_type):
    overlap_elements = find_as_designed_elements_that_overlap_element(
        start_meter, end_meter, element_type)
    return [
        ('Gradientenhöhe_A', Utils.millimeter_to_feet(
            approximate_parameter(overlap_elements, 'Gradientenhöhe_A'))),
        ('Gradientenhöhe_B', Utils.millimeter_to_feet(
            approximate_parameter(overlap_elements, 'Gradientenhöhe_B'))),
        ('Querneigung', DB.UnitUtils.ConvertToInternalUnits(
            approximate_parameter(overlap_elements, 'Querneigung'),
            get_degree_forge_type())),
        ('rotXY_A', DB.UnitUtils.ConvertToInternalUnits(
            approximate_parameter(overlap_elements, 'rotXY_A'),
            get_degree_forge_type())),
        ('rotXY_B', DB.UnitUtils.ConvertToInternalUnits(
            approximate_parameter(overlap_elements, 'rotXY_B'),
            get_degree_forge_type())),
    ]
```

5.6 Final solution

In this section, we present the final solution for the automatic generation of as-built 3D BIM models. First, we open an as-designed 3D BIM tunnel model in Revit, as presented in Figure 5.5. In the upper left corner, we can see two buttons, which are part of our pyRevit Plugin for Revit, presented in more detail in Figure 5.6.

By clicking the ”Load Data from TIMS” button we successfully generate a JSON file with the current snapshot of the construction data. The construction data is acquired through the TIMS foreman interface, as previously depicted in Figure 4.4. Thus, by clicking the ”Load Data from TIMS” button we load the latest version of construction data, with changes made in the TIMS foreman interface.

Next, by clicking the ”Generate Model” button we trigger the generation process. If we are in the wrong view, where a tunnel curve cannot be located, we will get the alert window as shown in Figure 5.7, with a list of documents that contain a tunnel axis. Otherwise, when we open the correct view, as depicted in Figure 5.8 the generation process starts.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

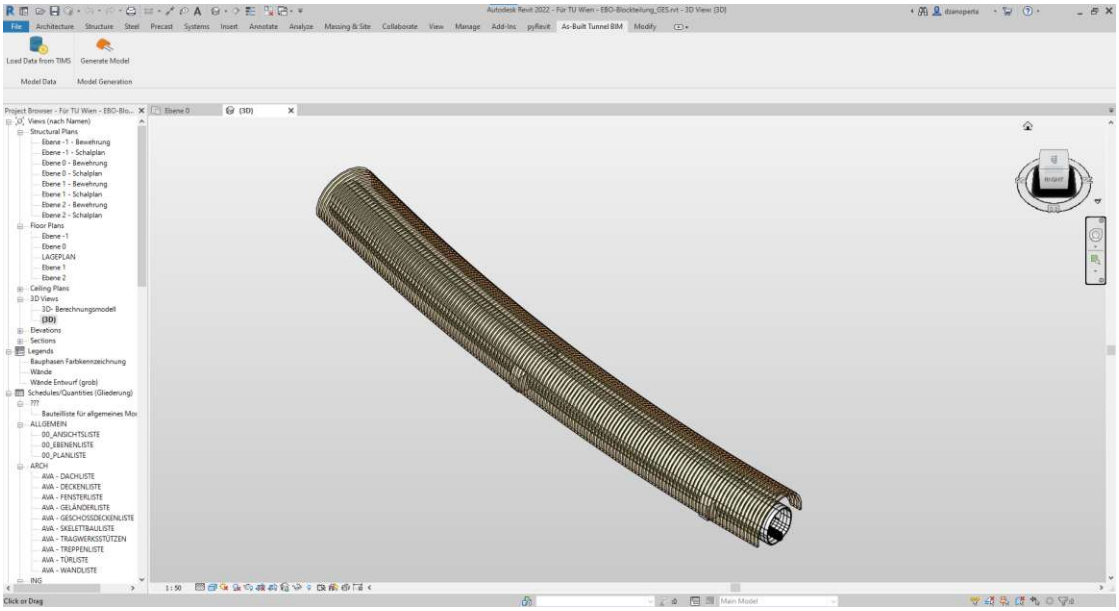

Figure 5.5: As-designed 3D BIM model in the Revit environment

Figure 5.6: PyRevit Plugin Buttons

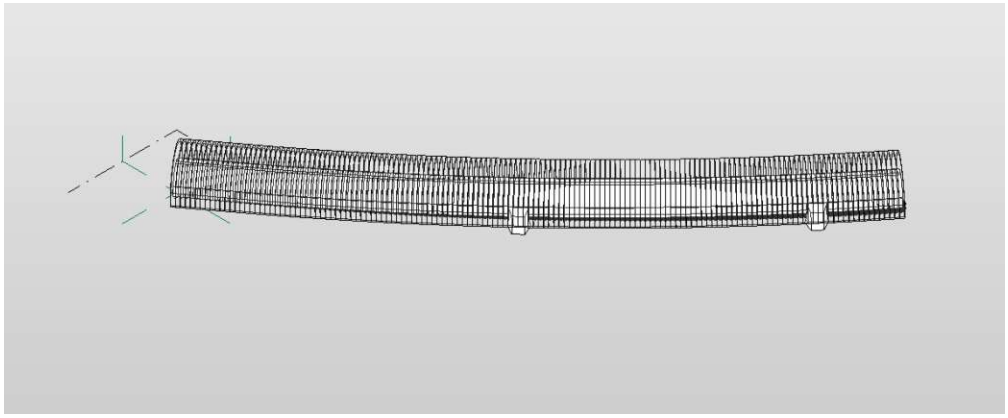

Figure 5.8: As-designed 3D BIM model in the Revit environment with accessible tunnel curve

Next, we provide a name of a used as-designed element to create the construction family.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

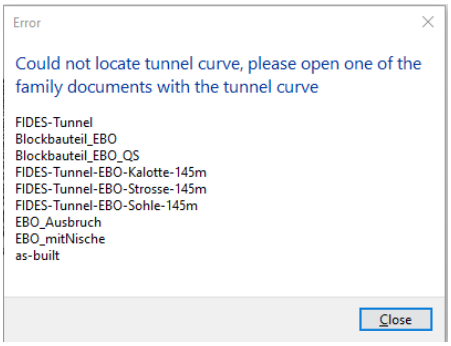

Figure 5.7: Locating the tunnel curve alert

The input window is shown in Figure 5.9. The process continues, the plugin finds the as-designed BIM family and prepares an as-built BIM family for the as-built 3D BIM model.

Figure 5.9: Locating the as-designed BIM family

Then, the user is prompted to choose a file with the construction data that (s)he wants to use for the as-built 3D BIM model as shown in Figure 5.10. Notice that we see data files loaded from TIMS API with appropriate timestamps, representing the time at which the data was loaded.

After we located the as-designed BIM model and loaded the construction data, the plugin starts adding tunnel elements to the new tunnel curve. Depending on the number of tunnel elements this process can take from a couple of seconds to a couple of minutes. After the process finishes, the final solution is generated, as depicted in Figure 5.11. You can see the difference between the as-built tunnel round elements, with their different start and end meters, digitally presenting the actual state of the real tunnel. Additionally, by clicking on a tunnel element, a property window opens, containing all information about the selected tunnel element (tunnel round), such as the used construction material, comments, start and end meter, and the duration of the tunnel round excavation. The properties window is shown in Figure 5.12.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

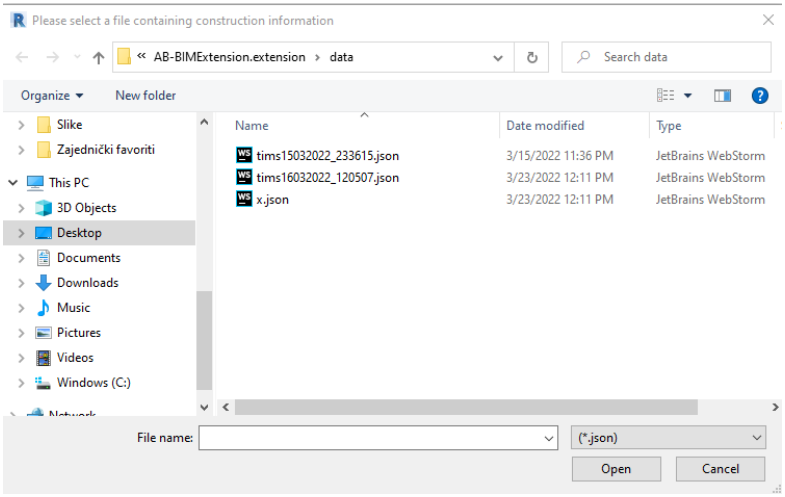

Figure 5.10: Selecting the construction data file

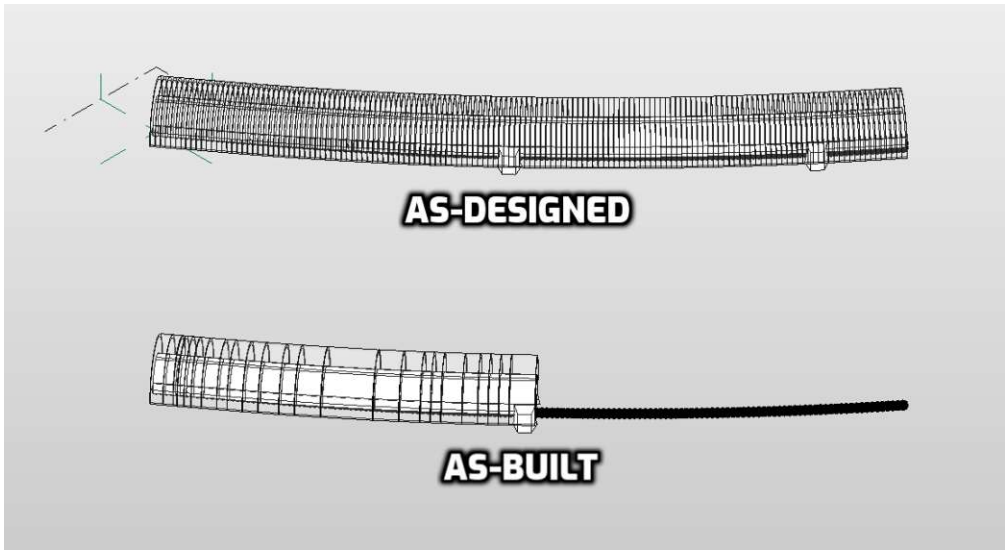

Figure 5.11: The generated as-built 3D BIM model

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

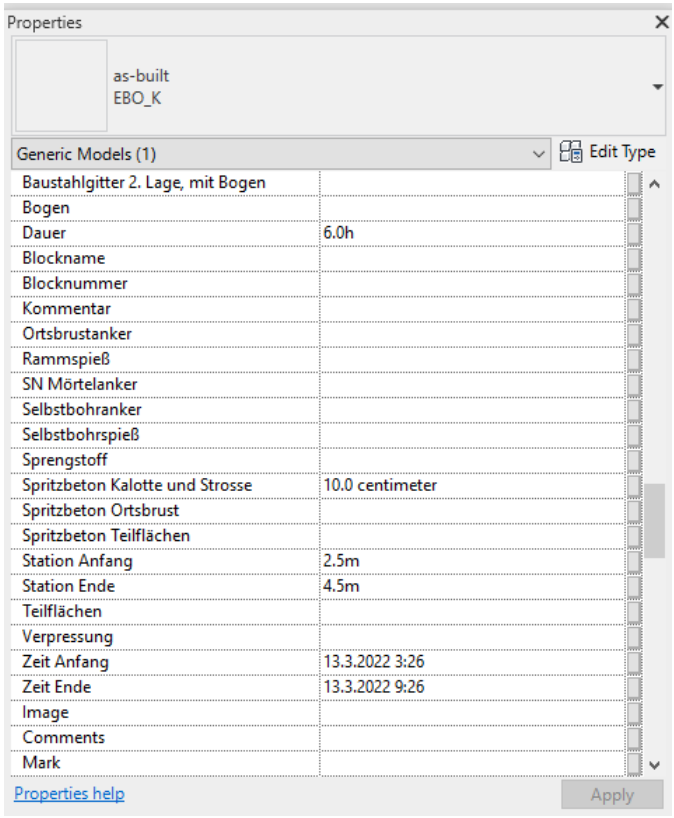

Figure 5.12: Properties of a tunnel element

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Evaluation

We conducted a two-fold evaluation of the prototype for the automatic generation of as-built 3D BIM models. First, to demonstrate our prototype efficacy and effectiveness we compared our prototype against the manual method for the creation of as-built BIM models on several defined KPIs in the Methodology section. Second, we conducted a series of expert interviews from the construction domain to evaluate the ultimate utility of the prototype.

6.1 Prototype Efficacy and Effectiveness

We evaluated our prototype solution against the absence of the prototype (manual creation of as-built 3D BIM models) on predefined KPIs. The table 6.1 presents the results of the prototype efficacy and effectiveness evaluation. The *Model* refers to the as-built 3D BIM model and the *Activity* refers to the activity of creating as-built 3D BIM models.

KPI	Automatic prototype	Manual creation
Activity prerequisites	- parametric as-designed 3D BIM model - formatted construction data	- construction data
Activity performance	- approximately 1-5 minutes, depending on the size of the construction data	- hours, days, weeks...
Activity simplicity	- simple (3-4 steps) - prompt to provide the input data	- complex - manual modeling of elements - extensive knowledge about BIM modeling.
Activity robustness	- bound to the Revit environment - not generic, but can be adapted to use different as-designed 3D BIM models	- robust
Model completeness	- contains all information - real-time data from TIMS	- contains all information
Model level of detail	- can be extended to use additional families, but requires these families as input	- higher level of detail increases modeling time
Model accuracy	- accurate - positioning of elements approximated	- error-prone

Table 6.1: Prototype efficacy and effectiveness evaluation

Next, we compare the results of the prototype for the automatic as-built 3D BIM model generation prototype against the manual creation process.

- **Activity prerequisites** - The manual creation process does not require the as-designed 3D BIM model as input, however, routinely, the first stage of the BIM lifecycle is based on constructing an as-designed BIM model. Thus, the model is usually available, and there is no clear advantage for any method.
- **Activity performance** - the automatic prototype is more performant. It overcomes the creation-time bottleneck of the manual process. This does not only greatly speed up the process, but also opens new BIM model utilisation, which we will discuss in the next section.
- **Activity simplicity** - The automatic prototype process is simpler, requiring the user only to provide the input data, such as the as-designed BIM model and the construction data. On the contrary, the manual creation process can be complex, requiring the user to have extensive knowledge of BIM modeling as well as manual modeling of elements based on available construction data. As a result, the automatic process simplicity enables even the users with limited BIM knowledge to rapidly generate as-built 3D BIM models
- **Activity robustness** - The manual creation process is more robust. In this process, the BIM model creation depends only on the user's BIM expertise. On the other

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

hand, the automatic prototype currently only executes in the Revit environment, and it is adapted to the provided as-designed 3D BIM model. However, it is possible to make the prototype more generic towards the final product phase.

- **Model completeness** - Both methods of the BIM model generation create complete models, with all necessary information. Additionally, the automatic prototype also has a direct connection to real-time construction data, and, after the automatic prototype creates the model, it is still possible to modify and enrich it manually.
- **Model level of detail** - Both methods have the possibilities to provide more levels of detail. For the manual creation process, the higher level of detail proportionally increases the modeling time. The automatic prototype can be extended to provide a higher level of detail by using additionally modeled element families.
- **Model accuracy** - The automatic prototype creates a model with accurate construction data information and approximated positioning on elements. In contrast, manual creation can be error-prone, as all data is created by the user.

6.2 Prototype Utility

To examine the prototype utility we conducted a series of expert interviews. The interviews consisted of two phases, first we presented the problem and demonstrated the prototype, after which we asked our interviewees to provide us with feedback guided by the following set of qualitative questions:

- Would you use the prototype?
- Why/Why not would you use the prototype?
- What would you use the prototype for?
- How would you improve/extend the prototype?
- Additional feedback

Interview round - University of Leoben / Vienna University of Technology / Zentrum am Berg

Participants:

- Uni.Prof. Robert Galler - Tunnel construction / Professor
- Johannes Waldhart Msc. - Tunnel construction / BIM / Research Assistant
- Dipl.-Ing. Galina Paskaleva - Tunnel construction / BIM / Project Assistant

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Interview Summary:

- The prototype would be useful in the tunneling process, if BIM is used, and an as-designed BIM model exists.
- The prototype is useful for a day-by-day overview of the tunneling process. It offers the possibility to immediately find out which support element was built at which tunnel meter, whereas this is currently mostly tracked manually, by hand, and leads to big problems.
- The prototype extends the utility of a classical as-built BIM model, which is constructed only once, as the prototype can generate a 3D BIM model at any instance of the construction process.
- The prototype could be extended to be used for element lifespan supervision, to track the lifespan of each constructed tunnel element.
- The TIMS foreman interface is absolutely useful, and the connection between the TIMS system and the BIM prototype improves the utility of both instances. The prototype could be extended to add volume points to specific elements in the TIMS database.
- The BIM knowledge is a challenge for the prototype utility, as current civil engineer students are not thought how to work with BIM.
- To improve the utility, the prototype could be extended to have a dynamic data update from TIMS, without clicking any buttons.

Interview round - STRABAG Innovation and Digitalisation (SID)

Participants:

- Dipl. -Ing. Stephan Frodl - Tunnel construction / Group Manager
- Dipl. -Ing. Robert Pechhacker - Tunnel construction / Division Manager
- Dr. -Ing. Frank Schley - BIM Function Coordinator
- Dipl. -Ing. Christoph Kellner - BIM Integration
- Dipl. -Ing. Christian Reiter - BIM Specialist

Interview Summary:

- The prototype is useful in the tunneling process, primarily for the tunneling process overview and visualization.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- The prototype could be extended to contain additional graphical elements to improve the visualization potential.
- The prototype could be useful for the tunnel construction billing process (prototype extension would be necessary).
- Due to prototype novelty, prototype utilities are yet to be explored.
- The interviewees expressed an interest in applying the prototype inside STRABAG internal systems. This argument strongly supports the prototype utility.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 7

Conclusion

This master thesis was directed toward digitalization in the tunnel construction domain, as part of the Business Informatics curriculum. By following the design science methodology, we explored the application domain and defined relevant state-of-the-art expertise and existing artefacts. Next, we defined the problem in the BIM model development lifecycle, specifically, the problem that there exists no automatic way to enrich as-designed 3D BIM models with data from the construction phase. By following the agile methodology approach in the design cycle, we successfully constructed a prototype that automatically creates as-built 3D BIM models by enriching the provided as-designed 3D BIM model with real-time construction data. First, we compared our prototype solution's effectiveness against the nonexistence of the prototype, the manual method to create as-built 3D BIM models. Finally, to evaluate the prototype utility, we conducted a series of interview rounds with experts from the construction domain.

After successfully evaluating the prototype we can conclude the following points:

- The automatic prototype is more effective and efficient in creating 3D BIM as-built models than the manual creation method.
- The prototype can generate a 3D BIM model at any instance of the construction process, which extends the utility of classical as-built BIM models.
- The prototype is useful primarily for a digital overview and visualization of the tunneling process. Construction companies already expressed an interest in using the prototype and all possible utilities are yet to be explored.
- The prototype for automatic generation of as-built 3D BIM models with proved effectiveness and usefulness is a contribution to digitalization in the tunnel construction domain.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- The prototype is only useful if BIM is used in the tunneling process, and an as-designed 3D BIM model is provided.
- The current version of the prototype is limited to the Revit environment, and extensive knowledge of the Revit API is necessary for its extension and performance optimization toward a final product.

7.1 Future Work

Some of the future work propositions for the prototype optimization and extension include; implementation in a lower-level programming language to optimize the performance and generation speed, generalization of the prototype to make it more adaptable to different as-designed BIM models, including additional graphical elements to enrich the visualization of the model, extending the model metadata with additional properties, such as geological and geographical data.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Appendix A - Load Data from TIMS Script

```
#! python3

import requests
import credentials
from round import Round
from section import Section
from material import Material
import json
import datetime
import os


def authenticate():
    response = requests.post("https://tunnel.big.tuwien.ac.at:8000/api/login/",
                             json={"user": credentials.username,
                                   "password": credentials.password})
    access_token = "Bearer " + response.text
    headers = {"Authorization": access_token}
    return headers


def get_sections():
    response = requests.get("https://tunnel.big.tuwien.ac.at:8000/api/"
                            "construction.section/", headers=headers)
    sections = []
    for item in response.json():
        section = Section(item['id'], item['name'])
        sections.append(section)
    return sections
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
def get_rounds(section_id):
    response = requests.get(
        "https://tunnel.big.tuwien.ac.at:8000/api/construction.tunnel.round/"
        "?q=[[\"section\", \"=\", {}]]".format(
            section_id), headers=headers)
    rounds = []
    for item in response.json():
        if item['comment'] is None:
            item['comment'] = ''
        start_datetime = convert_tims_datetime_object_to_string(
            item['start_time'])
        end_datetime = convert_tims_datetime_object_to_string(item['end_time'])
        round = Round(item['id'], item['start_chainage'], item['end_chainage'],
                      'Kalotte', item['comment'],
                      start_datetime, end_datetime, str(item['duration']) + 'h')
        rounds.append(round)
    return rounds


def convert_tims_datetime_object_to_string(object):
    return "{}.{}.{} {}:{}".format(object['day'], object['month'],
                                   object['year'], object['hour'],
                                   object['minute'])


def get_material(round_id):
    round_activity_ids = get_round_activity_ids(round_id)
    response = requests.get(
        "https://tunnel.big.tuwien.ac.at:8000/api/construction.tunnel.measure/"
        "?q=[[\"activity\", \"in\", {}]]".format(
            round_activity_ids), headers=headers)
    materials = []
    for item in response.json():
        material = Material(item['measure_definition.']['name'],
                            item['uom.']['name'], item['quantity'])
        materials.append(material)
    return materials


def get_round_activity_ids(round_id):
    response = requests.get(
        "https://tunnel.big.tuwien.ac.at:8000/api/construction.activity/"
        "?q=[[\"round\",\"=\",{}]]".format(round_id),
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
        headers=headers)
    activity_ids = []
    for item in response.json():
        activity_ids.append(item['id'])
    return activity_ids


def get_formatted_data():
    sections = get_sections()
    for section in sections:
        rounds = get_rounds(section.id)
        rounds = sorted(rounds, key=lambda x: x.start_meter)
        for round in rounds:
            round_material = get_material(round.id)
            round.material = serialize_data(round_material)
        section.rounds = (serialize_data(rounds))
    return {"sections": serialize_data(sections)}


def serialize_data(data_list):
    result = []
    for item in data_list:
        result.append(vars(item))
    return result


def store_data(data):
    file_path = create_file_path()
    with open(file_path, 'w') as f:
        json.dump(data, f, ensure_ascii=True)
        print("Data was successfully stored!")


def create_file_path():
    absolute_path_of_script = os.path.dirname(__file__)
    absolute_file_path = get_parent_dir(get_parent_dir(
        get_parent_dir(absolute_path_of_script)))
    return absolute_file_path + '\\data\\tims' + get_current_timestamp() + '.json'


def get_current_timestamp():
    current_datetime = datetime.datetime.now()
    timestamp_string = current_datetime.strftime("%d%m%Y_%H%M%S")
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
    return timestamp_string


def get_parent_dir(directory):
    return os.path.dirname(directory)


headers = authenticate()
data = get_formatted_data()
store_data(data)
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Appendix B - Generate Model Script

```
# -*- coding: utf-8 -*-
from Autodesk.Revit import DB
from Autodesk.Revit import UI
from rpw import db
from rpw.ui.forms import TextInput, Alert, SelectFromList
from not_found_exception import NotFoundException
from pyrevit import forms
from pyrevit import script
import json
import utils as Utils


uidoc = __revit__.ActiveUIDocument
doc = __revit__.ActiveUIDocument.Document
uiapp = __revit__.Application

TUNNEL_AXIS_ELEMENT_TYPES = ['Autodesk.Revit.DB.CurveByPoints']


def create_construction_family(new_family_name):
    print('Creating construction family')
    existing_family = locate_as_designed_family()
    family_doc = doc.EditFamily(existing_family)
    add_construction_parameters(family_doc)
    options = DB.SaveAsOptions()
    options.OverwriteExistingFile = True
    try:
        family_doc.SaveAs(new_family_name, options)
    except Exception as e:
        print('Overriding Revit file permissions')
        loadFamilyCommandId = UI.RevitCommandId.LookupCommandId('ID_FAMILY_LOAD')
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
            UI.UIApplication(uiapp).PostCommand(loadFamilyCommandId)
            raise Exception("Couldn't create family due to Revit file permissions,"
                            " please close the dialog and try again.")


def locate_as_designed_family():
    as_designed_element_name = TextInput('Loading As-designed Family',
                                         default='EBO_K',
                                         description='Please enter the name of'
                                                     ' an used as-designed '
                                                     'model.')
    child_family_element = Utils.get_element(doc, as_designed_element_name)
    return Utils.get_element_family(child_family_element)


def add_construction_parameters(family_doc):
    parameters_tuples = load_construction_parameters()
    for p in parameters_tuples:
        parameter_name = p[0]
        parameter_type = p[1]
        add_identity_parameter(family_doc, parameter_name, parameter_type)


def load_construction_parameters():
    return [
        ('Selbstbohranker', DB.ParameterType.Text),
        ('SN Mörtelanker', DB.ParameterType.Text),
        ('Ortsbrustanker', DB.ParameterType.Text),
        ('Baustahlgitter 1. Lage, ohne Bogen', DB.ParameterType.Text),
        ('Baustahlgitter 1. Lage, mit Bogen', DB.ParameterType.Text),
        ('Baustahlgitter 2. Lage, mit Bogen', DB.ParameterType.Text),
        ('Rammspieß', DB.ParameterType.Text),
        ('Selbstbohrspieß', DB.ParameterType.Text),
        ('Spritzbeton Kalotte und Strosse', DB.ParameterType.Text),
        ('Spritzbeton Ortsbrust', DB.ParameterType.Text),
        ('Spritzbeton Teilflächen', DB.ParameterType.Text),
        ('Bogen', DB.ParameterType.Text),
        ('Verpressung', DB.ParameterType.Text),
        ('Teilflächen', DB.ParameterType.Text),
        ('Sprengstoff', DB.ParameterType.Text),
        ('Kommentar', DB.ParameterType.Text),
        ('Zeit Anfang', DB.ParameterType.Text),
        ('Zeit Ende', DB.ParameterType.Text),
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

```
        ('Dauer', DB.ParameterType.Text),
    ]


def add_identity_parameter(family_doc, parameter_name, parameter_type):
    family_manager = family_doc.FamilyManager
    family_doc_transaction = DB.Transaction(family_doc)
    try:
        family_doc_transaction.Start("ADD PARAMETER")
        family_manager.AddParameter(parameter_name,
                                    DB.BuiltInParameterGroup.PG_IDENTITY_DATA,
                                    parameter_type, True)
        family_doc_transaction.Commit()
    except Exception as e:
        print(e)
        family_doc_transaction.RollBack()


def load_construction_family(family_name):
    print('Loading construction family')
    try:
        transaction.Start('LOAD CONSTRUCTION FAMILY')
        result = doc.LoadFamily(family_name)
        if not result:
            print('Family already loaded, using loaded family')
        transaction.Commit()
    except Exception as e:
        transaction.RollBack()
        raise Exception("Could not load family", e)


def create_tunnel_curve():
    print('Creating tunnel curve')
    as_designed_tunnel_curve = get_existing_tunnel_curve()
    new_xyz = DB.XYZ(200, -200, 0)
    try:
        transaction.Start('CREATE TUNNEL CURVE')
        new_tunnel_curve_ids = DB.ElementTransformUtils.CopyElement(
            doc,
            as_designed_tunnel_curve.Id,
            new_xyz
        )
        new_tunnel_curve = doc.GetElement(new_tunnel_curve_ids[0])
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
            transaction.Commit()
        except Exception as e:
            transaction.RollBack()
            raise Exception(e)
        return new_tunnel_curve


    def get_existing_tunnel_curve():
        result = search_for_tunnel_curve(doc)
        if result:
            return result
        else:
            search_families_having_tunnel_curve()


    def search_for_tunnel_curve(document):
        elements_collector = DB.FilteredElementCollector(document)\
            .WhereElementIsNotElementType()\
            .ToElements()
        for element in elements_collector:
            if element.GetType() and \
                    str(element.GetType()) in TUNNEL_AXIS_ELEMENT_TYPES:
                return element
        return None


    def search_families_having_tunnel_curve():
        available_families = []
        for family in Utils.get_families():
            family = doc.GetElement(family.Id)
            if family.IsEditable:
                fam_doc = doc.EditFamily(family)
                result = search_for_tunnel_curve(fam_doc)
                if result:
                    available_families.append(family.Name)
        content = Utils.format_list_to_string(available_families)
        Alert(title='Error',
              header='Could not locate tunnel curve, '
                     'please open one of the family documents'
                     ' with the tunnel curve',
              content=content)
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
def create_section_block(
        section_element_type_name,
        tunnel_curve,
        beginning_meter,
        ending_meter
    ):
    section_family_element_type = Utils.get_as_built_element(
        doc, section_element_type_name)

    try:
        transaction.Start("CREATE SECTION BLOCK")
        section_family_element_type.Activate()
        new_section_block = DB.AdaptiveComponentInstanceUtils.\
            CreateAdaptiveComponentInstance(
            doc,
            section_family_element_type
        )
        placement_point_a, placement_point_b = get_element_placement_points(
            new_section_block
        )
        placement_point_a.SetPointElementReference(
            create_new_point_on_edge(tunnel_curve, beginning_meter)
        )
        placement_point_b.SetPointElementReference(
            create_new_point_on_edge(tunnel_curve, ending_meter)
        )
        transaction.Commit()
    except Exception as e:
        transaction.RollBack()
        raise Exception("Couldn't create section block", e)

    return new_section_block


def get_element_placement_points(element):
    try:
        placement_points = DB.AdaptiveComponentInstanceUtils.\
            GetInstancePlacementPointElementRefIds(element)
        return doc.GetElement(placement_points[0]), \
               doc.GetElement(placement_points[1])
    except Exception as e:
        raise Exception("Couldn't get placement points", e)
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
def create_new_point_on_edge(edge, position_meter):
    return doc.Application.Create.NewPointOnEdge(
        edge.GeometryCurve.Reference,
        DB.PointLocationOnCurve(
            DB.PointOnCurveMeasurementType.SegmentLength,
            Utils.meter_to_feet(position_meter),
            DB.PointOnCurveMeasureFrom.Beginning
        )
    )


def set_element_parameter(element, parameter_name, parameter_value):
    try:
        transaction.Start('SET PARAMETER')
        parameter = get_element_parameter(element, parameter_name)
        parameter.Set(parameter_value)
        transaction.Commit()
    except Exception as e:
        transaction.RollBack()
        raise Exception("Couldn't set section parameter", e)


def get_element_parameter(element, parameter_name):
    for p in element.Parameters:
        if p.Definition.Name == parameter_name:
            return p
    raise NotFoundException("Parameter not found!", parameter_name)


def add_tunnel_element(start_meter, end_meter, material, comment,
                       start_time, end_time, duration):
    print('Adding tunnel element')
    as_designed_element_name = find_as_designed_element_name(start_meter,
                                                             end_meter)
    if as_designed_element_name is None:
        as_designed_element_name = TextInput(
            'Could not find element at position ('
            + str(start_meter) +' — ' + str(end_meter) + ')',
            description='Please enter the model type name for this tunnel round',
            default='EBO_K')
    section_element = create_section_block(
        as_designed_element_name, as_built_tunnel_curve, start_meter, end_meter)
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
    set_element_parameter(section_element, 'Kommentar', comment)
    set_element_parameter(
        section_element, 'Station Anfang', str(start_meter) + 'm')
    set_element_parameter(section_element, 'Station Ende', str(end_meter) + 'm')
    set_element_parameter(section_element, 'Zeit Anfang', start_time)
    set_element_parameter(section_element, 'Zeit Ende', end_time)
    set_element_parameter(section_element, 'Dauer', duration)
    add_section_material(material, section_element)
    set_section_position(start_meter, end_meter, section_element)


def add_section_material(material, section_element):
    print('Adding section material')
    for item in material:
        set_element_parameter(
            section_element,
            item['name'],
            str(item['value']) + ' ' + item['value_type']
        )


def set_section_position(start_meter, end_meter, section_element):
    print('Setting section position')
    position_parameters = approximate_section_position_parameters(
        start_meter, end_meter, section_element.Name)
    for parameter in position_parameters:
        parameter_name = parameter[0]
        parameter_value = parameter[1]
        set_element_parameter(section_element, parameter_name, parameter_value)


def approximate_section_position_parameters(
        start_meter, end_meter, element_type):
    overlap_elements = find_as_designed_elements_that_overlap_element(
        start_meter, end_meter, element_type)
    return [
        ('Gradientenhöhe_A', Utils.millimeter_to_feet(
            approximate_parameter(overlap_elements, 'Gradientenhöhe_A'))),
        ('Gradientenhöhe_B', Utils.millimeter_to_feet(
            approximate_parameter(overlap_elements, 'Gradientenhöhe_B'))),
        ('Querneigung', DB.UnitUtils.ConvertToInternalUnits(
            approximate_parameter(overlap_elements, 'Querneigung'),
            get_degree_forge_type())),
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
        ('rotXY_A', DB.UnitUtils.ConvertToInternalUnits(
            approximate_parameter(overlap_elements, 'rotXY_A'),
            get_degree_forge_type())),
        ('rotXY_B', DB.UnitUtils.ConvertToInternalUnits(
            approximate_parameter(overlap_elements, 'rotXY_B'),
            get_degree_forge_type())),
    ]


def find_as_designed_element_name(start_meter, end_meter):
    collector = db.Collector(of_class='FamilyInstance')
    elements = collector.get_elements()
    for e in elements:
        try:
            if e.Symbol.Family.Name != 'as-built' and has_blocknummer(e):
                element_start_meter, element_end_meter = \
                    find_as_designed_model_position(e)
                if element_overlap(start_meter, end_meter, element_start_meter,
                                   element_end_meter):
                    return e.name
        except Exception as e:
            continue
    return None


def has_blocknummer(element):
    try:
        p = get_element_parameter(element, 'Blocknummer')
        return True
    except Exception as e:
        return False


def find_as_designed_elements_that_overlap_element(
        start_meter, end_meter, type):
    as_designed_elements = []
    collector = db.Collector(of_class='FamilyInstance')
    elements = collector.get_elements()
    for e in elements:
        if e.name == type and e.Symbol.Family.Name != 'as-built':
            element_start_meter, element_end_meter = \
                find_as_designed_model_position(e)
            if element_overlap(start_meter, end_meter, element_start_meter,
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
                                element_end_meter):
                    as_designed_elements.append(doc.GetElement(e.Id))
    return as_designed_elements


def find_as_designed_model_position(element):
    p = get_element_parameter(element, 'Blocknummer')
    blocknummer = int(p.AsValueString())
    element_start_meter = blocknummer - 1
    element_end_meter = blocknummer
    return element_start_meter, element_end_meter


def element_overlap(
        element_A_start, element_A_end, element_B_start, element_B_end):
    if is_position_between(element_A_start, element_B_start, element_B_end)\
            or is_position_between(
        element_A_end, element_B_start, element_B_end):
        return True
    return False


def is_position_between(current_position, start_position, end_position):
    if start_position <= current_position <= end_position:
        return True
    return False


def approximate_parameter(elements, parameter_name):
    parameter_values = []
    avg_value = 0
    for element in elements:
        parameter = get_element_parameter(element, parameter_name)
        parameter_value = extract_double_from_string(
            clean_string(parameter.AsValueString()))
        parameter_values.append(parameter_value)
    if len(parameter_values) > 0:
        avg_value = sum(parameter_values) / len(parameter_values)
    return avg_value


def clean_string(value):
    value = value.replace("Âř", "")
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
    return value


def extract_double_from_string(string_number):
    for t in string_number.split():
        try:
            return float(t)
        except ValueError:
            return 0


def get_degree_forge_type():
    for u in DB.UnitUtils.GetAllUnits():
        if DB.UnitUtils.GetTypeCatalogStringForUnit(u) == 'DEGREES':
            return u


def load_construction_data():
    print('Loading construction data')
    Alert("Click button \'Load data from TIMS\' to generate current data "
          "snapshot from TIMS. You are also able to add your own construction "
          "data",
          header="Adding Construction Data",
          title="Information")
    file_path = forms.pick_file(
        title='Please select a file containing construction information',
        file_ext='json')
    data = open(file_path, 'r').read()
    return json.loads(data)


def add_construction_data(construction_data):
    print('Adding construction data')
    cross_section_type = SelectFromList(
        'Select cross section type of tunnel rounds you want to generate',
        ["Kalotte", "Strosse", "Sohle"])
    for item in construction_data['sections']:
        for round in item['rounds']:
                add_tunnel_element(
                    round['start_meter'],
                    round['end_meter'],
                    round['material'],
                    round['comment'],
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

```
                round['start_datetime'],
                round['end_datetime'],
                round['duration']
            )


# Transactions are context-like objects that guard any changes made to a
model
transaction = DB.Transaction(doc)

try:
    create_construction_family('as-built.rfa')
    load_construction_family('as-built.rfa')
    as_built_tunnel_curve = create_tunnel_curve()
    data = load_construction_data()
    add_construction_data(data)
    Alert("As-built model generated successfully!",
          header="Automatic Generation Finished")
except Exception as error:
    Alert(str(error), header="Automatic Generation Finished",
          title="User error occured")
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Appendix C - Utility Helper Class

```
from rpw import db
from not_found_exception import NotFoundException


def get_element(revit_document, name):
    collector = db.Collector(of_class='FamilySymbol')
    elements = collector.get_elements()
    for e in elements:
        if e.name == name:
            return revit_document.GetElement(e.Id)
    raise NotFoundException("Element not found", name)


def get_as_built_element(revit_document, name):
    collector = db.Collector(of_class='FamilySymbol')
    elements = collector.get_elements()
    for e in elements:
        if e.name == name and e.Family.Name == 'as-built':
            return revit_document.GetElement(e.Id)
    raise NotFoundException("Element not found", name)


def get_element_family(element):
    if element.Family:
        return element.Family
    raise NotFoundException("Element family not found", element)


def get_families():
    collector = db.Collector(of_class='Family')
    return collector.get_elements()
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

```
def format_list_to_string(list):
    content = ''
    for element in list:
        content += element + '\n'
    return content


def meter_to_millimeter(meter_value):
    return meter_value * 1000


def millimeter_to_feet(millimeter_value):
    return millimeter_value / 304.8


def meter_to_feet(meter_value):
    return millimeter_to_feet(meter_to_millimeter(meter_value))
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] Salman Azhar, Abid Nadeem, Johnny YN Mok, and Brian HY Leung. Building information modeling (bim): A new paradigm for visual interactive modeling and simulation for construction projects. In *Proc., First International Conference on Construction in Developing Countries*, volume 1, pages 435–46, 2008.

[2] Tomo Cerovsek. A review and outlook for a 'building information model'(bim): A multi-standpoint framework for technological development. *Advanced engineering informatics*, 25(2):224–244, 2011.

[3] David Chapman, Nicole Metje, and Alfred Stärk. *Introduction to tunnel construction*. Crc Press, 2017.

[4] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[5] Martin Fowler. *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch*. Addison-Wesley, 2012.

[6] Alan R Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[7] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

[8] Marco Huymajer, Dzan Operta, Alexandra Mazak-Huemer, and Christian Huemer. The tunneling information management system: A tools for documenting the tunneling proicess in natm projects. *Geomechanics and Tunnelling*, 2022.

[9] M Karakuş and RJ Fowell. An insight into the new austrian tunnelling method (natm). In *7th Regional Rock Mechanics Symposium, Sivas, Turkey*, 2004.

[10] Barbara Kitchenham, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1):7–15, 2009.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[11] Christian Koch, Andre Vonthron, and Markus König. A tunnel information modelling framework to support management, simulations and visualisations in mechanised tunnelling projects. *Automation in Construction*, 83:78–90, 2017.

[12] Sang-Ho Lee, Sang I Park, and Junwon Park. Development of an ifc-based data schema for the design information representation of the natm tunnel. *KSCE Journal of Civil Engineering*, 20(6):2112–2123, 2016.

[13] Jeffrey A Livermore. Factors that impact implementing an agile software development methodology. In *Proceedings 2007 IEEE SoutheastCon*, pages 82–86. IEEE, 2007.

[14] Jelena Ninić, Christian Koch, and Janosch Stascheit. An integrated platform for design and numerical analysis of shield tunnelling processes on different levels of detail. *Advances in Engineering Software*, 112:165–179, 2017.

[15] Jelena Ninić, Hoang-Giang Bui, Christian Koch, and Günther Meschke. Computationally efficient simulation in urban mechanized tunneling based on multilevel bim models. *Journal of Computing in Civil Engineering*, 33(3):04019007, 2019.

[16] Ken Peffers, Marcus Rothenberger, Tuure Tuunanen, and Reza Vaezi. Design science research evaluation. In *International Conference on Design Science Research in Information Systems*, pages 398–410. Springer, 2012.

[17] Nicolas Prat, Isabelle Comyn-Wattiau, and Jacky Akoka. Artifact evaluation in information systems design-science research-a holistic view. *PACIS*, 23:1–16, 2014.

[18] Stylianos Providakis, Chris DF Rogers, and David N Chapman. Predictions of settlement risk induced by tunnelling using bim and 3d visualization tools. *Tunnelling and underground space technology*, 92:103049, 2019.

[19] Abubakar Sharafat, Muhammad Shoaib Khan, Kamran Latif, and Jongwon Seo. Bim-based tunnel information modeling framework for visualization, management, and simulation of drill-and-blast tunneling projects. *Journal of Computing in Civil Engineering*, 35(2):04020068, 2021.

[20] Rosemarie Streeton, Mary Cooke, and Jackie Campbell. Researching the researchers: using a snowballing technique. *Nurse researcher*, 12(1):35–47, 2004.

[21] Bilal Succar. Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in construction*, 18(3):357–375, 2009.

[22] Robert Wenighofer, Johannes Waldhart, Nina Eder, and Katharina Zach. Bim use case–payment of tunnel excavation classes–example zentrum am berg. *Geomechanics and Tunnelling*, 13(2):237–248, 2020.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

[23] Xun Xu, Ling Ma, and Lieyun Ding. A framework for bim-enabled life-cycle information management of construction project. *International Journal of Advanced Robotic Systems*, 11(8):126, 2014.

[24] Xianfei Yin, Hexu Liu, Yuan Chen, Yaowu Wang, and Mohamed Al-Hussein. A bim-based framework for operation and maintenance of utility tunnels. *Tunnelling and Underground Space Technology*, 97:103252, 2020.

[25] Katharina Christina Zach. Ein datenmodell zur digitalen dokumentation des bauprozess im tunnelbau. Master's thesis, Montan Univrsity Leoben, 2021.

[26] Ying Zhou, Lieyun Ding, Yang Rao, Hanbin Luo, Benachir Medjdoub, and Hua Zhong. Formulating project-level building information modeling evaluation framework from the perspectives of organizations: A review. *Automation in construction*, 81:44–55, 2017.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.