www.itcon.org - Journal of Information Technology in Construction - ISSN 1874-4753

AUTOMATED TRANSLATION FROM DOMAIN KNOWLEDGE TO SOFTWARE MODEL: EXCEL2UML IN THE TUNNELING DOMAIN

SUBMITTED: February 2023
REVISED: July 2023
PUBLISHED: July 2023
EDITOR: Žiga Turk
DOI: 10.36680/j.itcon.2023.019

Galina Paskaleva, Dipl.-Ing.,
TU Wien, Institute of Information Systems Engineering, Business Informatics Group, Favoritenstr. 9-11, Vienna, A-1040, Austria;
galina.paskaleva@tuwien.ac.at; www.big.tuwien.ac.at

Alexandra Mazak-Huemer, Prof. Dipl.-Ing. Mag. Dr.techn.,
JKU Linz, Department of Business Informatics - Software Engineering, Altenberger Strasse 69, Science Park 3, Linz, A-4040, Austria;
amh@rfte.at; https://se.jku.at/alexandra-mazak-huemer/

Marlène Villeneuve, Associate Professor, PhD
Montanuniversität Leoben, Chair of Subsurface Engineering, Erzherzog Johann-Straße 3, Leoben, A-8700, Austria;
marlene.villeneuve@unileoben.ac.at; https://pure.unileoben.ac.at

Johannes Waldhart, BSc,
iC Consulenten Ziviltechniker GesmbH, Schönbrunner Str. 297, Vienna, A-1120, Austria;
j.waldhart@ic-group.org

SUMMARY: *The development of software tools is a collaborative process involving both the domain experts and the software engineers. This requires efficient communication considering different expertise and perspectives. Additionally, the two groups utilize language and communication tools in disparate ways. This, in turn, may lead to hidden misunderstandings in the requirement analysis phase and potentially result in implementation problems later on, that is difficult and costly to correct. In this paper, we demonstrate the above mentioned challenge via a use case from the tunneling domain. In particular, during the requirement analysis phase for a software capable of handling the data model of the subsoil. The domain experts in the field can best express the complexity of their domain by describing its artifacts, which in most cases are incomprehensible to the software engineers. We outline a method that interleaves requirement analysis and software modeling to enable an iterative increase of the accuracy and completeness of the information extracted from those artifacts and integrated into a flexible software model, which can produce testable software code automatically. Furthermore, we present a prototypical implementation of our method and a preliminary evaluation of the approach.*

KEYWORDS: *domain models, requirement analysis, data model, software engineering*

REFERENCE: *Galina Paskaleva, Alexandra Mazak-Huemer, Marlène Villeneuve, Johannes Waldhart (2023). Automated translation from domain knowledge to software model: EXCEL2UML in the tunneling domain. Journal of Information Technology in Construction (ITcon), Vol. 28, pg. 360-384, DOI: 10.36680/j.itcon.2023.019*

1. INTRODUCTION

In spite of the prominence of digital modelling tools in the Architecture, Engineering and Construction (AEC) industry for many decades, the final output of the design, construction and operation phases remains predominantly analogue and paper-based (Barbosa et al., 2017). Nevertheless, similar to the stationary industry, the AEC industry is transitioning to a completely digital environment as digital tools become more available not only for design but also for construction and operation as well as the projects themselves become more complex. The resulting increased adoption of Building Information Modelling (BIM) (Kaewunruen et al., 2018) has highlighted the critical role communication plays, not just between various domains within the AEC industry, but also between those and the Software Engineering (SE) domain (Tallgren et al., 2020). On the one hand, it is vital that geologists, civil, structural and geotechnical engineers, and contractors communicate in the language of their domain. On the other hand, it is just as important that the software engineers developing the tools for BIM understand the requirements of those domains, i.e., have them translated into their own domain language. For example, (Tallgren et al., 2020) and (Arayici et al., 2006) show that shared understanding is a major motivating factor for participation in the development process of a Computer Integrated Construction (CIC) software.

Modeling at various levels of abstraction is a well-established design step in the software development process. It enables software engineers to structure software into platform-independent constituent modules prior to coding (OMG, 2022). The communication between domain experts (or, in this context, users of software) and software engineers in this design step lays the foundation of the software functionality, which should ideally be well understood and agreed on by all stakeholders (Parsanezhad et al., 2016). However, there is a significant disparity in the communication tools employed by both groups (Arayici et al., 2006). While domain experts from the AEC industry use natural language and spreadsheets to express their domain requirements, software engineers rely on formal languages, such as the Universal Modelling Language (UML). Using such a formal well-defined language avoids any ambiguity and establishes structure, both in terms of data and function.

Spreadsheet tools, such as Excel™, are commonly used in the AEC industry for defining requirements, for design, for monitoring operations (David et al., 2017), and even as a database. This is largely due to their ability to store and operate on large amounts of data in a two-dimensional matrix structure. For this reason, most domain experts are proficient in using spreadsheets, including for defining requirements for software design. Such requirement definitions contain the standardized terms of the domain. For example, expressing the *compressive strength* of a material in terms of *cohesion* and *friction angle* enables unambiguous communication within the domain. However, this type of standardization of terms is of no help to the software engineer, who has to develop software expressing the semantics of such domain-specific standardized terms.

All of this makes sense, since the AEC industry is quite knowledge-intensive, and requires the capture of both explicit and tacit knowledge (Zahedi et al., 2022). In order to illustrate the challenge when interleaving both worlds in more detail, we present the following use case from the geotechnics domain.

1.1 Motivating Example

Multiple models produced in the geotechnics domain contain interpretations based on tests, observations, extant surveys, maps, etc. (German Tunnelling Committee (ITA-AITES), 2022b). Those interpretations are expressed as distributions of various properties along the tunnel alignment or, in the case of larger underground structures, within a three-dimensional volume. The properties may include rock type and quality, location and orientation of contact surfaces, etc., grouped into categories, such as geology, hydrogeology, geotechnics, geochemistry, and others. Fig. 1 makes it evident that these categories are concurrent, i.e. each part of the subsoil is categorized multiple times according to different aspects. For example, the same location along the tunnel alignment (the red mark in Fig. 1) can be categorized simultaneously as *G3*, *H1*, and *GT1*. This is intuitively clear to the domain experts due to the experience and knowledge they possess: factual ("know what"), operational ("know-how"), normative ("know why"), etc. (Häußler and Borrmann, 2021). However, in the context of software development, the following question arises: how to translate that knowledge into requirements of the software engineering domain?

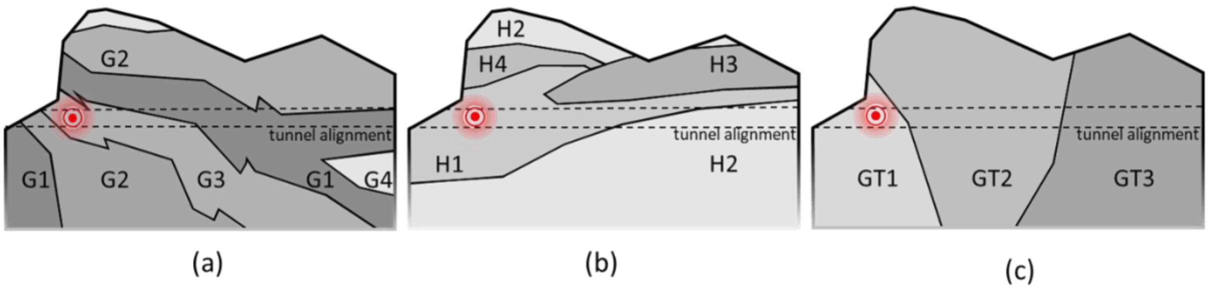


FIG. 1: Some of the different aspects of subsoil: geology (a), hydrogeology (b), and geotechnics (c). Each aspect produces a different segmentation of the longitudinal section along the tunnel alignment.

The domain experts have to articulate their needs, i.e., make their tacit understanding of the knowledge structure in their domain explicit. In Fig. 2, there is a small (intentionally blurred) excerpt of one such attempt at communication via spreadsheets. The full document contains about ten times the amount of information shown in the image. It is evident that, without knowledge of formal languages needed for Software Requirements Engineering (SRE), it is a difficult and time-consuming task to locate that structure in the artifacts produced by the domain, e.g., design drawings, geological long sections, etc. What's more, such artifacts contain information, not the knowledge it underpins (Zahedi et al., 2022). To convey the *knowledge* to the software engineering team via requirements without distortion or loss is a cumbersome and complex task. The approach described in this work aims to provide a standardized workflow that leads to the systematic handling of this task.

FIG. 2: An example of a data structure definition by domain experts in geotechnics as a spreadsheet.

The rest of this work is structured as follows. In section 2, we give an overview of the SRE process with emphasis on the early platform-independent stages. In section 3, we describe the methods we apply to accommodate the needs of both domain experts and software engineers. Section 4 examines comparable approaches. Section 5 provides a critical view of our approach and potential venues for its further development. Section 6 concludes this work.

2. PRELIMINARIES

In this section, we present the fundamentals of the early stages of software development to illustrate the critical role efficient communication, including translation, between software engineers and future software users (i.e., domain experts) plays in the process.

2.1 Capturing Domain Knowledge

In 2009 (Christiansson et al., 2009) identified the missing ontologies on multiple levels, both abstract and specific, as a major drawback in the development of tools for the AEC industry. Since then, considerable effort has been invested in that field. Some of the international standardization initiatives in the AEC domains, which usually start with the definition of a dedicated domain ontology, include the Industry Foundation Classes (IFC) (buildingSMART, 2022), the DAUB recommendation (German Tunnelling Committee (ITA-AITES), 2022b), the multiple Open Geospatial Consortium (OGC) standards, available at https://www.ogc.org/docs/is, and many national and regional guidelines.

These standards attempt to capture domain knowledge, a process which typically utilizes requirements engineering techniques. In essence, it is an elaborate translation workflow from the language of the domain experts into the language of software engineers. It is exactly this translation process that we focus on in this work.

2.2 Software Requirement Engineering

There is no universally agreed-on terminology for the phases of software development (Jaffe, 2021). Nevertheless, Fig. 3 depicts one of the more common ones, which we will use to situate the SRE process. It produces the requirement specification and is one of the first and most critical steps in software development. It starts with the conceptual phase, which involves articulating user expectations and researching available technologies (see the top left part of the curve in Fig. 3). User expectations can be formulated as a user story, e.g., a description in a Natural Language (NL) followed by a conversation and confirmation tests (Cohn, 2004). User expectations can also be expressed in a formal language, or any other form, e.g., as a spreadsheet.

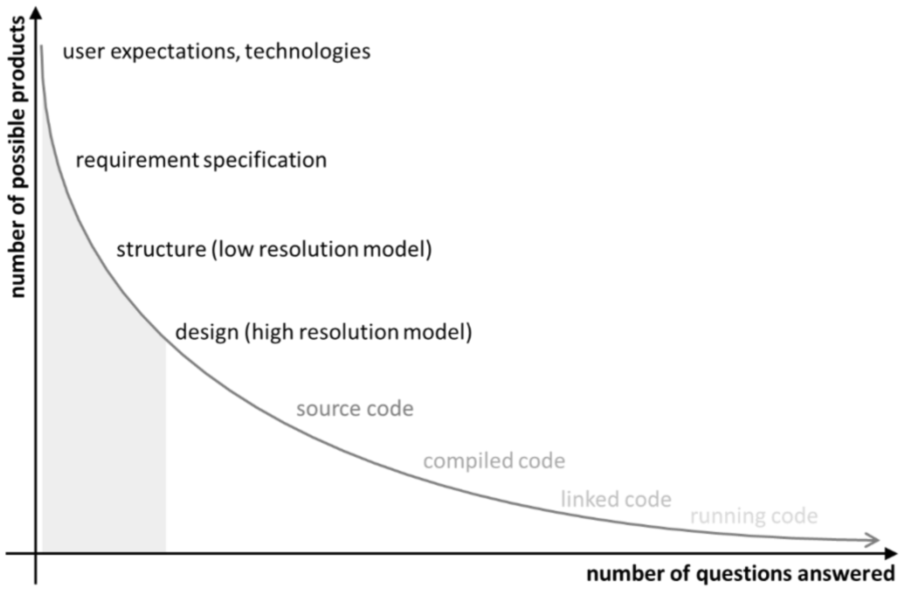


FIG. 3: The relationship between the adaptability of a software solution to its level of completion, based on the cone of uncertainty (Boehm, 2001).The shaded part underneath the curve indicates the extents of the SRE process.

2.2.1 Definitions

The SRE process is iterative and fuzzy (Parsanezhad et al., 2016, Jaffe, 2021). Despite this, there are some key quality assurance milestones whose definitions we will list here. Just as the DIN EN ISO 9000-2015-11 (https://www.beuth.de/de/norm/din-en-iso-9000/235671064) defines the terms "quality" and "requirement" in the wider context of quality management systems (Häußler et al., 2020), so does the ISO/IEC/IEEE 29148:2018 (ISO/IEC, 2018) in the context of software engineering.

- **3.1.19 requirement** “statement which translates or expresses a need and its associated constraints and conditions”;
- **3.1.20 requirements elicitation** “use of systematic techniques, such as prototyping and structured surveys, to proactively identify and document customer and end user needs”;
- **3.1.25 requirements validation** “confirmation that requirements (individually and as a set) define the right system as intended by the stakeholders”;
- **3.1.26 requirements verification** “confirmation by examination that requirements (individually an as a set) are well-formed”.

(ISO/IEC, 2018) goes on to list some of the criteria of a well-formed requirement: (i) it solves a particular problem or achieves a specific objective, (ii) it is qualified by measurable conditions, (iii) it can be subject to constraints, (iv) it is always about the system, never about the user, and (v) it is verifiable.

Depending on the domain, additional norms may have to be translated into requirements, e.g., the ISO 26262 (https://www.iso.org/obp/ui/}iso:std:iso:26262:-1:ed-2:v1:en) for critical systems, such as electronic components in vehicles, or the DO-178C Software Considerations in Airborne Systems and Equipment Certification in avionics (Jaffe, 2021). As we can see, translation plays a central role in the SRE process.

2.2.2 Best Practices and Issues

According to (Arayici et al., 2006), in addition to the recommendations from the previous section, best practices in SRE include as follows: good user involvement, allocation of 15 to 30% of project resources to SRE, providing specification templates and examples, developing models and prototypes, and maintaining traceability between requirements and solutions.

When these are not followed, multiple issues can arise. Failing to involve the users to a sufficient degree or to consider their feedback properly leads to misunderstanding, misinterpretation of the requirements, incompleteness, and inconsistency (Arayici et al., 2006, Häußler and Borrmann, 2021). Missing specification templates and examples may lead to hidden divergence of definitions, e.g., of fundamental terms like "constraint" or "objective", even "requirement" (Parsanezhad et al., 2016, Dalpiaz et al., 2018). Not building models or prototypes reduces the user reviews to observations after the deployment phase (last in Figure 3), two thirds of which could be effectively useless to the software developers (Panichella and Ruiz, 2020). Even the traceability between requirements and solutions can become difficult in cases of inconsistently classified requirement types in guidelines, e.g., high-level and low-level safety requirements in avionics (Jaffe, 2021). Finally, even the scope and scale of the axes in Fig. 3 could shift dramatically, as both starting and stopping point of the software development may differ, as well as the needed level of abstraction of the requirements and the number of refinement steps, which depend on the project phase (Jaffe, 2021) and the specific domain (Parsanezhad et al., 2016).

All those shortcomings affect the time and cost aspects of any software development project (Osama et al., 2021). For example, in middleware projects, about 50% of defects and about 80% of rework can be traced back to a poor requirement specification (Uddagiri et al., 2020). In other words, the quality of the requirement specification has a massive influence on the software development process in its most adaptable, and therefore most crucial, phases.

2.2.3 Location and Duration of Software Requirements Engineering

As Fig. 3 shows, the degree of adaptability of developing software is the highest right at the beginning, during the first four steps, when there are still a large number of open decisions. Even between articulating user expectations and the formalization of those as a requirement specification, there is a sharp drop in adaptability. However, as was discussed in the previous section, without a working software model to test user assumptions, and without good traceability, or a translation workflow, it is difficult to produce a full set of requirements.

On the other hand, if the SRE process extends over the phases of software structure definition and design, user feedback on the software models can help revise the requirements iteratively (Parsanezhad et al., 2016) up until the production of source code and even beyond since software models are capable of automatically generating code (Brambilla et al., 2017). It is exactly this extension of the time available for adapting the requirement specification which we aim to achieve with our approach.

2.2.4 Formulating Requirements

Requirements can be formulated in a multitude of ways. NL formulations are in wide use, but even those are subject to formal requirements (ISO/IEC, 2018). User stories, for example, could be evaluated for proper syntax, sound semantics (e.g., problem-oriented, unambiguous, and conflict-free), and pragmatics (e.g., uniqueness, independence, and completeness) (Dalpiaz and Brinkkemper, 2018). In general, any requirement can be expressed as a user story, e.g. the design process in any of the AEC domains (Zahedi et al., 2022).

In addition, requirements can be formulated in a multitude of formal languages. Use case diagrams, which are part of the Universal Modeling Language (UML) (see https://www.uml.org/), can capture user interactions with a system, but not necessarily the interactions within the system (Arayici et al., 2006). The Business Process Model and Notation (BPMN) (see https://www.bpmn.org/) is often used as a tool for the definition of the Information Delivery Manual in the AEC industry (Häußler et al., 2020) and can be applied as formal notation in SRE as well. Automated quality assurance methods can be applied much easier to formal requirement formulations. For critical systems, such as electronic components in vehicles, ISO 26262 even mandates the application of formal methods for software quality assurance, which necessitates the formulation of formal requirements only (Osama et al., 2021).

The automated quality checking of NL requirements is more challenging and expensive (Dietsch et al., 2020). In addition, such requirements do not provide an automated path towards simulating functionality, which delays the validation phase and could incur unnecessary implementation effort and cost. Various pipelines may include required phrase matching, sentence structure compliance, relationships between concepts, and ambiguity checks (Osama et al., 2021), or text mining techniques for the extraction of implied meaning (Saxena and Chakraborty, 2014). However, formal confirmation tests are seldom integrated in the process correctly (Cohn, 2004). For this reason, there are multiple methods for the formalization of NL requirements, which we will present in more detail in section 4.

In general, NL requirement specifications enjoy considerable popularity due to their apparently informal style. However, from the point of view of quality assurance, this can be misleading. As (Lucassen et al., 2015) point out, there are multiple criteria NL requirements have to fulfill to be considered well-formed - syntactic, semantic, and pragmatic. We will not list all of them here. Instead, we will concentrate on our motivating example and the criteria relevant to the requirements we need to elicit from geotechnics domain experts.

2.3 Communication Support Requirements: Motivating Example Part 2

For this reason, we must further specify the type of software we aim to design. In section 1.1, we spoke about categorizing the subsoil and connecting observations and interpretations in the geotechnics domain, which suggests information management and exchange support, concerns common to the entire AEC industry. Support for communication workflows is typically provided by middleware. However, many of the tools developed to facilitate pre-construction communication do not enjoy wide user acceptance because they require a change in the already established domain and cross-domain workflows. This leads to pronounced user resistance (Tallgren et al., 2020), which is one of the reasons for the high activity in international standardization we mentioned in section 2.1. The aim of this standardization is the development of data models suitable for communication support. Naturally, the first step in any such process is the requirements elicitation from domain experts. What makes the process particularly challenging, in this case, is the varied background of those experts - they come from different countries, have worked under different guidelines and conditions, they may be involved in various branches of industry, governance, or research. *For this reason, the use case we aim to address with our approach is the SRE process during the development of a data model suitable for communication support, specifically within the geotechnics domain.* We will discuss the applicability of our approach to other domains in the AEC industry in section 5.

If we were to attempt the typical NL requirement specification, according to (Lucassen et al., 2015), we would have to develop a strategy for dealing with, at the very least, the following semantic features:

- **(F1) Ambiguity**. For example, "*is* a lithological unit" can mean "*is of type* lithological unit" or "*is part of* a lithological unit" or "*is a specialization* of lithological unit" due to the ambiguity of the word "is" (Kühne, 2006). On the other hand, synonyms (Dalpiaz et al., 2018) or homonyms can also lead to ambiguity. For example, "material" might mean "any substance found in a borehole" for the domain expert, but it can easily be interpreted as a "man-made material", such as concrete, by the software engineer as this is the more common usage of the term in the AEC industry. For a full taxonomy of ambiguity, see also (Berry and Kamsties, 2004);
- **(F2) Technical jargon**. Avoiding technical jargon of any kind is one of the central premises of the user story (Cohn, 2004). However, it is not possible to achieve it in a very specialized field, such as geotechnics;
- **(F3) Lack of conceptual soundness**. For example, a requirement for a particular specialized view of the data when designing a data model is not sound, as it addresses a feature that cannot be covered by a data model, but is customarily delegated to the user interface;
- **(F4) Overlaps or contradictions**. A NL requirement specification has no safeguards against semantic overlap or conflict. For example, the same concern could be addressed by multiple (and possibly contradictory) requirements with no formal mechanism in place to detect the overlap or conflict (Dalpiaz et al., 2018);
- **(F5) Non-problem-oriented statements**. Requirements should act as translations of *functional concepts* into *solution concepts* (Ye et al., 2009). If one or both are unclear or missing, the requirement is not well-formed (ISO/IEC, 2018). For example, "Chainage shall be associated with a tunneling class" or "It should be possible to create tendering documents" are not well-formed requirements, since they both seem to contain only functional concepts. However, "Chainage shall be associated with a

tunneling class to enable the creation of tendering documents." contains both and is, therefore, a well-formed requirement. It is of note that there is a difference between a *solution concept* and an *implementation instruction*. If the above mentioned requirement was as follows, "Chainage shall be associated with a tunneling class to enable the creation of tendering documents in software X.", it again would not be considered well-formed.

Furthermore, the following pragmatic aspects should be considered (Lucassen et al., 2015):

- **(F6) Validatability**. The specification should not contain requirements whose fulfillment cannot be measured (ISO/IEC, 2018). For example, in our case, a requirement of the user interface has no place in the specification of a data model;
- **(F7) Uniformity**. All requirements should conform to a pre-defined template and cover a scope of similar size;
- **(F8) Uniqueness**. Duplicate requirements should be removed;
- **(F9) Explicit Dependencies**. If there is a tacit dependency between requirements it should be made explicit. Otherwise it cannot be considered or validated;
- **(F10) Completeness**. The requirement specification should cover 100% of the software functionality. In practice, completeness is achievable only over multiple iterations.

In summary, we want to enable domain experts and software engineers to design a data model cooperatively by taking the above-listed aspects into account. In other words, we want to facilitate a high-quality translation process, not just between different languages, but between different perspectives, different prioritization hierarchies, and even between different work rhythms. This is, obviously, a huge and error-prone task to undertake, which is why we aim to provide as much flexibility as possible without sacrificing precision in places where it is necessary. A very good fit for these requirements is software *modeling* as it allows us to vary the level of abstraction fairly independently across the model, but also to generate highly specialized code, if needed. Here we will give a brief outline of the concept of modeling, specifically modeling data, information, and knowledge.

2.4 Modeling

The term modeling will be used extensively in this work, since is applies to multiple aspects of digitalization, both from the domain experts' and the software engineers' point of view. For the domain expert (e.g., in geotechnics) in our use case, modeling means building an abstract formal representation of the relevant domain concepts (Kühne, 2006), including defining the domain taxonomy and the relationships between its elements. In addition, this can include the formal representation of relevant use cases, processes, data exchange workflows, etc. In software development, modeling means building an abstract platform-independent representation of the software based on the requirement analysis and specification. These two types of modeling often run in parallel, especially in the earliest stages of the software design.

In section 2.1 we showed that a standard, i.e., a domain model, has the task of capturing domain knowledge. A standard has, by definition, a certain degree of universality, even if it is within a single domain, which precludes it from representing a single person's skill, understanding, or opinion. However, within the group of domain experts developing the standard some common understanding could emerge. Furthermore, effective communication could facilitate reflection and synthesis, whose results can be captured by the developing model. Since knowledge is a property of people (Rowley, 2007), the model cannot capture it directly, but the *modeling process* can - by (formally) guiding some of the processes that convert information into knowledge. This is a significant aspect of our approach.

2.5 Software Modeling Approaches

As we already established, the development of a software starts with eliciting requirements. The waterfall method necessitates a complete requirement specification before implementation even starts (Royce, 1970, Boehm, 1987). In a complex field with multiple domains involved, such as the AEC industry, it is unrealistic to expect all requirements to emerge before the users have had the opportunity to test the software and, thus, test their assumptions about their own domain (Flewelling, 2018). For this reason, the agile method is much more suitable for our approach. In it, requirement elicitation can be initiated and the resulting specification gradually refined with each development cycle, while continuously incorporating user feedback into the requirement set.

The implementation itself does not start with producing code. It starts with software models defined in a suitable formal language. Models can play the role of drafts, guidelines or programs (Combemale et al., 2017) and have some major advantages over code. They are platform-independent and can be used to generate code automatically in any programming language (Brambilla et al., 2017). Functionality and conformity to the requirements can be tested automatically. They are particularly well-suited for Rapid Application Development (RAD) in the agile method. In summary, software models put the focus not on a particular data item or process, but on the adaptivity of software development, which is exactly the aspect where communication and translation between domain experts and software engineers play the largest role.

One of the most widely used modelling languages in software engineering is UML. Any type of UML model in the context of the agile method has the ability to facilitate communication between domain experts and software engineers by producing an easily adaptable formal schema of the way information is structured. From the perspective of domain experts, its ability to produce compact visualizations of the discussed schema has a significant advantage over text-based approaches. In general, formal languages with a graphical concrete syntax have wider acceptance in the AEC industry, e.g. the Grasshopper (see https://www.grasshopper3d.com/) plug-in for Rhinoceros 3D (see https://www.rhino3d.com/), Dynamo Sandbox (see https://dynamobim.org/download/) for Autodesk REVIT (see https://www.autodesk.de/products/revit/overview?term=1-YEARŹtab=subscription) and Marionette (see https://www.computerworks.de/produkte/vectorworks/vectorworksarchitektur/marionette.html) for Vectorworks (see https://www.vectorworks.net/en-GB/2023?igeo=true) (Preidel and Borrmann, 2016). What such languages have in common is a graph which carries information not only by means of labelling and connectivity, but also through the use of color and shape. Furthermore, a graph can represent not only entities and relationships (as in a database), but a procedural algorithm (Ilčík et al., 2015), a workflow, a transformation both in software modeling (Brambilla et al., 2017) and in engineering (Kolbeck et al., 2022), a state transition, and many more. Modelling requirements directly as graphs, or translating them to graphs, has the potential to reduce the amount of tacit assumptions present in NL formulations by explicitly identifying elements and relationships that are normally regarded as "intuition" or "expert knowledge". Of particular relevance to our approach is that *graph rewriting* is well suited to depicting workflows in problem solving. Such graphs and the rules that operate on them have been shown to help in the formalization of domain knowledge (Kolbeck et al., 2022), which is exactly what we want to achieve.

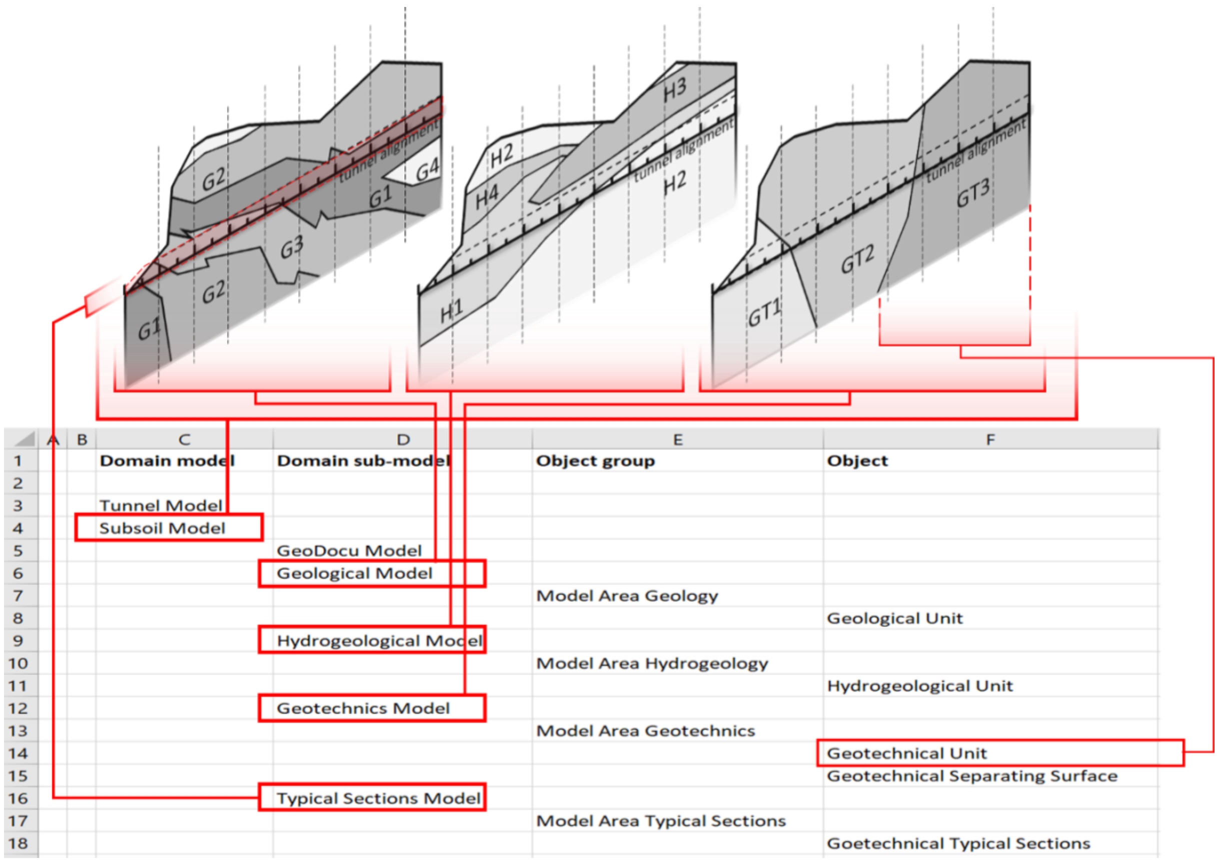
However, graphs are limited by the amount of visual clutter they produce when sufficiently large (Dalpiaz et al., 2018). This necessitates careful modularization and separation of concerns. Hierarchical graphs, for example, such as trees, port graphs, or hierarchical hypergraphs, can be used since they can be applied at both very coarse and very fine levels (Kolbeck et al., 2022).

Finally, software modelling includes automated conformity checks. Hard-coding design standards or guidelines for compliance checking in the engineering domains has proven to be laborious for the software engineer and to lower user acceptance (Häußler et al., 2020, Preidel and Borrmann, 2016). Conformity checks depicted, for example, as graph matching rules that are fully editable by domain experts could alleviate the situation during the SRE process (Kolbeck et al., 2022).

3. APPROACH

In this section, we present our approach to interleaving requirement analysis and software modelling accompanied by continuous automated translation between domain experts and software engineers. We will show the resulting benefits by employing it to determine the requirements on the data structure for holding subsoil information in our motivating example.

Fig. 4 and Fig. 6 show two of many possible data structures for modeling the subsoil expressed as spreadsheets. The 2022 DAUB recommendation (German Tunnelling Committee (ITA-AITES), 2022b) contains the concept of an element hierarchy (see the top row in the table in Fig. 4): A *Domain model* containing *Domain submodels*, which in turn contain *Object groups*, which consist of *Objects*, which may contain *Partial objects* (not shown in the spreadsheet). Applied to the subsoil it produces a parallel data structure for geology, hydrogeology, and geotechnics by declaring those as *Domain sub-models* (German Tunnelling Committee (ITA-AITES), 2022b), which can be subdivided down to *Geological*, *Hydrogeological* and *Geotechnical Units*, respectively, on the *Object* level (see the last column in Fig. 4).



	A	B	C	D	E	F
1			Domain model	Domain sub-model	Object group	Object
2						
3			Tunnel Model			
4			Subsoil Model			
5				GeoDocu Model		
6				Geological Model		
7					Model Area Geology	
8						Geological Unit
9				Hydrogeological Model		
10					Model Area Hydrogeology	
11						Hydrogeological Unit
12				Geotechnics Model		
13					Model Area Geotechnics	
14						Geotechnical Unit
15						Geotechnical Separating Surface
16				Typical Sections Model		
17					Model Area Typical Sections	
18						Goetechnical Typical Sections

FIG. 4: A data model extracted from the DAUB recommendation 2022 (German Tunnelling Committee (ITA-AITES), 2022b).

The standards for geoscience and water observation developed by the OGC propose the data structure shown in Fig. 5. However, when referenced by domain experts, a spreadsheet representation, such as in Fig. 6, is more widely used, due to convenience.

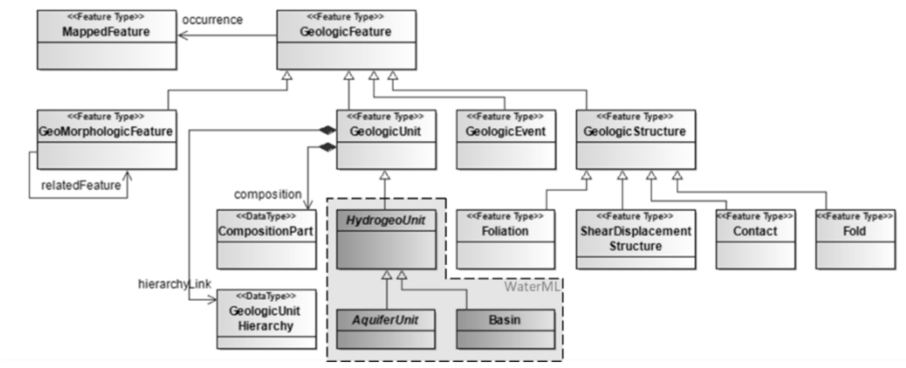
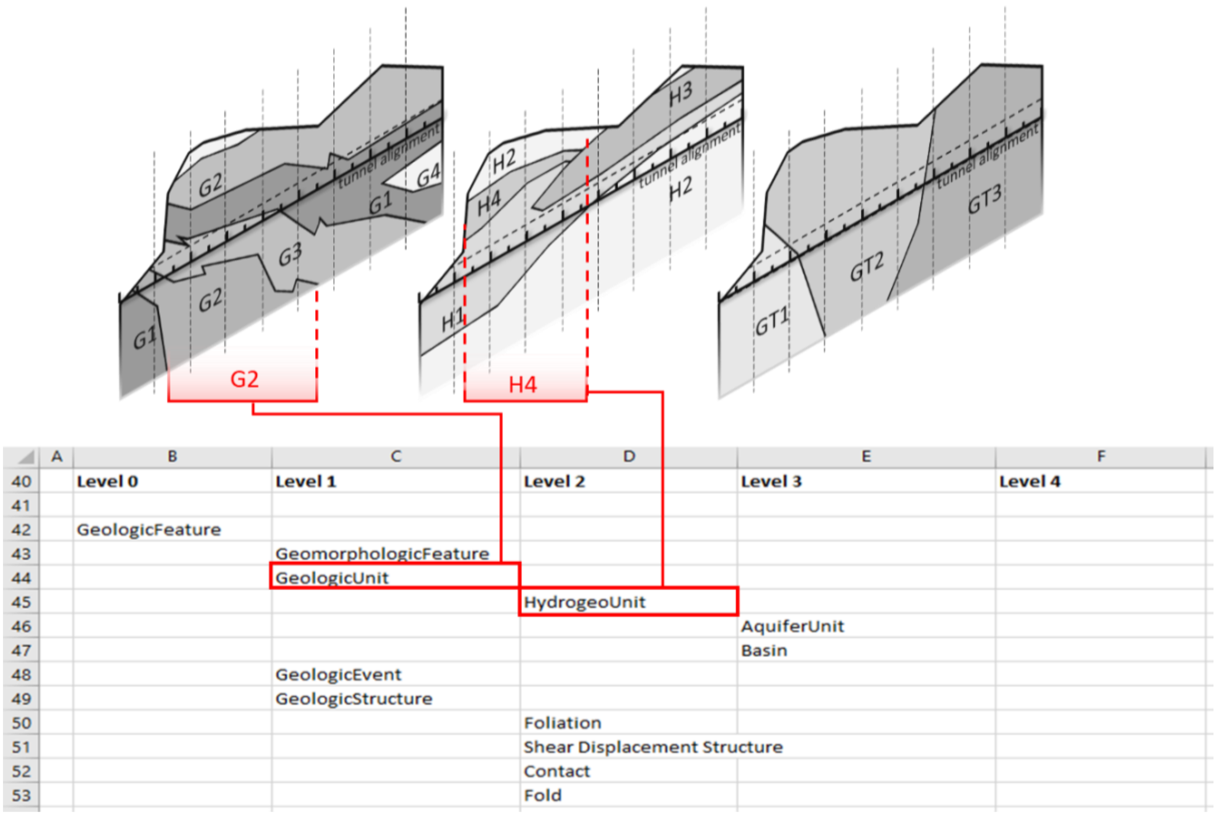

<<Feature Type>> MappedFeature
occurrence
<<Feature Type>> GeologicFeature
<<Feature Type>> GeoMorphologicFeature
relatedFeature
<<Feature Type>> GeologicUnit
<<Feature Type>> GeologicEvent
<<Feature Type>> GeologicStructure
composition
<<DataType>> CompositionPart
HydrogeoUnit
<<Feature Type>> Foliation
<<Feature Type>> ShearDisplacement Structure
<<Feature Type>> Contact
<<Feature Type>> Fold
hierarchyLink
<<DataType>> GeologicUnit Hierarchy
WaterML
AquiferUnit
Basin

FIG. 5: An excerpt of the UML class diagrams for GeoSciML and WaterML defined by OGC (see https://www.ogc.org/docs/is).

If we compare the spreadsheets in Fig. 4 and Fig. 6, they show not just different conceptual models, but also a different view of each of those models. Fig. 4 displays the containment structure of the conceptual model, whereas Fig. 6 - the type structure. For example, if we compare the relationship between the elements in columns C and D in each table, in the case of Fig. 4, we have a *Subsoil Model* "containing" a *Geological Model*, a *Hydrogeological Model*, a *Geotechnics Model*, and a *Typical Sections Model*. In the case of Fig. 6, we have a *GeologicUnit* not containing, but being the "generalization of", *HydroGeoUnit*. In the UML diagram in Fig. 5 the difference between the containment and the generalization relationship is clear due to the UML syntax. On the one hand, *GeologicUnit* contains *CompositionPart*, and on the other, it is the generalization of *HydroGeoUnit*, which is the generalization of *AquiferUnit* and *Basin*.

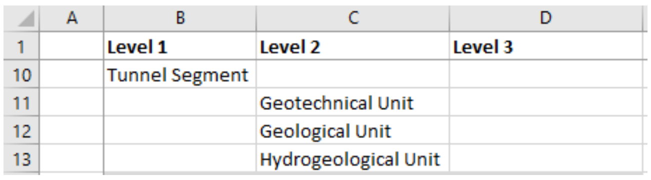
It is to be noted that the excerpts of the shown models do not contain information about all the relationships between the various data elements. We leave this out for brevity. However, such considerations are of vital importance in practice and contribute significantly to the complexity of any data model. The question we have to answer is, how can all these different modeling possibilities be communicated between domain experts and software engineers?



	A	B	C	D	E	F
40		Level 0	Level 1	Level 2	Level 3	Level 4
41						
42		GeologicFeature				
43			GeomorphologicFeature			
44			GeologicUnit			
45				HydrogeoUnit		
46					AquiferUnit	
47					Basin	
48			GeologicEvent			
49			GeologicStructure			
50				Foliation		
51				Shear Displacement Structure		
52				Contact		
53				Fold		

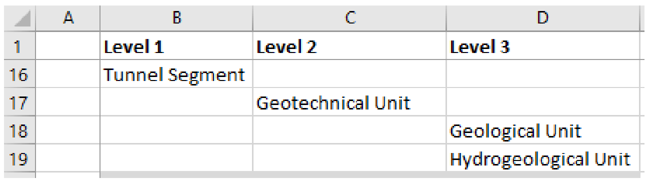
FIG. 6: A spreadsheet representation of the data model extracted from the GeoSciML and WaterML standards.

In section 2.2 we outlined the different degrees of formalisation a user requirement can adopt. On the one hand, even synonyms could be misleading, since, in most cases, they represent similar but not identical concepts (Dalpiaz et al., 2018), which speaks for a high degree of formalisation. On the other hand, there is a danger that this might have a negative effect on flexibility (Parsanezhad et al., 2016) and impede communication and domain expert involvement, which speaks for formalisation in a familiar environment for the domain expert, e.g. spreadsheets (David et al., 2017). According to the investigation presented by (Uddagiri et al., 2020), spreadsheet templates are suitable for middleware projects with up to 100 interfaces, which should suit our motivating example. Furthermore, the report on the standardisation activities at buildingSmart International (bSI) on the projects IFC-Road and IFC-Rail demonstrates a successful utilization of Excel as a tool for gathering "data requirements" (Borrmann et al., 2020). This gives us our means of communication.



	A	B	C	D
1		Level 1	Level 2	Level 3
10		Tunnel Segment		
11			Geotechnical Unit	
12			Geological Unit	
13			Hydrogeological Unit	

FIG. 7: A first draft of the template to be used by the domain experts. Levels are interpreted as associations.



	A	B	C	D
1		**Level 1**	**Level 2**	**Level 3**
16		Tunnel Segment		
17			Geotechnical Unit	
18				Geological Unit
19				Hydrogeological Unit

FIG. 8: The same draft of the template as in Fig. 7. Levels are interpreted as both associations and containers.

In addition, in order for this communication to be effective, we have to establish a formally well-defined workflow including both sides of the conversation (Tallgren et al., 2020, Panichella and Ruiz, 2020), or, in other words, a fast and accurate translation. As we noted in section 2.4, it is the process of modelling and communication that has the potential to extract not merely information, but knowledge. What's more, a well-structured workflow has been shown to improve user acceptance even in SRE (Dietsch et al., 2020). Fig. 9 shows such a workflow as a UML activity diagram. Several actions are not part of the typical software development process. For example, *Action 1* and *Action 2* involve the production of a spreadsheet specification template by the software engineer (see Fig. 7) and its testing, with a focus on usability, by the domain expert, respectively. *Action 3* allows the domain experts to express their domain knowledge in a familiar environment, i.e. the spreadsheet, and receive instant feedback in the form of an automatically translated UML class diagram (see *Action 4a*) that provides a graphical representation of that knowledge. More importantly, this feedback provides the software engineers with a software model that can be used as is for requirement testing (see *Action 4b*).

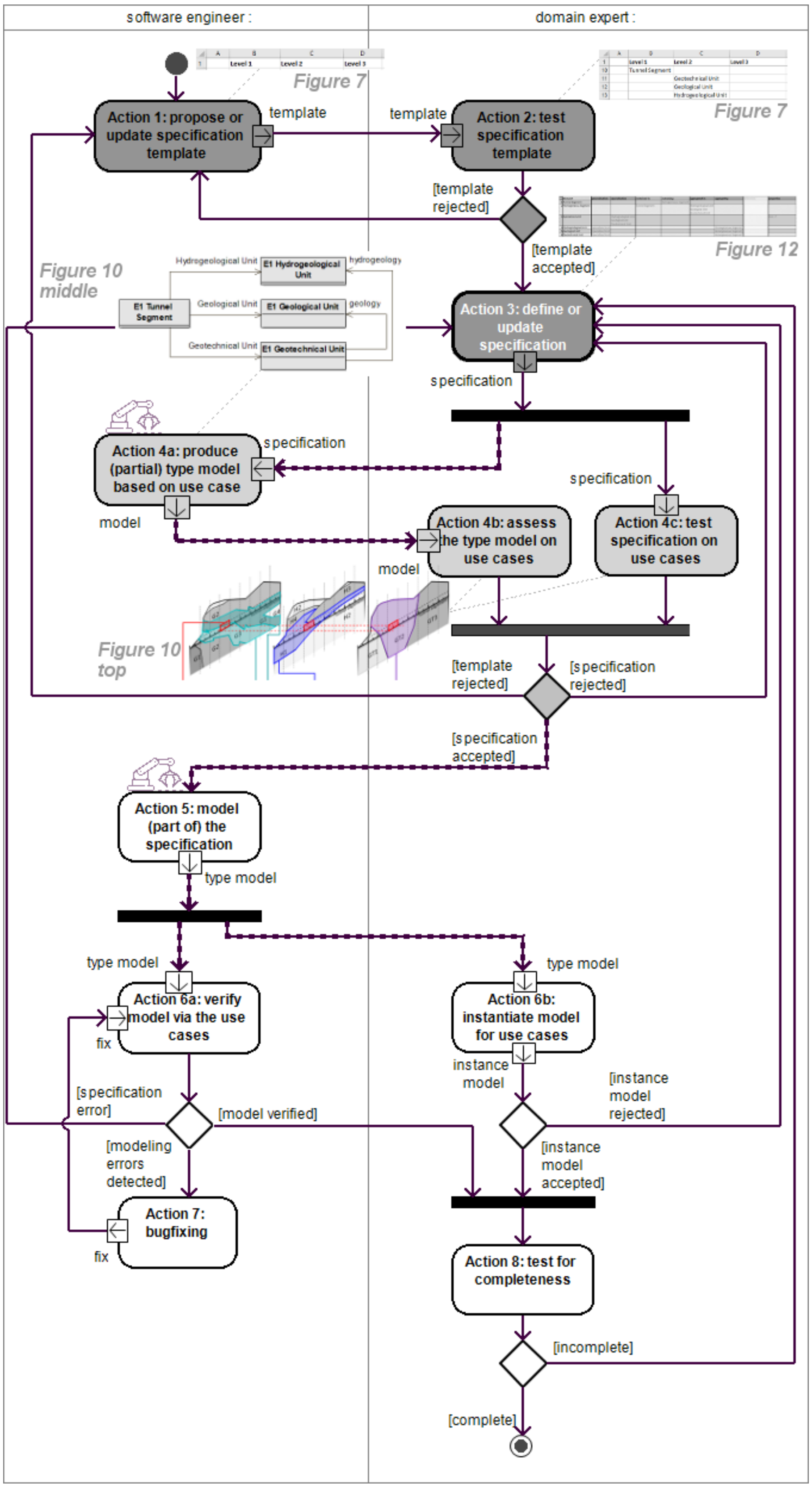
These steps and the automated translation between modeling languages encourage extensive communication between the domain expert and software engineer at the very start of the project, which allows problems and misunderstandings to be identified at a time when their correction costs the least amount of effort (Cohn, 2004). In Fig. 9 this is expressed in the activity flows encompassing all actions from *Action 1* to the transition to *Action 5*. Here, both the software engineers and the domain experts have the opportunity to test different aspects of the specification via a preliminary software model, each in their preferred language. If all tests are successful, we can regard the requirement specification as complete and can proceed with *Action 5*, which includes the production of a (higher resolution) software model. Otherwise, we return to *Action 3* for another iteration of specification refinements, or even to *Acton 1* for adaptation of the specification template. This adaptation of the template is effectively a refinement of the translation process, in our case, between a spreadsheet and UML. It is of note that *Action 3: define or update specification*, and therefore *Acton 1* as well, can be revisited multiple times until the detailed software model is complete, similar to the process described in (Cohn, 2004).

3.1 Application to the Motivating Example

In this section we visit each action depicted in the diagram in Fig. 9 as it applies to our motivating example.

Action 1: propose or update specification template. The software engineers create a spreadsheet template for the specification that could, for example, allow for a listing of data elements as shown in the table excerpts in Fig. 7. This template provides a structuring mechanism through the use of *levels*. However, there is an inherent ambiguity at those levels. Is an element on Level 1, for example, the "parent", the "container" or the "type" of elements on Level 2?

Action 2: test specification template. The domain experts produce a specification for the data structure using this template. If the software engineers interpret the levels in the template as "containers" they could say that *Tunnel Segment* contains a *Geotechnical Unit*, a *Geological Unit* and a *Hydrogeological Unit*. From the domain experts' perspective, this interpretation is incorrect. In reality, if we take the example of the DAUB recommendation in Fig. 4, the levels should be interpreted as simple associations, in the sense that there is a connection from *Tunnel Segment* to all elements on Level 2.



software engineer :
domain expert :
Figure 7
Action 1: propose or update specification template
template
template
Action 2: test specification template
Figure 7
[template rejected]
[template accepted]
Figure 12
Figure 10 middle
Action 3: define or update specification
specification
Action 4a: produce (partial) type model based on use case
specification
specification
model
Action 4b: assess the type model on use cases
Action 4c: test specification on use cases
model
Figure 10 top
[template rejected]
[specification rejected]
[specification accepted]
Action 5: model (part of) the specification
type model
type model
type model
Action 6a: verify model via the use cases
fix
Action 6b: instantiate model for use cases
instance model
[instance model rejected]
[specification error]
[model verified]
[modeling errors detected]
[instance model accepted]
Action 7: bugfixing
fix
Action 8: test for completeness
[incomplete]
[complete]

FIG. 9: A UML activity diagram depicting the synchronization of software and domain modelling performed by the software engineer and the domain expert. The thicker lines indicate automated translation.

Transition to Action 3. Following *Action 2* we have to decide if the template offers adequate translation support or not. If the intended goal of the domain experts is to work with primarily one type of relationship, the template in Fig. 7 would be sufficient as long as it is used consistently. If, on the other hand, the domain experts need different types of relationships, the template has to either offer a dedicated sheet per type of relationship, or each sheet has to allow for multiple types. An example of the potential confusion is shown in Fig. 8, where two interpretations of the levels are mixed in the same sheet. On the one hand, the *Tunnel Segment* is associated with a *Geotechnical Unit*, on the other hand, the *Geotechnical Unit* contains a *Geological Unit* and a *Hydrogeotechnical Unit*. Only NLP and expensive consistency checks (Elrakaiby et al., 2018) coupled with domain knowledge might be able to detect this.

Action 3: define or update specification. Here the domain experts define the actual specification by applying the template, e.g. filling in the spreadsheet.

Action 4a: produce (partial) type model based on use case. After the first version of the specification has been produced by the domain experts, the software engineers can create a first (partial) software model. In our case, this happens as an automated translation via our tool *Excel2UML*, i.e. without the involvement of software engineers. At this stage, their role can be reduced to reviewing the resulting UML model.

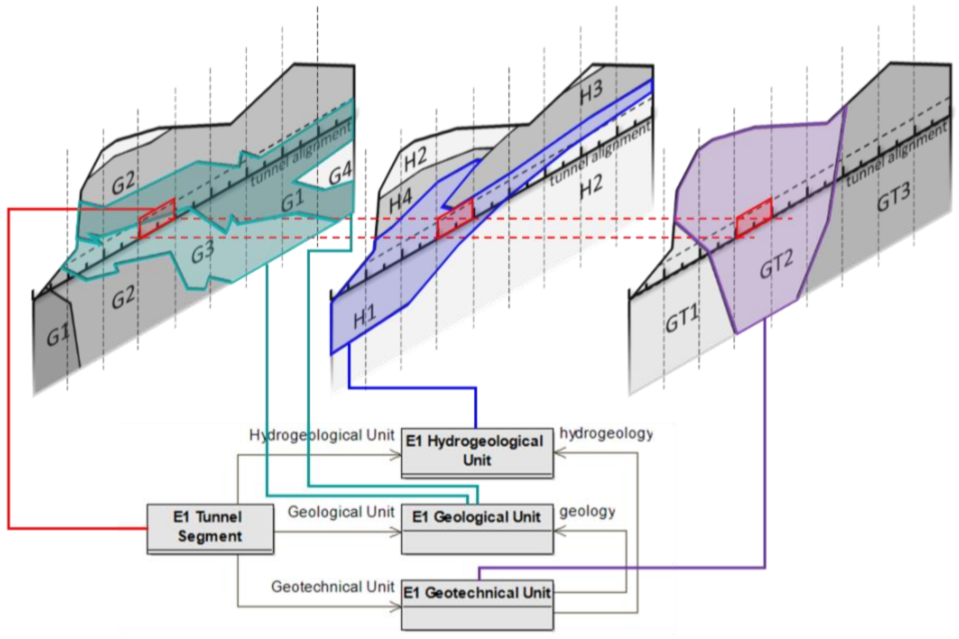
Action 4b: assess the type model on use cases. This model can be simultaneously assessed by the domain experts themselves as the graphical representation of the data structure makes it easier to read, comprehend and discuss (Tallgren et al., 2020). In addition, our tool allows the model to be instantiated here automatically, and the resulting specific instance structures can be compared with the user expectations, thereby providing nearly instant feedback. For example, here the domain engineers can evaluate the types of structural dependencies between geology, hydrogeology, and geotechnics that are achievable by the application of this model on a pre-selected set of use cases.

Action 4c: test specification on use case. A more freeform testing of the specification on these use cases is also suitable at this stage. In the context of our motivating example, this could mean examining the exact perspective on the modelled domain (see Fig. 10, Fig. 11, and Fig. 12). In each figure, the top part shows the subdivision of the subsoil according to, from left to right, the subdomain of geology, hydrogeology, and geotechnics. The tunnel segment between two typical sections is displayed in red. The middle part of the figure shows the corresponding UML model generated from the initial specification using our spreadsheet template, as shown in the bottom part of the figure.

For example, in Fig. 10 we have a data model that allows for arbitrary subdivisions along the tunnel alignment (cf. *Tunnel Segment*) that simply point to the geological, hydrogeological, and geotechnical units relevant for them. There is no hierarchy between the subdomains. In Fig. 11, the segmentation along the tunnel alignment is handled identically. However, there *is* a hierarchy between the subdomains - the geotechnical domain has the dominant perspective and, therefore, forces additional subdivisions of the geological and hydrogeological units. In Fig. 12, the segmentation along the tunnel alignment is dependent on the structures resulting from all three subdomains. It reflects a requirement that each segment is homogeneous in its geology, hydrogeology, and geotechnics. This necessitates a more complex geometric representation of each segment.

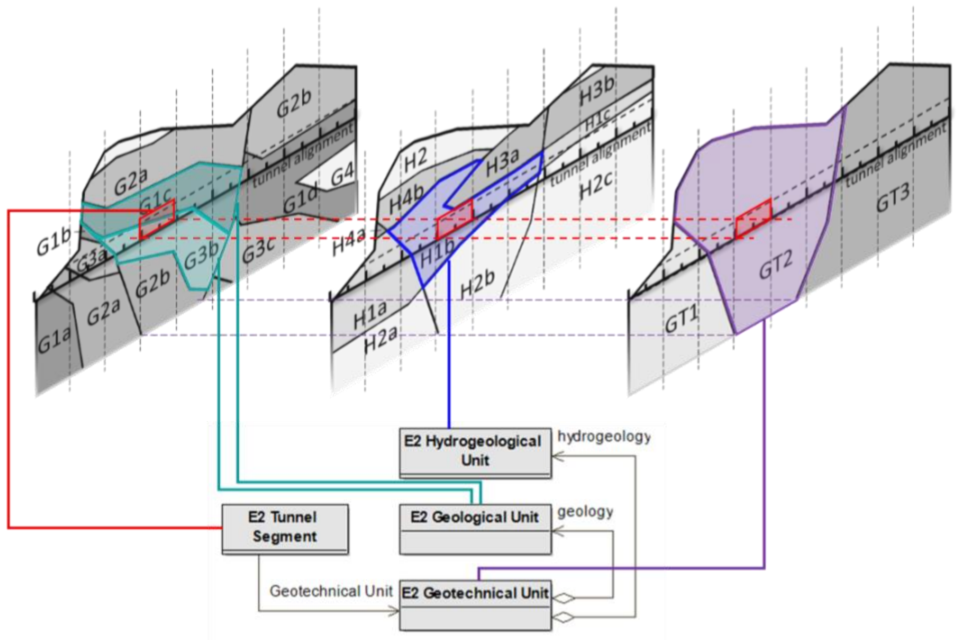
The workflow described by *Action 4b* and *Action 4c* triggers some of the processes responsible for converting information into knowledge (Awad and Ghaziri, 2004, Rowley, 2007): e.g., reflection and synthesis. The subsequent discussion of the instantiated models adds expert opinion and experience to the domain information contained in the filled-in template, thereby eliciting a feeling of ownership and commitment in all involved parties, which has been noted to improve overall performance (Tallgren et al., 2020).

Transition to Action 5. After performing the tests above, another decision has to be made. On the one hand, the template for defining the specification can be rejected even at this stage, if the translation is found to be inadequate. On the other hand, the specification itself can be refined further by cycling through *Action 3*, *Action 4a*, *Action 4b*, and *Action 4c* multiple times. Even if the specification is accepted and the workflow proceeds to *Action 5*, we can still return to *Action 3* at a later stage, e.g. after *Action 6b* or *Action 8*. It is of note that the activity diagram in Fig. 9 covers the process depicted in Fig. 3 only up to *design (high-resolution model)*, i.e. the grey part of the chart. Only after we have completed the process in Fig. 9 will the production of platform-dependent code fully commence.



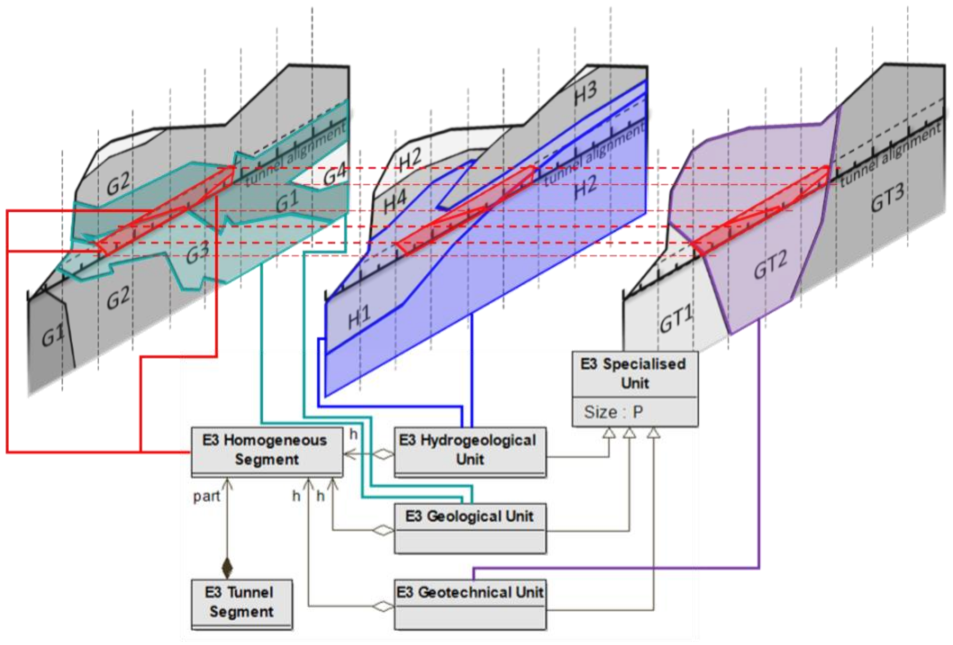
ID	Element	generalisation	specialisation	contained in	containing	aggregated in	aggregating	references	properties
1	Tunnel Segment							Hydrogeological Unit Geological Unit Geotechnical Unit	
2	Hydrogeological Unit								
3	Geological Unit								
4	Geotechnical Unit							Hydrogeological Unit Geological Unit	

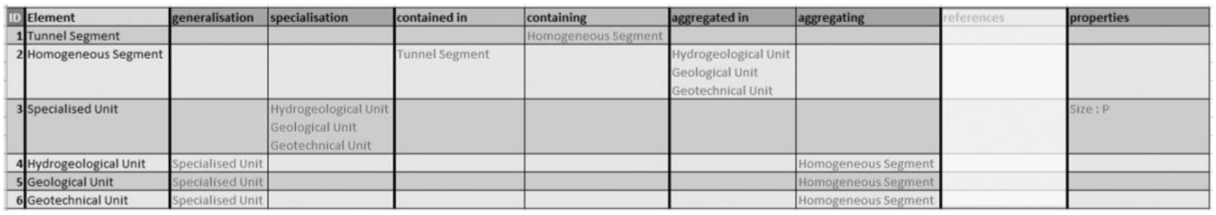
FIG. 10: Data structure design including only references.



ID	Element	generalisation	specialisation	contained in	containing	aggregated in	aggregating	references	properties
1	Tunnel Segment							Geotechnical Unit	
2	Hydrogeological Unit					Geotechnical Unit			
3	Geological Unit					Geotechnical Unit			
4	Geotechnical Unit						Hydrogeological Unit Geological Unit		

FIG. 11: Data structure design including aggregation.





ID	Element	generalisation	specialisation	contained in	containing	aggregated in	aggregating	references	properties
1	Tunnel Segment				Homogeneous Segment				
2	Homogeneous Segment			Tunnel Segment		Hydrogeological Unit Geological Unit Geotechnical Unit			
3	Specialised Unit		Hydrogeological Unit Geological Unit Geotechnical Unit						Size : P
4	Hydrogeological Unit	Specialised Unit					Homogeneous Segment		
5	Geological Unit	Specialised Unit					Homogeneous Segment		
6	Geotechnical Unit	Specialised Unit					Homogeneous Segment		

FIG. 12: Data structure design allowing more complex relationships and geometry.

Action 5: model (part of) specification. Here, the formally defined specification is automatically translated into a type model, i.e. a model defining the types of objects we can work with – for example, *Tunnel Segment*, *Geotechnical Unit*, *Geological Unit*, and *Hydrogeological Unit* (cf. Fig. 12). This is where the software model takes shape and, through multiple iterations, develops into the full high-resolution software model.

Action 6a: verify model via use cases. The (intermediate) type model has to be verified (see definition 3.1.26 in section 2.2.1). When it is produced separately from the specification, this is an indispensable step, as it is the only method for guaranteeing that the model conforms to the specification. In our case, since the model is created automatically from the specification, which is produced directly by the domain expert, this conformity relationship is realized by design through the automated translation.

Action 6b: instantiate model for use cases. The (intermediate) type model has to be validated as well (see definition 3.1.25 in section 2.2.1), i.e. it has to satisfy all use cases that stand at the very beginning of the process. Since, in our approach, the type model is instantiated automatically it produces (empty) instance models. These can be enriched with specific information by the domain experts and directly compared to the relevant use cases to make sure that the instance models and, therefore, the specification itself is fit for the purpose defined at the start of the process. For this reason, after this action, there is another decision to be made, which can result in a return to *Action 3* for specification refinement.

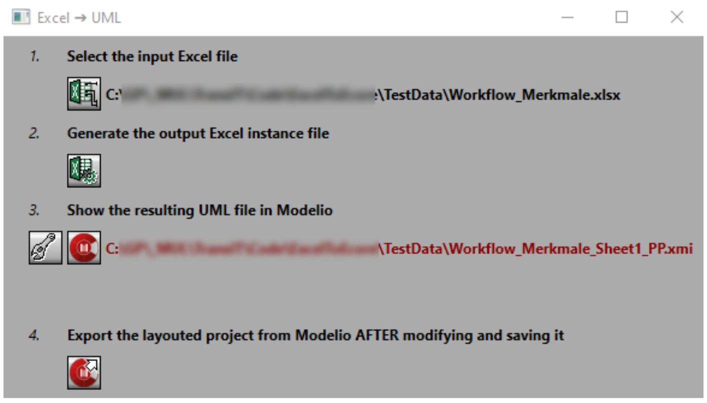
Action 7: bugfixing. This is an action typical for manual model development. In our case, through the automatic creation of the type model, any errors (or bugs) are likely to result from problems in the specification, which is why we can return to *Action 3* even after *Action 6a* for another iteration of specification refinement.

Action 8: test for completeness. Here the domain experts have one last opportunity for evaluating the models, and therefore the specification, which in our case, produces them automatically. Even after this action, we can still jump back to *Action 3* and refine the specification. (Uddagiri et al., 2020) recommend using a multi-dimensional viewpoint model for completeness checks. In our case, viewing the data model separately from the perspective of *geology*, *hydrogeology*, and *geotechnics* could prove useful.

In summary, this workflow enables a full interleaving of software requirement engineering and software development in the early stages, when adaptability is high and the cost of revisiting decisions is low. In addition, the method for requirement elicitation, e.g., in our case the template, can be adapted even during the early stages of software development to provide the best environment for communication between domain experts and software developers. In essence, we can adapt the translation between the language of the domain experts and the language of the software engineers to the very end of the SRE process. In the case of our motivating example, it results in the template shown at the bottom of each of Fig. 10, Fig. 11, and Fig. 12. It expands the concept of levels we discussed under *Action 1* and *Action 2* to four different types of relationships: specialization, containment, aggregation, and association. In addition, an element's properties of elementary type, such as text or numbers, are also handled separately. This demonstrates that throughout the workflow shown in Fig. 9 the translation itself undergoes a considerable adaptation.

3.2 The Tool for Automated Translation

Here, we will give a brief description of the *Excel2UML* translation tool we implemented in order to evaluate our approach. It is an open source tool written in C# and can be downloaded from our university website (*https://doi.org/10.48436/v0ng0-xy233*). In the activity diagram depicted in Fig. 9, it is applied along each transition marked by a thicker line.


Excel ➔ UML

1. Select the input Excel file

C:\[redacted]\TestData\Workflow_Merkmale.xlsx

2. Generate the output Excel instance file

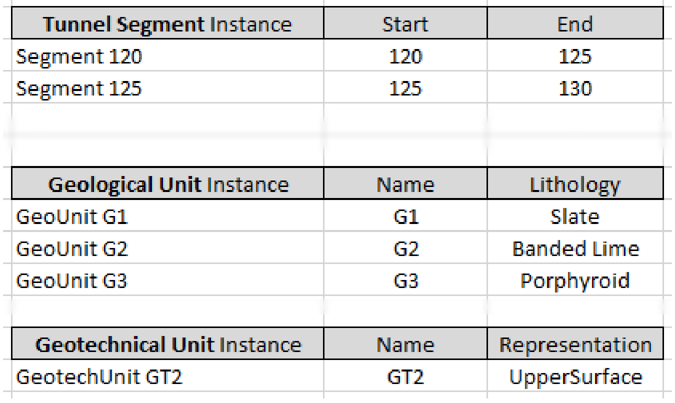
3. Show the resulting UML file in Modelio

C:\[redacted]\TestData\Workflow_Merkmale_Sheet1_PP.xmi

4. Export the layouted project from Modelio AFTER modifying and saving it

FIG. 13: The user interface of the Excel2UML translation tool.

The user interface in Fig. 13 lists the steps required for the translation of a requirement specification in an Excel spreadsheet to a UML model. In **step 1**, the user (e.g., the domain expert) selects the file and sheet containing the filled in template devised by the software engineer (e.g., see the templates shown at the bottom of Fig. 10, Fig. 11, or Fig. 12). This is the basis for the type model (see *Action 4a* in Fig. 9).


Tunnel Segment Instance	Start	End
Segment 120	120	125
Segment 125	125	130

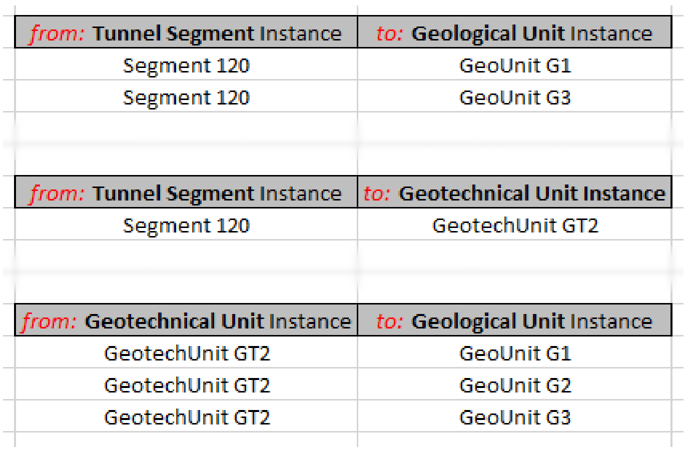
Geological Unit Instance	Name	Lithology
GeoUnit G1	G1	Slate
GeoUnit G2	G2	Banded Lime
GeoUnit G3	G3	Porphyroid

Geotechnical Unit Instance	Name	Representation
GeotechUnit GT2	GT2	UpperSurface

FIG. 14: An excerpt of the filled in instance model corresponding to Fig. 10: Instances.

In **step 2**, the tool generates the type model (see *Action 4a* or *Action 5* in Fig. 9), and directly instantiates it (see *Action 4b* or *6b* in Fig. 9) to enable the testing of use cases. Excerpts of those instantiated models are shown in Fig. 14 and Fig. 15. For example, from the type *Tunnel Segment* the tool produces the header of the first table in

Fig. 14, where the domain expert can fill in some specific tunnel segments, such as *Segment 120* and *Segment 125*. One such segment (*Segment 120*) is highlighted in red in the top portion of Fig. 10. The other table headers in Fig. 14 provide the opportunity for defining some instances of *Geological Unit*, e.g., *GeoUnit G1*, *GeoUnit G2*, and *GeoUnit G3*, as well as some instances of *Geotechnical Unit*, e.g., *GeotechUnit G2*. All of these are shown in the longitudinal sections in Fig. 10. The definition and instantiation of attributes, such as *Name* and *Lithology* for the *Geological Unit*, were omitted here for brevity.



from: **Tunnel Segment** Instance	*to:* **Geological Unit** Instance
Segment 120	GeoUnit G1
Segment 120	GeoUnit G3

from: **Tunnel Segment** Instance	*to:* **Geotechnical Unit Instance**
Segment 120	GeotechUnit GT2

from: **Geotechnical Unit** Instance	*to:* **Geological Unit** Instance
GeotechUnit GT2	GeoUnit G1
GeotechUnit GT2	GeoUnit G2
GeotechUnit GT2	GeoUnit G3

FIG. 15: An excerpt of the filled in instance model corresponding to Fig. 10: Relationships.

In addition to instantiating types, **step 2** instantiates the relationships between them (see Fig. 15). For example, the type model in the centre of Fig. 10 shows an association between *Tunnel Segment* and *Geological Unit*. This relationship is instantiated as the first table header in Fig. 15. Following this, the domain expert can fill in the fact, that, in the specific use case in Fig. 10, *Segment 120* actually overlaps with *GeoUnit G1* and *GeoUnit G3*, i.e. is associated with them. In the same manner, the associations between *Tunnel Segment* and *Geotechnical Unit* and between *Geotechnical Unit* and *Geological Unit* are instantiated in the next two table headers in Fig. 15. In essence, this step allows for extensive use case testing. Missing or superfluous tables indicate problems in the type model itself.

In **step 3**, the type model can be visualised as a UML class diagram in an open source modeling software, Modelio (https://www.modelio.org/), for inspection both by the software engineers and the domain experts. Finally, in **step 4**, an adapted UML diagram can be exported for further processing by the software engineers.

3.3 The Relationships

In this section, we will discuss the types of relationships between model elements that emerged through the application of our approach and the advantages as well as disadvantages associated with them. In essence, they are the most commonly used relationships in a UML class diagram.

- **Generalization**. Element A is the *generalization* of element B if element B inherits all properties and relationships of element A. Element B can have additional properties and relationships, or restrictions on existing ones, that make it a more specialized version of element A. Therefore, it can be described as the *specialization* of A;
- **Containment**. Element B makes sense and can exist only within Element A, i.e. it is *contained* in Element A. Therefore, each instance B_i of Element B belongs to exactly one instance A_j of Element A. Should instance A_j cease to exist, so does instance B_i;
- **Aggregation**. Element B makes sense and can exist on its own or within Element A. Therefore, each instance B_i of Element B can belong to zero, one, or multiple instances A_j of Element A. The continued existence or deletion of instance A_j does not affect the existence of instance B_i. In this case, Element A *aggregates* Element B;
- **Unidirectional Reference**. Element B knows Element A if Element B *references* Element A. The relationship can have any multiplicity, i.e. any number of instances B_i of Element B can reference any number of instances A_j of Element A.

Requiring of the domain experts to consider four types of relationships makes at least the initial phases of the template evaluation and application more challenging. However, as we demonstrated with our example here, the judicious application of these relationships has the potential to encourage critical thinking and creativity as defined by (Facione, 1990) and emerges naturally in the specification process. As we can see, it can give rise to different data model designs, each with its own focus. The three designs depicted in Fig. 10, Fig. 11, and Fig. 12 demonstrate this versatility. Comparing the filled templates at the bottom of each figure reveals that different data structures stem from very different specifications since each type of relationship occupies its own dedicated column and performs a different structuring role. The additional clarity that comes with the formalization of relationships between elements enables the domain experts to communicate to the software engineering team not just the domain's *semantics*, but also its *pragmatics*, since, "*meaning is at the heart of both semantics and pragmatics*" (Fetzer, 2004).

4. RELATED WORK

In this section we present existing approaches to the software development phases *user expectations*, *technologies*, *requirement specification*, *structure (low resolution model)*, and *design (high resolution model)* depicted in Fig. 3, and compare them to our work. Some are partial, other complete solutions.

4.1 Requirement Elicitation

In the AEC industry, there have been multiple strategies for obtaining requirements for CIC software. For example, (Arayici et al., 2006) utilize the Contextual Design method, which allows the inclusion of typical domain workflows and interfaces in their proper work context in the requirement elicitation process. Similarly to our approach, they rely on incremental prototyping followed by end-user tests. They propose a methodology for the development of a SRE framework for CIC systems and identify 44 key issues for its evaluation. One of them is the mutual understanding of the stakeholders' perspectives, something that is indispensable in our approach as well.

A framework for requirement elicitation from domain experts based on the Methodology for Knowledge-Based Engineering Applications (MOKA) is presented in (Häußler and Borrmann, 2021). MOKA includes a cycle of the following steps: "identify", "justify", "capture", "formalize", "package", and "activate" with the aim to minimize the communication barriers between domain experts and software engineers. It has been used in aerospace, mechanical engineering, and manufacturing. The authors give an example of the application of MOKA to the domain of railway infrastructure. In the "capture" phase they obtain expert knowledge by means of interviews documented in ICARE (Illustrations, Constraints, Activities, Rules, and Entities) forms, including the performance of a task and its explanation. The authors use UML diagrams in the "formalize" step, which when using the fUML (https://www.omg.org/spec/FUML/1.5/About-FUML/) supported subset, could be actually executed. Those diagrams are created not by the domain experts, but manually, by the software engineers, due to the lack of uniform notation in MOKA, which the authors identify as a communication disadvantage. Finally, the result of the entire process was expressed as formal diagrams created by the software engineers in collaboration with the domain experts. In comparison, in our work, we give many opportunities for collaboration but remove the necessity for the software engineers to manually translate the domain experts' input, in order to avoid the above mentioned disadvantage.

In (Zahedi et al., 2022), the authors present a methodology for eliciting decision-making information from designers. Explanation tags and constraints, based on a dedicated metamodel, are attached to the BIM model elements to provide a decision tracking aid. Links to international guidelines fulfil a similar role. The designers' intent is communicated via text, not in their typical language of sketching, drawing, or painting, which might be a hindrance in effective communication. Furthermore, the applied explanation tags require an organization in a full taxonomy, which should ideally be developed by the designers themselves. In our approach, we allow the domain experts to use at least one of their typical tools and don't expect the emergence of a full taxonomy until the very end of the workflow depicted in Fig. 9.

Another methodology for gathering requirements from various stakeholders in the AEC industry by conducting formal interviews is presented in (Ye et al., 2009). The authors state that the interviews offered much more flexibility than online questionnaires. The next step involved the utilization of the Hamburger Model by Gielingh, essentially locating the "functional concept" and the "solution concept" in the NL requirements, followed by attaching a relative importance index. The verification was done by hand and the results were grouped in the following requirement themes: "energy management", "comfort", "life cycle costing", "customer-orientation", "flexibility", and "building process". Similarly, in the templates that start the SRE process in our approach, and gradually evolve as the workflow progresses, we give the domain experts the opportunity to add their own categorization, in addition to the various relationships that emerge. For example, some users chose to use some of the categorization proposed by (German Tunnelling Committee (ITA-AITES), 2022a), such as "Object" and "Partial Object", other chose domain-specific ones, e.g., "energy efficiency" or "risk".

In section 2.3, we pointed out the difficulty in adoption of new tools in the AEC industry, at least in part, due to them requiring a change in well-established workflows. In (Tallgren et al., 2020), the authors present an approach to gathering workflow information for a new collaborative planning tool by combining observations, field notes and interviews. During prototyping, there were multiple evaluation phases, in which the users were recorded interacting with the software and with their colleagues. This level of collaboration was also one of the main principles we followed in designing both the workflow and the tool that supports it.

4.2 Formal Requirement Extraction

As we established in section 2.2.4, NL requirements are quite common. (Dalpiaz and Brinkkemper, 2018) show that 90% of agile developers fall back on user stories, 70% of which adhere to quite simple templates containing only a few placeholders, e.g., for "role", "action", or "benefit". What’s more, in the AEC industry, there is a push to include many guidelines and norms, written in NL, into the set of requirements of a CIC software. Consequently, there are multiple methods for translating such texts into a formal language for easier processing. However, neither the Requirement, Applies, Select, Exception (RASE) template nor Natural Language Processing (NLP) produce reliable results (Häußler et al., 2020, Zahedi et al., 2022).

One of the reasons is that guidelines and norms often describe complex and multi-factorial decision processes that exceed the capabilities of simple templates, such as RASE, which are hindered by the lack of higher order predicates, and consequently, cannot encode experience or procedural knowledge (Preidel and Borrmann, 2016). Therefore, in (Häußler et al., 2020), the authors show a method for automatic classification of the rules contained in some of the guidelines of the Deutsche Bahn AG into classes and their subsequent translation into executable Business Process Modeling Notation (BPMN) models. They report a success rate of 52% of all rule sets for twelve rule classes. An important feature of the BPMN models is the inclusion of scripts for various routines into the model elements. Furthermore, Decision Model Notation (DMN) allows the integration of decision tables that go beyond the "if then else" mechanism of BPMN. The encoding of guidelines as BPMN diagrams is a field we intend to explore in our future work by devising a method for translating spreadsheet requirements into BPMN diagrams.

Another domain where the translation of guidelines into formal requirements is absolutely critical is avionics. This a domain that relies heavily on a complex body of knowledge organized in multiple taxonomies. However, taxonomies can be overused and contribute to confusion rather than to clarity. For example, as Jaffe points out in (Jaffe, 2021), the taxonomy of requirement types includes: “*functional, performance, high level, low level, lower level, derived, interface, design, operational, system, system operational, safety-related, security, initial, user, and detailed requirements*”, with no clear distinction between the types. This ambiguity of terms may result from the need for consensus, so that more general terms are adopted in place of more precise ones, or from the accommodation of legacy documents. Avoiding confusion is one of the reasons we restrict the number of formal relationships in a spreadsheet template to a minimum in our approach. However, in order to allow the domain experts the freedom to use a less than well-defined taxonomy, they themselves consider essential, we provide them with the additional categorization fields we mentioned in the previous section.

It is to be noted that our approach was evaluated on a use case of extracting a suitable data structure for a particular domain. In more complex scenarios with multiple conflicting viewpoints or the involvement of legacy technologies, requirements may take various forms, including pieces of code or even sentiment (Werner et al., 2019). In essence, freeform requirements cannot be entirely avoided. The following publications demonstrate various strategies for formalizing those.

According to (Dalpiaz et al., 2018), important checks to consider when dealing with multiple viewpoints include consistency within one viewpoint and consistency between viewpoints. The main utility of viewpoints is to make differences in terminology explicit, which can manifest as "consensus", "correspondence", "conflict", or "contrast". This differentiation helps with the detection of ambiguity.

A formalization method based on examples is presented in (Bragilovski et al., 2022). The authors propose example-based guidelines for the derivation of formal requirements out of user stories. The results from the controlled experiment suggest that those are only partially useful, e.g. for more complex domains.

An approach that bypasses user involvement, at least in the early stages, is reverse engineering - specifically application and transaction logs in complex middleware systems, as demonstrated in (Uddagiri et al., 2020). This can provide a good overview of functional requirements and pinpoint deficits in the requirement specification. In our case, however, since the requirements of data models are not functional, we cannot make use of such techniques.

4.3 Visualization Methods

Multiple works indicate that visualizations during the SRE process enhance both the mutual understanding and the quality of communication, which is one of the reasons we chose to translate the spreadsheet requirements into easy to visualize UML class diagrams, or graphs.

(Kolbeck et al., 2022) outline a method for representing the structure of a product as a graph whose nodes represent entities and whose edges represent the relationships between them, i.e., quite similar to the data structures we captured in our motivating example. Based on such graphs, the engineering workflow, including decision-making, can be formulated as graph transformation rules. This includes (i) the representation of the problem as a graph, (ii) the generation of solutions as graph transformations, (iii) the evaluation of these solutions via, e.g., graph matching, and (iv) guidance for the next step in the search for solutions.

Furthermore, requirements in the form of graphs can be subject to formal reasoning, as presented in (Elrakaiby et al., 2018). The authors propose a formal calculus for discussions about the correctness, completeness or consistency of a requirement set, represented by graphs, between end users and software engineers, which results is a refinement graph.

Other approaches make use of visual programming. In (Preidel and Borrmann, 2016) the authors introduce the Visual Code Checking Language (VCCL) as a means of representing a guideline rule visually. It has a hierarchical graph structure, which at the lowest level consists of method nodes with input and output ports. In (Preidel et al., 2017) the authors present a graph-based querying mechanism for rules represented as graphs and, after evaluation, conclude that formal textual query languages perform worse with domain experts than formal graphical query languages, such as vQL4BIM and the aforementioned VCCL, due to the users' lack of programming knowledge. This is a promising development. It demonstrates that both structural as well as functional requirements can be represented as graphs, which allows for the application of similar methods in their elicitation and processing, something we intend to explore in our future work.

4.4 Relationships in Data Models

In section 3.3, we presented our motivation for restricting ourselves to only four different relationship types in our template. However, BIM related standards actually tend to have many more. Here we will give a few examples.

As one of the most comprehensive open BIM standards, the Industry Foundation Classes (IFC) (buildingSMART, 2022) defines a multitude of relationships, including assignment to specific types (e.g. *IfcRelAssignsToProduct*), association with specific types (e.g. *IfcRelAssociatesMaterial*), connection of various kinds (e.g. *IfcRelConnectsStructuralMember*), declarations (e.g. *IfcRelDeclares*), decompositions (e.g. *IfcRelDecomposes* or its subtype *IfcRelNests*), and definition (e.g. *IfcRelDefinesByType* or *IfcRelDefinesByProperties*).

IFC is first and foremost a multi-domain data model for the AEC industries. Therefore, it contains many domain-specific elements such as *IfcCableFitting* for the electrical domain. In addition to that however, it contains elements that allow a very generic approach to domain modeling, akin to a metamodel (Brambilla et al., 2017). This is where the above listed relationship types contribute to the differentiation of elements, even if the elements themselves are as generic as, for example, *IfcProcess*. This mechanism is meant to supplement the already existing domain-specific model, since state-of-the-art in the AEC industries moves faster than the implementation cycle of IFC, and domain expert-driven additions become necessary.

Another example is the DIN EN 17632 guideline (CEN/TC-442, 2021). It outlines generic methods for data exchange and integration in BIM, which involve the definition of multiple relationships, e.g., for locations in space and time alone: *hasBoundary*, *hasInterior*, *hasPart*, *hasPeriod*, *begins*, *ends*, *triggers*, *hasState*, *transforms*, etc. A related guideline, for document exchange, the DIN EN ISO 21597-2 (ISO/TC 59 and CEN/TC 442, 2020), offers the following types of relationships: *IsSpecialisedAs*, *HasPart*, *HasMember*, *Supersedes*, *IsElaboratedBy*, *Controls*, *IsIdenitcalTo*, *ConflictsWith*, and *IsAlternativeTo*, among others.

It is to be noted that this great diversity of relationships is justified in a multi-domain industry as means of communication. However, in a tool for requirements specification, the relationship complexity may obscure the domain semantics and make errors difficult to locate (Preidel et al., 2017, Tallgren et al., 2020). In fact, the relationships listed above could be regarded as specializations of the UML-typical relationships we use in our approach. For example, *hasBoundary* and *hasPeriod* are in essence subtypes of Aggregation; *IfcRelDecomposes*, *hasPart*, or *HasMember* - subtypes of Containment, etc. Therefore, it is possible for a requirement refinement step to include even the refinement of the relationships offered by the template. This would indeed allow the physical world and digital models representing (parts of) it to be coupled to an ever increasing degree of closeness (Christiansson et al., 2009), which is one of the major advantages of software modeling, and facilitate a more gradual and controlled SRE process.

5. DISCUSSION

Let us now return to section 2.3 and to the five semantic and five pragmatic aspects of NL requirements we wanted to address in our approach.

- **(F1) Ambiguity**. We showed that even a template that defines only levels can enforce consistency in the presence of a consensus about the meaning of "level". Our template goes one step further and differentiates between four different types of relationships: specialization, containment, aggregation and association. We excluded the "type of" relationship as it is reserved for the instantiation of the type model into an instance model, or in the context of a software, for the instantiation of a class into an object. The reduction of ambiguity greatly contributes to effective translation between the languages of different experts;
- **(F2) Technical jargon**. As the purpose of developing a data model for a highly specialized domain is to organize and structure technical jargon, this is not something we aim to achieve. In order to reduce the usage of software engineering jargon, we restricted our template, at least initially, to only four types of relationships;
- **(F3) Lack of conceptual soundness**. The utilization of an appropriate template makes requirements of, e.g., the user interface quite easy to distinguish from requirements of the data model simply because those cannot be properly connected - no appropriate relationship, e.g., *represents*, is available to the domain expert. In general, if there are (nearly) disjunct sets of elements in the UML model, that are not separated by design (as might be the case with hydrogeology and geochemistry), it is an indication that there may be conceptual "cross-contamination" from other domains. It is also of note, that our workflow depicted in Fig. 9 explicitly allows the refinement of the template, thereby improving the contextual soundness of the resulting specification;
- **(F4) Overlaps and contradictions.** Since our approach generates a UML model from the spreadsheet specification automatically, duplicates and conflicts lead to invalid models. This allows us to recognize and correct such problems without delay;
- **(F5) Non-problem-oriented statements.** This is handled by the automatic translation of the filled-in template into a UML model. By analyzing the graphical appearance of this model, it becomes apparent if there are elements that are largely disconnected from the rest, or elements that are connected to nearly all other elements. Both are signs of poor design. The first case could prompt a discussion about the necessity of including the disconnected element in the model at all. The second case could indicate that, for example, *generalization* has not been utilized properly;
- **(F6) Validatability**. In our case, in the context of a fairly specialized domain, the fulfillment of requirements can be measured by automatically instantiating the data model and attempting to manually represent the chosen use cases;
- **(F7) Uniformity**. This aspect is an integral part of any uniform template by design;

- **(F8) Uniqueness**. Even if there is a semantic duplication through, e.g., a spelling error, the UML model should display this as two elements that have identical connectivity, which should trigger a re-examination of the model;
- **(F9) Explicit Dependencies**. This is where the differentiation of the relationships admissible in the template plays an important role. There is a balance to be found between connecting everything to everything, just to be "on the safe side", which robs the connections of their meaning, and barely using any relationships for fear of too much flexibility and mixing up "fact" and "opinion". This is a significant challenge, which brings the domain *knowledge* of the involved parties to the front;
- **(F10) Completeness**. In section 4, we mentioned the concept of viewpoints as a tool to aid completeness (Dalpiaz et al., 2018). In our motivating example, we can adopt the different aspects, *geology*, *hydrogeology*, *geotechnics*, and others, as viewpoints. An additional tagging system in the template allows each element to be tagged as belonging to one or more aspect, or viewpoint. In this way, each viewpoint can be evaluated separately and in conjunction with the others, which still does not guarantee completeness, but brings us closer to it.

In addition to the aspects enumerated above, our approach provides the following benefits to the participants in the workflow:

- (B1) Our approach can be utilized as a learning tool. The domain experts can learn data modelling on a familiar platform and, if they so choose, transition to pure UML modelling or to a hybrid approach;
- (B2) The automatically generated graphical representation of the data structures makes them easier to read, comprehend and discuss;
- (B3) Finally, the automation inherent to our approach provides the domain experts with automatically generated spreadsheets ready for the definition of specific instances of the data types they have defined. As the distinction between a type and its instances is not always easy to make or communicate, but is absolutely essential to the software development process, this has the potential to uncover fundamental misunderstandings as early in the SRE process as possible.

6. CONCLUSION

In this paper, we present an approach to software requirement engineering which allows it to be extended well into the code production phase of a software solution for the AEC industry, while at the same time encouraging engagement from both the domain experts and the software engineers involved in the project. While there are many frameworks that pursue the same goal, they can easily become too complicated or too rigid to use comfortably, especially by domain experts with limited programming knowledge. Previous research shows that personal conversations, e.g., via interviews, deliver some of the best results, in no small part due to the increase in motivation and engagement of all participants.

Our approach involves an Excel sheet template which allows the geotechnics domain expert to define model elements connected by as many or as few relationships as needed for a particular task. It also enables model elements to be tagged as belonging to one or more viewpoints in the familiar Excel environment. Through the automatic translation of the template content into a UML type model and a UML instance model, this approach gives instant feedback on the data structure to both the domain experts and the software engineers, and it also provides ready-to-fill-out templates for model instances in order to test the model on use cases.

The workflow we described allows multiple iterations of template and specification refinement with the aim to adapt the translation between the languages of different domains and provide as much accuracy and as much abstraction, as deemed appropriate by the participants in the communication. In addition, by using fUML and automated code generators, the approach enables an automated prototype production and testing. The process is designed with balance between formalization and conversation in person in mind, since it is the human interaction that unlocks the domain knowledge and amplifies its influence on the final software specification.

REFERENCES

Arayici Y., Ahmed V. and Aouad G. (2006). A requirements engineering framework for integrated systems development for the construction industry, *ITcon*, Vol. 11, 35-55.

Awad E. M. and Ghaziri H. M. (2004). Knowledge management, Pearson Education International, Upper Saddle River, NJ, USA.

Barbosa F., Woetzel J., Sridhar M., Parsons M., Bertram N., Brown S., Mischke J. and Ribeirinho M. (2017). Reinventing construction: a route to higher productivity, *https://www.mckinsey.com/business-functions/operations/our-insights/reinventing-construction-through-a-productivity-revolution*, last accessed 17-June-2022.

Berry D. M. and Kamsties E. (2004). Ambiguity in requirements specification, *Perspectives on software requirements* (do Prado Leite J. C. S. and Doorn J. H., editors), Springer US, Boston, MA, USA, 7-44.

Boehm B. (1987). Improving software productivity, *Computer,* Vol. 20, No. 9, 43-57.

Boehm B. W. (2001). Software engineering economics, *Pioneers and their contributions to software engineering: sd&m conference on software pioneers, bonn, june 28/29, 2001, original historic contributions*, Springer Berlin Heidelberg, Berlin, Heidelberg, Germany, 99-150.

Borrmann A., Esser S., Jaud Š., König M. and Liebich T. (2020). Begleitung der internationalen standardisierungsprojekte IFC-Road & IFC-Rail: abschlussbericht gesamtprojekt, Technische Universität München.

Bragilovski M., Dalpiaz F. and Sturm A. (2022). Guided derivation of conceptual models from user stories: a controlled experiment, *Requirements engineering: foundation for software quality* (Gervasi V. and Vogelsang A., editors), Springer International Publishing, Basel, Switzerland, 131-147.

Brambilla M., Cabot J. and Wimmer M. (2017). Model-driven software engineering in practice, Morgan & Claypool, 822 College Ave #457, Kentfield, CA 94914, USA.

buildingSMART (2022). IFC4.3.x, *http://ifc43-docs.standards.buildingsmart.org/*, last accessed 21-October-2022.

CEN/TC 442 (2021). DIN EN 17632: 2021 building information modelling (BIM) - semantic modelling and linking (SML) - draft, *https://www.en-standard.eu/din-en-17632-semantischer-modellierungs-und-verknupfungsstandard-smls-fur-die-datenintegration-in-der-gebauten-umwelt-deutsche-und-englische-fassung-pren-17632-2021/*, last accessed 06-December-2022.

Christiansson P., Svidt K. and Sørensen B. (2009). Future integrated design environments, *ITcon, special issue next generation construction IT: technology foresight, future studies, roadmapping, and scenario planning*, Vol. 14, 445-460.

Cohn M. (2004). An overview, *User stories applied for agile software development*, Addison-Wesley, Boston, USA, 4-16.

Combemale B., France R., Jézéquel J., Rumpe B., Steel J. and Vojtisek D. (2017). What's a model?, *Engineering modeling languages: turning domain knowledge into tools*, Taylor & Francis Group, LLC, Boca Raton, FL 33487-2742, USA, 2-17.

Dalpiaz F., van der Schalk I. and Lucassen G. (2018). Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP, *Requirements engineering: foundation for software quality* (Kamsties E., Horkoff J. and Dalpiaz F., editors), Springer International Publishing, Basel, Switzerland, 119-135.

Dalpiaz F. and Brinkkemper S. (2018b). Agile requirements engineering with user stories, *2018 IEEE 26th international requirements engineering conference (RE)*, 506-507.

David A., Leeb M. and Bednar T. (2017). Comparison of the planned and the real energy consumption of the world's first (plus-)plus-energy office high-rise building, *Energy procedia*, Vol. 132, 543-548.

Dietsch D., Langenfeld V. and Westphal B. (2020). Formal requirements in an informal world, *2020 IEEE workshop on formal requirements (FORMREQ)*, 14-20.

Elrakaiby Y., Ferrari A. and Mylopoulos J. (2018). CaRE: a refinement calculus for requirements engineering based on argumentation semantics, *2018 IEEE 26th international requirements engineering conference (RE)*, 364-369.

Facione P. (1990). Critical thinking: a statement of expert consensus for purposes of educational assessment and instruction (the delphi report).

Fetzer A. (2004). Recontextualizing context, *English and american studies in german*, Vol. 2004, No. 2005, 16-17.

Flewelling P. (2018). Gathering agile user requirements, *The agile developer's handbook*, Packt Publishing, 35 Livery Place, Livery Street, Birmingham, England.

German Tunnelling Committee (ITA-AITES) (2020). Digital design, building and operation of underground structures. BIM in tunnelling model requirements - part 1: object definition, coding and properties. supplement to DAUB recommendation BIM in tunnelling (2019), *https://www.daub-ita.de/fileadmin/documents/daub/gtcrec5/2020-11_DAUB_BIM_im_Untertagebau_Modellanforderungen_T1_en_Rec.pdf*, last accessed 17-June-2022.

German Tunnelling Committee (ITA-AITES) (2022a). Empfehlung digitales planen, bauen und betreiben von untertagebauten. modellanforderungen - teil 2 informationsmanagement. ergänzung zur DAUB-Empfehlung BIM im untertagebau, *https://www.daub-ita.de/fileadmin/documents/daub/gtcrec5/2022-08_DAUB_BIT_Modellanforderungen_T2_Informationsmanagement_Rec_DE.pdf*, last accessed 18-October-2022.

German Tunnelling Committee (ITA-AITES) (2022b). Empfehlung digitales planen, bauen und betreiben von untertagebauten. modellanforderungen - teil 3 baugrundmodell. ergänzung zur DAUB-Empfehlung BIM im untertagebau, *https://www.daub-ita.de/fileadmin/documents/daub/gtcrec5/2022-08_DAUB_BIT_Modellanforderungen_T3_Baugrundmodell_Rec_DE.pdf*, last accessed 18-October-2022.

Häußler M. and Borrmann A. (2021). Knowledge-based engineering in the context of railway designs by integrating BIM, BPMN, DMN and the methodology for knowledge-based engineering applications (MOKA), *Journal of information technology in construction*, Vol. 26, 193-226.

Häußler M., Esser S. and Borrmann A. (2020). Code compliance checking of railway designs by integrating BIM, BPMN and DMN, *Automation in construction*, Vol. 121, No. 103427, 1-24.

Ilčík M., Musialski P., Auzinger T. and Wimmer M. (2015). Layer-based procedural design of façades, *Comput. graph. forum*, Vol. 34, No. 2, 205-216.

ISO/IEC (2018). ISO/IEC/IEEE international standard - systems and software engineering - life cycle processes - requirements engineering - redline, *ISO/IEC/IEEE 29148:2018(e) - redline*, 1-209.

ISO/TC 59 and CEN/TC 442 (2020). DIN EN ISO 21597-2: 2020 information container for linked document delivery - exchange specification - part 2: link types, *https://www.iso.org/standard/74390.html*, last accessed 06-December-2022.

Jaffe M. (2021). Levels of requirements, robustness, unicorns, and other semi-mythical creatures in the requirements engineering bestiary: why "types" of software requirements are often misleading, *2021 IEEE/AIAA 40th digital avionics systems conference (DASC)*, 1-8.

Kühne T. (2006). Matters of (meta-) modeling, *Software & systems modeling*, Vol. 5, No. 4, 369-385.

Kaewunruen S., Rungskunroch P. and Welsh J. (2018). A digital-twin evaluation of net zero energy building for existing buildings, *Sustainability*, Vol. 11, No. 1, 159.

Kolbeck L., Vilgertshofer S., Abualdenien J. and Borrmann A. (2022). Graph rewriting techniques in engineering design, *Frontiers in built environment*, Vol. 7, 1-19.

Lucassen G., Dalpiaz F., van der Werf J. M. E. and Brinkkemper S. (2015). Forging high-quality user stories: towards a discipline for agile requirements, *2015 IEEE 23rd international requirements engineering conference (RE)*, 126-135.

OMG (2022). Unified modeling language, *https://www.uml.org/what-is-uml.htm*, last accessed 17-June-2022.

Osama M., Zaki-Ismail A., Abdelrazek M., Grundy J. and Ibrahim A. (2021). Enhancing NL requirements formalisation using a quality checking model, *2021 IEEE 29th international requirements engineering conference (RE)*, 448-449.

Panichella S. and Ruiz M. (2020). Requirements-collector: automating requirements specification from elicitation sessions and user feedback, *2020 IEEE 28th international requirements engineering conference (RE)*, 404-407.

Parsanezhad P., Tarandi V. and Lund R. (2016). Formalized requirements management in the briefing and design phase, a pivotal review of literature, *ITcon*, Vol. 21, 272-291.

Preidel C. and Borrmann A. (2016). Towards code compliance checking on the basis of a visual programming language, *ITcon*, Vol. 21, 402-421.

Preidel C., Daum S. and Borrmann A. (2017). Data retrieval from building information models based on visual programming, *Visualization in engineering*, Vol. 5, No. 18, 1-14.

Rowley J. (2007). The wisdom hierarchy: representations of the DIKW hierarchy, *Journal of information science*, Vol. 33, No. 2, 163-180.

Royce W. W. (1970). Managing the development of large software systems, *Technical papers of western electronic show and convention*, 1-9.

Saxena S. K. and Chakraborty R. (2014). Decisively: application of quantitative analysis and decision science in agile requirements engineering, *2014 IEEE 22nd international requirements engineering conference (RE)*, 323-324.

Tallgren M. V., Roupé M., Johansson M. and Bosch-Sijtsema P. (2020). BIM-tool development enhancing collaborative scheduling for pre-construction, *ITcon*, Vol. 25, 374-397.

Uddagiri V., Eswarachary L., Jagadeesan M. and Kharat V. (2020). Improving the quality of requirements in middleware requirements specifications, *2020 IEEE 28th international requirements engineering conference (RE)*, 412-415.

Werner C., Li Z. S. and Ernst N. (2019). What can the sentiment of a software requirements specification document tell us?, *2019 IEEE 27th international requirements engineering conference workshops (REW)*, 106-107.

Ye J., Hassan T., Carter C. and Kemp L. (2009). Stakeholders' requirements analysis for a demand-driven construction industry, *Itcon, special issue building information modeling applications, challenges and future directions*, Vol. 14, 629-641.

Zahedi A., Abualdenien J., Petzold F. and Borrmann A. (2022). BIM-based design decisions documentation using design episodes, explanation tags, and constraints, *Journal of information technology in construction*, Vol. 27, 756-780.