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

A MOKA MITIGATION APPROACH FOR COMPUTATIONAL DESIGN CHALLENGES

SUBMITTED: December 2025
PUBLISHED: June 2026
EDITOR: Žiga Turk
DOI: 10.36680/j.itcon.2026.032

Marcus Sandberg, Associate Professor,
Dep. of Civil, Environmental and Natural Resources Engineering
Luleå University of Technology, Sweden
https://orcid.org/0000-0002-2699-2533
marcus.sandberg@ltu.se

Rasmus Mikaelsson, M. Sc.,
Dep. of Civil, Environmental and Natural Resources Engineering
Luleå University of Technology, Sweden
https://orcid.org/0009-0009-2922-3420
rasmus.mikaelsson@hotmail.com

SUMMARY*: Computational Design (CD) has emerged as a key digital technology in the Architecture, Engineering and Construction (AEC) industry, enabling designers to automate tasks, explore complex design alternatives, and collaborate more effectively. Despite these advantages, the adoption of CD faces numerous challenges related to limited knowledge, planning, change management, code readability, and data management. This study investigates how these challenges can be mitigated through the Methodology of Knowledge-Based Engineering Applications (MOKA). Using previously identified CD challenges from Mikaelsson (2022), each challenge was categorized according to the six phases of MOKA—Identify, Justify, Capture, Formalize, Package, and Activate—to analyse where mitigation can occur. The results show that most challenges can be addressed within the first two phases, Identify and Justify. Furthermore, MOKA complements existing research by offering a structured, process-oriented framework to guide CD development and implementation. Although MOKA has its limitations, the study concludes that there are opportunities to enhance computational design practice in AEC if parts of MOKA is applied iteratively during building design. Further details on our mitigation approach are presented and discussed in this study.*

KEYWORDS*: parametric design, KBE, design automation, BIM, generative design.*

REFERENCE*: Sandberg, M., & Mikaelsson, R. (2026). A MOKA mitigation approach for computational design challenges. Journal of Information Technology in Construction (ITcon), 31, 738-760. https://doi.org/10.36680/j.itcon.2026.032*

1. INTRODUCTION

Many important decisions made in the early design phase of buildings influence costs, environmental impact, producibility and other key outcomes. Maierhofer and Menges (2019) state that up to 80 % of a building's ecological and economic performance is determined by decisions taken during this early phase. Consequently, architects and engineers have substantial influence on building projects. At the same time, the architectural, engineering and construction (AEC) industry is becoming increasingly complex and resource-intensive due to the climate crisis and higher material costs. Digitalization is one of the enablers for addressing these challenges, as digital technologies have the potential to change how architects and engineers (from here on referred to as *designers*) carry out building design. In recent years, development and implementation efforts have accelerated (Samuelson & Stehn, 2023).

Computational design (CD) is one such domain of methods and applications. It has evolved from primarily parametric Computer-Aided Design (CAD) models into more exploratory and generative approaches aimed at fully automating or augmenting designer's work to balance competing design objectives (Caetano et al., 2020; Matern, 2021). Mikaelsson (2022) found that CD can provide a more efficient workflow and enable real-time collaboration, allowing design solutions to be generated for tasks too complex (Abrishami et al., 2015) or too time-consuming to perform manually (Boissieu, 2022). Moreover, CD can facilitate effective collaboration and communication between designers and clients (Abrishami et al., 2015) and support more informed decision-making (Mikaelsson, 2022) as it allows automatic evaluation of multiple parameters. However, several challenges associated with CD remain to be addressed (Mikaelsson, 2022).

As with other digital technologies there are early adopters, like programming skilled designers, that on their own initiative start experimenting and exploring possibilities for personal use, such as automating time-consuming tasks or generating new design ideas. But, scaling up the usage from primarily bottom-up towards more top-down implementation, to be sponsored resource wise and supported by the management, is still challenging.

Lack of experience, feedback and reuse of design solutions by designers cause reinvention of the wheel and misses out on cost and time savings. Designers are familiar to use off-the-shelf applications as they are the most digitalized actors of the AEC industry. However, there remains a limited body of knowledge on the development and maintenance of digital applications that require programming—an issue particularly relevant to computational design (CD), where such tools are often developed by the end users themselves. With "developed" we here mean using programming to either build an CD application from scratch (in Python e.g.) or using an existing application (such as Grasshopper or Dynamo) to develop a CD design application.

From the field of computer science there are methodologies for creating computer applications, e.g. SCRUM (Schwaber, 1997), but there is a need for methodologies closer to the AEC context and the computer environment of CAD and BIM. The closest methodology found is MOKA (Methodology of Knowledge-Based Engineering Applications), which originates from research on rule-based CAD-tools from the aerospace and the automotive industries (Stokes, 2001).

There is evidence of the benefits of using MOKA (apart from a few drawbacks) for development of rule-based CAD applications in manufacturing (Sharka, 2007; Barreiro, 2010) and the railway industry (Häußler & Borrmann, 2021). But, according to our knowledge the missing of applications of MOKA within AEC stated by Sandberg (2015) still stays valid to this day for building design. In other words, there are according to the authors' best knowledge no studies trying MOKA for building design. Specifically, there is no comprehensive methodology for CD development and implementation (i.e. a "MOKA version" for CD). Mikaelsson (2022) argues that *"It would therefore be of interest to find solutions that could be implemented and solve several challenges simultaneously."*. When focusing on the broader challenges of adopting or implementing digital tools in the AEC industry – beyond those specific to CD - we identified studies that address parts of the challenges associated with CD. These include designers' knowledge of mathematics and programming (e.g. Sergeeva, 2020), planning for the implementation of digital tools (e.g. Stoyanova, 2020), change management (e.g. Mancini, 2023), code readability (e.g. Fakhoury, 2019) and data management (e.g. Loyola, 2018). Even though these studies exist we see a problem with the lack of an overarching methodology for implementing computational design (or digitalisation for that matter) in the AEC industry.

Therefore, this paper presents a study where MOKA is used to categorise and discuss the mitigation of challenges associated with CD. This, since one of the authors in a previous study interviewed designers about challenges

when developing and using CD. Based on these found CD challenges, presented by Mikaelsson (2022), we analyse and discuss how the different challenges are connected to the MOKA phases and how the phase mitigates the specific challenge. We also discuss limitations with MOKA and as a validation, how it compares to other literature, which can be connected to more specific parts of the CD challenges. These include designers' knowledge and education, digitalisation planning, change management, code readability and data management.

The six phases of MOKA (*Identify*, *Justify*, *Capture*, *Formalize*, *Package* and *Activate*) are used to categorise each challenge and then a discussion of how MOKA could mitigate each challenge, is conducted.

A research question is stated:

How could computational design challenges be mitigated?

We found that MOKA could be used to categorize computational design challenges and that most of the challenges are related to the first phases namely *Identify* and *Justify*. We also discussed how the other MOKA phases can mitigate the challenges. We argue that most of the found CD challenges could be mitigated by using MOKA as an overarching structure (being aware of its limitations) and in some cases MOKA is complemented with specific literature for details such as code readability and change management.

This study extends the current knowledge on how CD challenges could be handled by using MOKA as a structure and process. This contributes to how MOKA as a manufacturing industry-based methodology could be applied in the AEC industry.

2. METHODOLOGY

2.1 Data source and challenge identification

This study's starting point is the computational design (CD) challenges presented by one of the authors in a previous study (Mikaelsson, 2022). In Mikaelsson's study, 16 computational designers (architects and engineers) from the Swedish AEC-industry were interviewed regarding challenges and opportunities associated with CD. The interview transcriptions were revisited, and the master thesis was reviewed to identify relevant challenges, while filtering out those mentioned by only one respondent or those with questionable descriptions. The various challenge descriptions were then grouped by relevance to each other. Each group was determined to represent one main challenge and was labelled with a short description. The various challenges were grouped together in a similar fashion as already presented by Mikaelsson (2022). While Mikaelsson (2022) constitutes the primary empirical source, the present study does not aim to re-validate the challenges themselves, but rather to analyse how these challenges can be interpreted and potentially mitigated using the MOKA methodology.

2.2 Analytical framework: MOKA as coding scheme

The motivation of using MOKA for the categorisation and discussion of mitigation is because it is the closest methodology for development of design automation applications such as those within computational design, although it is criticised by some researchers (see section 3.2.4). The MOKA methodology, consisting of six phases (*Identify, Justify, Capture, Formalize, Package, Activate*), was used as an analytical framework to categorise the identified challenges. The steps for each MOKA phase (Stokes, 2001) where use as criteria to connect the challenges to the phases. See section 3.2 for a more detailed description of MOKA.

2.3 Challenge-to-phase assignment procedure

The grouped challenges were visually mapped around the six MOKA phases. This was an iterative process where both the challenges in the report (Mikaelsson, 2022) as well as the MOKA literature was revisited and discussed several times before the final version of the categorisation was created. In cases where there was initial disagreement regarding the assignment of a challenge to a specific MOKA phase, both authors independently revisited the challenge description and the theoretical definition of the relevant MOKA phases. Discrepancies were then resolved through structured discussions until consensus was reached.

The MOKA literature and CD challenges were analysed to determine where in the MOKA-lifecycle each challenge belonged. Most of the challenges were grouped in two groups: *Identify* & *Justify* and *Capture* & *Formalize*. This was since MOKA describes its phases as being interconnected and performed iteratively. These two groups were

then broken down, and each challenge was linked to its corresponding MOKA phase based on the theoretical description of that phase. Challenges were allowed to be assigned to more than one phase. This process made it easier to analyse the many challenges. Analysis of theory and challenges was conducted iteratively.

The challenges were then analysed using MOKA to determine how each challenge could be mitigated through the steps associated with the various MOKA phases. This process was conducted iteratively by both authors.

2.4 Validation

To compare MOKA with other literature, to show that MOKA fills a gap on the mitigation of similar digitalization challenges, another categorization effort of the CD challenges was conducted in two stages. This can be seen as a theoretical validation conducted by comparing the proposed mitigation approaches for each challenge with findings from existing literature. The authors noted a pattern where most challenges had a connection to one of four areas based on their MOKA mitigation steps: "*Lack of sufficient knowledge"*, "*Plan for digitalization"*, "*Change management"*, and "*Code readability"*. These areas were labelled and then the challenges were analysed and categorised based on them. Literature was searched in these four areas to find mitigation approaches to similar challenges. The second stage of categorization was conducted to check if more areas needed to be added. After adding three additional areas - "*Client communication"*, "*Business models"*, and "*Data management"*- an additional literature review was conducted for all areas, even though most challenges were still connected to the original four. However, two of the new areas were later cancelled and grouped as "*Other challenges*" since too few challenges, two in total, were connected to the two areas "*Client communication"* and "*Business models"*.

The chosen six areas were also used to analyse and discuss how other literature complements MOKA and how MOKA complements existing literature.

3. LITERATURE REVIEW

3.1 Computational design

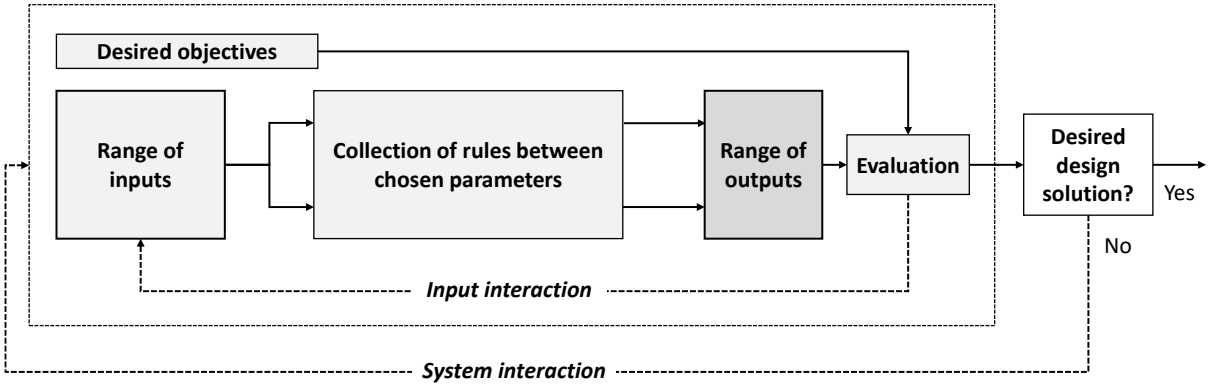

Figure 1: Computational design system, inspired by Boissieu (2022).

Computational design (CD) is a technology within computer-aided design (Boissieu, 2022) that allows the designer to generate multiple design solutions automatically (Caetano et al., 2020), which is beneficial in the early stages of decision-making (Kochański & Borkowski, 2024). Figure 1 shows a schematic representation of the parts of a CD system. The designer choses which parameters, rules and range of input values that are necessary for the desired design solution (Boissieu, 2022). The range of outputs, that is generated from these rules, can then be evaluated by the designer or by a chosen algorithm (or mechanism) that the designer sets (Mukkavaara, 2021). There are two different approaches of Computational design: Parametric design and Generative design. In parametric design the designer interacts with and evaluates the range of inputs and outputs based on the desired outcome while in generative design this process is executed by an algorithm (Caetano et al., 2020). Therefore, it is important for the designer to decide the desired objectives for the final design. If the generated design solution does not satisfy the desired objectives either the designer or the algorithm interacts with the range of inputs to generate a new design solution (Caetano et al., 2020). However, the designer may need to modify the

computational design system, as it can be difficult to predict how the parameters, rules, inputs and algorithms will affect the generated design (Mukkavaara, 2021). Thus, designers need to apply an iterative process with computational design to find the desired design solution (Caetano et al., 2020).

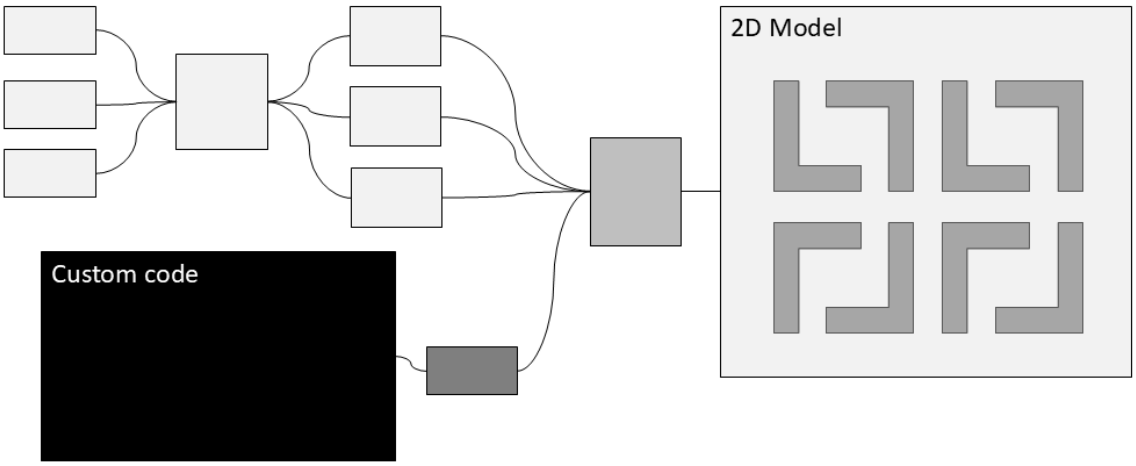

Figure 2: Abstract illustration of Grasshopper and Dynamo with the relations of VPL, TPL and a 2D representation of residential area.

To create and interact with a computational design system the designer uses programming languages to communicate with the computer, such as textual programming languages (TPL) or visual programming languages (VPL) (Ma et al., 2021). Textual programming is more suitable than visual programming for complex tasks while it is also more difficult to learn than visual programming (Ma et al., 2021). Visual programming uses visual blocks pre-programmed with textual programming that can be connected and reused in several combinations (Ma et al., 2021). Two examples of visual programming tools in the AEC industry are Grasshopper and Dynamo. The designer can also use computational design through a software application that allows the designer to generate design solutions without using a programming language.

Zarei (2012) writes that, to master computational design, a designer must acquire the knowledge of both a programmer and a mathematician in addition to expertise in their own field. Computational design thus requires programming, mathematical, and building design knowledge to achieve the desired design solution. Computational design challenges the designer with a different way of thinking and working (Mikaelsson, 2022).

3.2 MOKA and other methodologies for developing design applications

The number of methodologies for developing computer applications for design activities are scarce, especially within the AEC industry. The field of CD is discussed in several studies, but rarely with a focus on how the CD application itself was developed. Instead, the focus usually is on the resulting CD application. From the computer science area there are a couple of methodologies worth mentioning. First, however, we elaborate on an important term that features in the application of these methodologies – namely, the concept of "knowledge-based" as used in knowledge-based systems (KBS) in CommonKADS (de Hoog et al., 1994) and knowledge-based engineering in MOKA (Stokes, 2001). The term "knowledge-based" could simply be explained as rule-based, i.e. the knowledge from of a specific discipline could be formulated as rules. As an example, the sentence "A balcony railing must be at least 1.1 m high if the fall height exceeds 3 m" could be seen as knowledge which could be formulated as a rule. This sentence also contains information (balcony railing height is 1.1 m, fall height is more than 3 m) and data (1.1 m and 3 m). Details of this reasoning can be further studied in Ackoff (1989). In KBSs, applications can typically be queried in much the same way one would consult a human expert, which is why these systems were sometimes referred to as expert systems. In KBE applications, rules are typically linked to geometry. For example, a rule for balcony railings could specify that the railing height is automatically set to 1.1 m when the fall height exceeds 3 m, while for fall heights up to 3 m the designer may adjust the railing height to 1 m.

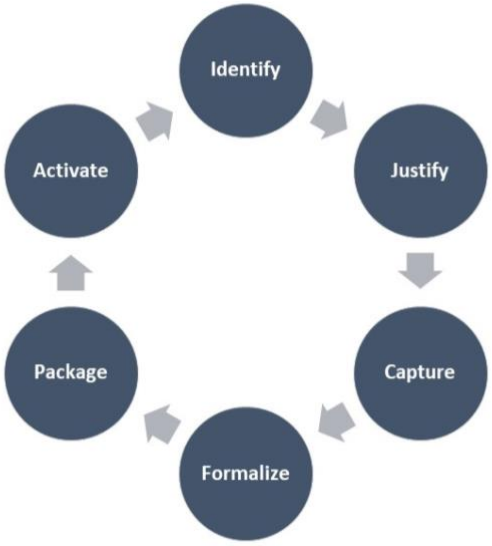

Figure 3: The MOKA life-cycle.

For KBS development, the methodology CommonKADS exist, (de Hoog et al., 1994), as well as a methodology presented by Guida and Tasso (1994). Another methodology is SCRUM, an agile software development framework that is not targeted to a specific application domain (unlike KBE, which is oriented toward mechanical engineering). In SCRUM, development proceeds through iterative short sprints with frequent evaluations.

MOKA (Methodology and software tools Oriented to Knowledge-based engineering Applications) was developed as part of an effort to help Europe keep pace with engineering automation advances in the United States and Japan, particularly within the automotive and aerospace industries (Stokes, 2001). This engineering automation field is called knowledge-based engineering and is in short, a way of automating CAD workflows beyond the possibilities of parametric design (La Rocca, 2012). So, CD and KBE have a common denominator in CAD and programming. MOKA consists of six phases, as shown in Figure 3. Since MOKA is not available through open access or to purchase, we here provide a comprehensive summary of its key parts.

3.2.1 Identify

Identify establishes the objectives, scope, and a high-level technical specification for the design automation application. This phase also involves investigating if knowledge exist about the identified opportunity and some initial thoughts on how this knowledge could be captured. *Identify* has six steps.

The first step involves identifying stakeholders, clarify motivation and requirements. It is also used to determine whether funding should be allocated for developing a business case and creating a detailed plan moving forward. Starting from identified opportunities to automate engineering work, this phase deals with investigating if resources are available in terms of knowledge, people, tools and techniques, hardware, and software.

The second step includes defining roles and scope adding details to how far in the organisation, in what types of projects that the KBE tool should be used and how the intended system will interact with the designer. This step also covers finding out if KBE is the right tool to enable the identified opportunities.

The third step involves identifying possible knowledge sources and determining whether they are suitable for a KBE approach. If knowledge changes rapidly, if domain experts disagree on key points or if experts lack time to both do their normal jobs as well as explaining their knowledge, the knowledge sources are not suitable for KBE.

The fourth step covers ways to capture and analyse knowledge. Interviews and observations are two examples of methods used to capture knowledge, while multi-dimensional techniques and goal decomposition are two categories of knowledge analysis methods.

The fifth step includes identifying target KBE platforms and availability of translators. This means looking at what software to be used for the KBE application and if this software already is used in the organisation.

The last and *sixth step* involves the project leader – referred to here as the knowledge engineer – aggregating the results from the previous five steps into a concept-level KBE specification and presenting it to all stakeholders. If all agree, the process moves further to *Justify*, otherwise it loops back to earlier steps in *Identify* or even aborts.

3.2.2 Justify

Justify involves assessing the costs and timeline outlined in the previous plan, as well as the risks and opportunities presented in a business case. Like the *Identify* phase, the *Justify* phase is also divided into six steps.

The first step deals with estimation of resource requirements and costs. This is done by dividing the work into discrete modules and estimating the cost for personnel, software, hardware etc.

The second step includes assessing technical, cultural, and commercial risks. The identification of technical risks began in the *Identify* phase but is revisited and updated here. Cultural risks concern how the KBE tool affects individuals, groups, the company, and external stakeholders. Commercial risks include long and short-term benefits, changing supply or demand, what if the KBE tool is not used etc.

In *the third step,* acceptance criteria are defined to enable the assessment of the KBE application's success or failure. Here, the agreed objectives and the KBE tool specification must be assessed to identify checks, e.g. milestones, that can be carried out during development to ensure that progress is proceeding as planned.

The fourth step involves generating a project plan. Based on the first plan, ideas in previous steps, key components of the risk and opportunity assessments and the milestones for the acceptance criteria are brought together.

In *the fifth step*, the earlier steps of *Justify* are collected into a business case. This includes risks, costings, aims, the project plan and the concept specification.

The last and *sixth step* focuses on gaining management approval and allows all stakeholders to assess the proposed KBE application from many perspectives. This may involve going back to previous steps to gather more information and break down parts into further details before approval.

3.2.3 The other parts of MOKA

This section gives an overview of the other four parts of MOKA to get the fuller picture of the whole life cycle. These parts include *Capture*, *Formalize*, *Package* and *Activate*.

Capture involves gathering raw knowledge and structuring it into the so-called Informal Model. It is the first step of taking knowledge and preparing it for the KBE system. *Capture* contains five steps: preparation, collecting, and structuring knowledge, checking the Informal model and lastly annotating and filing models in a knowledge repository.

The first step is about preparing for collecting knowledge. Based on the project plan, this step involves checking that the knowledge sources earlier identified, still are relevant and available. Knowledge sources are typically human experts, documents, and computer files. One challenge is that experts tend to be busy. Another is determining whether the previously selected knowledge-capture techniques are appropriate and whether the people involved know how to use them. Here you also need to think about how to store the raw knowledge in terms of videos, sketches, interview notes, extracted texts, drawings, digital files etc. Output from this step is a detailed knowledge capture plan including preparation (e.g. questions to ask), storage, referencing and retrieval plans, capturing sessions and availability of documents and digital files and their access plans.

The second step covers collecting the required knowledge. This is an iterative step were initial capture, in-depth capture, verification and resolving inconsistencies are visited several times, when collecting knowledge from humans. It could also include iterations with the following phases of *Formalize* and *Package*. The initial capture (or acquisition) usually includes open or unstructured interviews as well as focused interviews while the in-depth acquisition usually includes structured interviews and example solving where the expert demonstrates how relevant engineering problems are solved. To verify the correctness of the captured knowledge the knowledge engineering needs to see the design process in action by e.g. letting the expert walk through a made-up design problem. If several experts are involved as knowledge sources, there are risks for inconsistencies that need to be solved. When dealing with capturing knowledge from documents, text fragmentation can be used for structured documents or text transformation for unstructured documents.

The third step focuses on structuring the raw knowledge. This is done using a setup of forms and charts called ICARE (Illustration, Constraint, Activity, Rule, Entity) that helps structure, determine useful parts, organize and classify knowledge as well as identify gaps and blurred areas. All this involves arranging the knowledge into a so-called Informal Model.

The fourth step includes checking the Informal Model together with experts. This is done to make sure that the model representation is correct without conflicts, within scope of the KBE application and detailed enough.

The last and *fifth step* involves annotating and filing models in a knowledge repository. This means checking that the Informal Model is updated with latest information as well as documenting the act of creating the Informal Model.

Formalize then translates the Informal Model into so-called Formal Model which later is used to implement the KBE application in a software. *Formalize* constitutes of six steps: preparation, building the Product Model, building the Process Model, certifying the models and translating the model into neutral format and lastly, incorporating models into the knowledge repository.

The first step covers preparing for formalization which is done by revisiting the plan and revising it for updated resource, time and cost estimates. If the Informal Model is large and suitable it could be modularized. It is time to start using the MOKA Modelling Language (MML) which is based on the Unified Modelling Language (UML).

The second step involves building the so-called Product Model using the MML. As well as using Entity and Constraint forms to build classes.

The third step includes building the so-called Process Model. This involves using Activity and Rule forms to form a flow of activities and connections between. The product and the process model can be done in parallel.

The fourth step focuses on certifying the Formal Model. This means checking for ICARE elements that are not referenced, duplicate parts or other inconsistencies.

The fifth step covers translating the Formal Model into a neutral format, so it gets closer to the actual *Package* step where the neutral format is used in the final a KBE software. MOKA recommends using XML (eXtensible Mark-up Language).

The last and sixth step involves incorporating models into the knowledge repository. Meaning making sure that the Formal Model is documented enough for future maintenance and re-use.

Package entails converting the MOKA Formal Model into code for a KBE application. It contains four steps: retrieving knowledge from the repository, translating knowledge into appropriate form for KBE package, building and updating, as well as testing the KBE application.

The first step includes retrieving knowledge from the repository.

The second step involves translating knowledge into appropriate form for the KBE package.

The third step focuses on building and updating the KBE application.

The fourth step covers testing the KBE application.

Activate encompasses distribution, installation, and use. It has three steps: distributing the KBE application to all end-user sites, introducing the new KBE application and training end-users and lastly the use of the application.

The first step includes distributing the KBE application to all end-user sites.

The second step focuses on introducing the new KBE application and training end-users

The third and final step involves use of the application.

3.2.4 Reports of MOKA use

Earlier studies of MOKA use comes from the manufacturing industry (Skarka, 2007; Barreiro, 2010) but also from the railway industry (Häußler & Borrmann, 2021). Advantages attributed to MOKA include its engineering focus, grounded in computer-aided design and engineering, in contrast to broader methodologies such as CommonKADS, which take a more general knowledge-based systems approach. Another advantage is having a clear distinction between the informal and formal model (Skarka, 2007). Häußler and Borrmann (2021) argue that MOKA's strengths are collecting and structuring knowledge while the need of domain specific adaptations (extensions) is seen as a weakness. Other weaknesses noted includes problems to use CAD applications to represent the informal model (Skarka, 2007). Verhaugen et al. (2012) note that MOKA is focused on the development of the KBE application rather than the use of the KBE application in the design process. Additionally, maintenance and re-use of knowledge is not included. Kügler et al. (2023) argue that MOKA is still the most elaborate methodology for KBE development even though they think it is quite outdated based on the assumption that knowledge was to be transferred from an engineer's head into the computer and not considering transfer of knowledge from other sources such as Product Life-cycle Management, Intranet, documents and simulation data. Also, MOKA might need quite much effort to pursue which is not always needed in simple automation cases (Kügler et al., 2023).

According to the authors best knowledge MOKA has not been developed further since 2001, although it has been the source of inspiration for the KNOMAD methodology by Verhaugen et al. (2012), it has been combined with CommonKADS by van der Velden et al. (2015), and it has been combined with graphical notation methods used by Häußler and Borrmann (2021).

Despite these limitations, MOKA was selected for this study as it remains one of the most comprehensive and well-established methodologies for knowledge-based engineering. To the authors' best knowledge, no alternative methodology provides a similarly structured and extensively documented framework that is directly applicable to the development of computational design applications in an AEC context.

3.3 Mitigating digitalisation challenges in AEC

In this section we want to address the lack of literature within general approaches for mitigating digitalisation challenges within the AEC industry. Therefore, we have chosen subsections connecting to some of the main challenges when pursuing digitalisation in the AEC-industry. When it comes to general digitalisation challenges in the AEC industry, lack of knowledge is usually mentioned (Chen et al., 2025; Samuelson & Stehn, 2023). Apart from knowledge, inability to change is also commonly referred to (Kotter, 1995; Bosch-Sijtsema et al. 2021). Here we also discuss the ability to plan for the digitalization, code readability approaches for making coding easier and data management.

3.3.1 Lack of sufficient knowledge within programming and mathematics

To create digital technology, you need knowledge within programming and to use digital design tools such as CD require mathematical skills. Chen et al. (2025) have mapped the competence of digital tools for AEC actors where they have included BIM but not CD. Their findings highlight the importance of focused training and support efforts for contractors, SMEs, and start-ups. With regard to building architects, Sergeeva (2020) argues that mathematics will become even more important for architects in the future in order to keep pace with developments in computing. Likewise, Zhang et. al (2025) discusses the importance for architects to learn programming to better understand computational design.

3.3.2 Change management

Humans are generally resistant to change, and this tendency often increases with age, although there are of course exceptions (Kotter, 1995). Mancini (2023) argues that Kotter's (1995) eight steps for change are not directly applicable to digitalization, as they do not place sufficient emphasis on communication, agility and participation. However, the eight steps contain important parts for a generic change project.

3.3.3 Planning for digitalization

When it comes to planning for digitalization there are, to the best of our knowledge, no papers on how specific companies within AEC should go about. There are examples on industry level for digital transformation in

construction (Samuelson & Stehn, 2023). Stoyanova (2020) presents recommendation for successful digitalization in construction including: developing a digitalization strategy, rethinking the workspace, improving internal organization, communication transition, digitalization of customer experience and building a digital platform. Manzoor et al. presents a strategy for mitigating BIM implementation barriers including lack of experience, lack of BIM training and high cost (2021).

3.3.4 Code readability

An important aspect of code maintenance is comprehension, or readability (Fakhoury, 2019), which determines a programmer's ability to understand the code and therefore affects their ability to maintain it. To increase readability a programmer can use indentation (Kesler et al., 1984; Fakhoury et al., 2019) and blank spaces to structure and separate the code (Fakhoury et al., 2019). Programmers can also write comments in the code to increase readability (Woodfield et al., 1981; Fakhoury et al., 2019). However, Fakhoury et al. (2019), Woodsfield et al. (1981), and Kesler (1984) only address textual programming. Ma et al. (2021) describe visual programming as easier to learn than textual programming while Zarei (2012) writes that learning VPL is still a steep learning curve for designers (Zarei, 2012). There are no papers, to our knowledge, that presents methods for increasing readability of visual programming.

3.3.5 Data management

The AEC industry's main data management technology is BIM (Loyola, 2018). To achieve the largest benefits of BIM, in practice, all participants must use the models as integrated repositories of all building data through the whole project lifecycle (Loyola, 2018). However, the benefits of BIM are lost in most projects since a lifecycle approach is not applied (Halttula et al., 2020; Loyola, 2018).

Furthermore, the same product data should be handled in all phases and by every stakeholder in a project (Halttula et al., 2020). However, Halttula et al. (2020) describes that this rarely happens in AEC projects where product data instead is stored and managed differently by many organisations. Only a minor fraction of AEC project data is properly collected, organized, and stored – instead, most data are non-systematized, extremely scattered, wrong, or simply lost (Loyola, 2018). Data management in the AEC industry is thus in practice still in its infancy.

Additionally, Abrishami et al. (2014) write that although CD tools assist designers in their projects, these tools fail to meet the basic principles of information modelling and data management requirements.

4. FOUND AND CATEGORISED CHALLENGES

In this section the selected challenges from Mikaelsson (2022) are listed in table 1 with the main corresponding MOKA phases indicated.

Table 1: Location of Computational Design (CD) challenges in MOKA cycle.

Challenge	Identify	Justify	Capture	Formalize	Package	Activate
Difficult to know if CD is the appropriate solution	X					
Uncertain if some scripts save time and/or provide add'l value	X					
Difficult to understand CD	X					
Few designers with CD competence	X					
Some designers lack programming knowledge	X					
Architects lack sufficient mathematical knowledge	X					
Stakeholders lack knowledge of CD	X					
Lack competence to develop software	X					
Lack computer power	X					
Some software tools are not made for CD's iterative process	X					
Constrained by current software	X					
IT-security process constrains new adoption of plugin-software	X	X				
High upfront investment	X	X				
Unrealistic pos. and neg. expectations by the des., coll. & stakeh.	X	X				
Fear of change		X				
Unappropriated business model		X				
Not enough time scheduled for developing scripts		X				
Lack support from management (during learning and using CD)		X				
Opposition from colleagues		X				
General fear of being automated and replaced		X				
Difficult to change existing processes		X				
CD process collides with existing design process		X				
Difficult to assess risks, costs, and benefits of CD implementation		X				
Difficult to collect trustworthy and structured data			X			
Difficult to interpret and maintain data				X		
Lack planning of code structure				X		
CD is unpredictable				X		
Complexity increases when more parameters are used				X		
Lack of structured scripts				X		
Difficult to understand old scripts				X		
Difficult to learn programming languages				X		
VPL can't handle complexity like TPL	X	X		X		
TPL is more difficult to learn than VPL	X	X		X		

In table 2 mitigations and motivations are presented with each challenge for the categorisation of table 1. Challenges with similar motivation and mitigation are presented together in table 2.

Table 2: Motivations for the categorisation of challenges in the MOKA life-cycle.

Challenge	MOKA-phase	Motivation of categorisation and mitigation of challenge
Difficult to know if CD is the appropriate solution	Identify	The designer needs to clarify motivations and requirements of the desired design solution to establish if the intended CD approach is beneficial enough to be developed and used. By spending time of breaking down motivations and requirements, it becomes possible to mitigate these challenges.
Uncertain if some scripts save time and/or provide additional value.	Identify	
Difficult to understand CD	Identify	Designers need to identify the requirements of the intended CD idea and available resources such as knowledge, techniques, hardware and software. The designers might need to change their CD idea to match their limitations with the identified requirements. Furthermore, designers must identify stakeholders and investigate how to communicate the knowledge that stakeholders lack and engage them in the process of developing CD. Stakeholders might have a different perspective than the designer and an advanced CD application might not be needed. Which could save resources for the designer who possibly had a too ambitious idea in the beginning. Designers motivated enough to explore CD outside of their available resources need to create a detailed plan on how to acquire the necessary resources, such as knowledge sources or possible collaborators.
Few designers with CD competence	Identify	
Some designers lack programming knowledge	Identify	
Architects lack advanced mathematical knowledge	Identify	
Stakeholders lack knowledge	Identify	
Lack competence to develop software	Identify	
Lack computer power	Identify	*Identify* includes investigating requirements of the CD application and what resources are available to the designer to meet those requirements, such as hardware (computer power) or software tools. Designers must create a plan to either acquire the necessary resources or change their intended CD application until the requirements meet their available resources. Designers should aim to acquire software tools that support an iterative process. Designers could also investigate if it is possible for them to develop iterative software tools. Designers need to investigate if and how their CD application is constrained by current software. If it is, designers should create a plan how to either develop their own software or how their CD application can be adapted to current software. This process should be conducted within the limitations of the designer and what CD applications are valuable to develop.
Some software tools are not made for CD's iterative process	Identify	
Constrained by current software	Identify	
IT-security process constrains new adoption of plugin-software	Identify, Justify	*Identify* includes investigating stakeholders and the designer's limitation**s**. An IT-security department can be a stakeholder. This challenge can be mitigated with steps in *Identify* and *Justify*. Designers need to create a plan with identified stakeholders, motivating the value of CD and the use of plugin-software as well as clarifying requirements, such as faster approval. This plan should also include potential risks and how to address those risks. The designer must engage stakeholders like IT-security and convince them to justify development of their CD application. Designers should investigate if plugin-software are required to develop the CD application. Designers should also investigate if other CD applications, that does not require plugin-software, are more valuable to clients.

Challenge	MOKA-phase	Motivation of categorisation and mitigation of challenge
High upfront investment	Identify, Justify	The first step of *Identify* involves clarifying motivations, identify requirements of the intended CD application and investigate if the necessary resources are available. *Justify* is about assessing costs, time, risks and opportunities with the CD application and finally to gain management approval. Designers need to investigate if the CD application can deliver value over time, such as being reused in multiple projects, or that it is more valuable than the cost of development. Designers should also investigate if the necessary resources could be reused in multiple or future CD applications.
Unrealistic positive and negative expectations by the designer, colleagues, and stakeholders	Identify, Justify	Designers should clarify motivations and identify requirements for the CD application as well as communicate this to colleges and stakeholders that should be involved in the design process. *Justify's* first step involves estimating the cost and time based on resource requirements of the CD application. The second step is instead about identifying risks with the application. Designers should also use the third step of *Justify* by creating acceptance criteria to determine success or failure as well as milestones, objectives and specifications of the CD application. Designers should create a project plan based on the previous steps in *Identify* and *Justify*.
Fear of change	Justify	*Justify* is about involving stakeholders, present a business case and get approval to move on. *Justify's* second step involves assessing cultural risks such as how the CD application effect on individual, group, company, and external level. The sixth step of *Justify* allows all stakeholders to assess the CD application from various perspectives.
Opposition from colleagues	Justify	
General fear of being automated and replaced	Justify	
Not enough time scheduled for developing scripts	Justify	Designers need to involve stakeholders, such as clients, management and colleagues in the development of their CD application and convince them of the value of CD to get their approval. The value of CD might be different to each stakeholder. By following the steps in *Justify* as well as the previous phase *Identify*, designers can provide more insight for stakeholders.
Lack support from management (during learning and using CD)	Justify	
Difficult to change existing processes	Justify	Designers will need approval to get support from stakeholders such as more time for learning, developing and using CD. Colleagues might also possess valuable knowledge that can help the designer develop CD. By involving stakeholders, designers could gain their trust and to change existing processes for a CD application. Getting approval depends on how well the designer have presented CD as a valuable business case. This is helped by earlier examples that can be demonstrated visually.
Computational design process collides with existing design process	Justify	
Difficult to assess risks, costs, and benefits of CD	Justify	*Justify* includes assessing risks, costs, and opportunities. This is done by breaking down the parts of the planned CD application, estimating its cost and discussing plausible opportunities.
Unappropriated business model	Justify	The fifth step of *Justify* focuses on collecting previous step's results into a business case. By including risks, costs, objectives, project plan and concept specifications, designers could together with business developers better assess which business model would be appropriate. But changing business model could be a complex process that require long term work since the architectural firms have a long history of using the hourly-based business model.
Difficult to collect trustworthy and structured data	Capture	*Capture* is about capturing and structuring knowledge. The trustworthiness of data can be checked with colleagues or the author of the data source.

Challenge	MOKA-phase	Motivation of categorisation and mitigation of challenge
Difficult to interpret and maintain data	Formalize	*Formalize* include preparation of knowledge (including data) collected during *Capture*. If comments are used to explain the code, it is easier to maintain. If using Grasshopper or Dynamo the designer can comment code by inserting spaces or textboxes, which also can be used to structure and separate different parts of a VPL script.
Lack planning of code structure	Formalize	*Formalize* involves breaking down the different coding parts and creating a structure. This structure can be used to plan for the coding.
CD is unpredictable Complexity increases when more parameters are used	Formalize Formalize	By structuring the coding implementation and understanding the input and output of each part it becomes easier to predict the final output.
Lack of structured scripts	Formalize	*Formalize* involves creating structured code.
Difficult to understand old scripts	Formalize	*Formalize* involves building the Formal model which enables the designer to reuse and better understand knowledge stored in the knowledge repository. This is also mitigated if the code is explained through comments.
Difficult to learn programming languages VPL cannot handle complexity like TPL TPL is more difficult to learn than VPL	Formalize Identify, Justify, Formalize Identify, Justify, Formalize	Designers must identify the requirements for their CD application. While VPL is easier to learn, TPL enables more complexity. Designers must evaluate which programming language that is more valuable within their limitations. If an application requires more complexity but the designer lack the competence to use TPL then the designer must evaluate (with *Justify*) if the application is valuable enough to continue. A different application with less complexity might be more valuable and then VPL could be used. *Formalize* can help designers break down a complex solution and get a better understanding how to proceed. Which could help designers understand how to use TPL or combine VPL with TPL.

In table 3 each challenge is categorised into literature fields and presented with its MOKA-phase in abbreviation (I for *Identify*, J for *Justify*, C for *Capture*, and F for *Formalize*).

Table 3: Challenges categorized into literature fields with MOKA-phases in abbreviations.

Challenge	Lack of sufficient knowledge	Plan for digitalization	Change management	Code readability	Data management	Other
Difficult to understand CD (**I**)	X					
Few designers with CD competence (**I**)	X					
Some designers lack programming knowledge (**I**)	X					
Architects lack sufficient mathematical knowledge (**I**)	X					
Difficult to know if CD is the appropriate solution (**I**)	X	X				
Lack competence to develop software (**I**)	X	X				
Lack computer power (**I**)		X				
Some software tools are not made for CD's iterative process (**I**)		X				
Constrained by current software (**I**)		X				
Uncertain if some scripts save time and/or provide additional value (**I**)		X				
High upfront investment (**I, J**)		X				
VPL can't handle complexity like TPL (**I, J, F**)		X				
Not enough time scheduled for developing scripts (**J**)		X				
Difficult to assess risks, costs, and benefits of CD implementation (**J**)		X				
IT-security process constrains new adoption of plugin-software (**I, J**)		X	X			
Unrealistic positive and negative expectations by the designer, colleagues, and stakeholders (**I, J**)			X			
Fear of change (**J**)			X			

Challenge	Lack of sufficient knowledge	Plan for digitalization	Change management	Code readability	Data management	Other
Lack support from management (during learning and using CD) (**J**)			X			
Opposition from colleagues (**J**)			X			
General fear of being automated and replaced (**J**)			X			
Difficult to change existing processes (**J**)			X			
Computational design process collides with existing design process (**J**)			X			
Lack planning of code structure (**F**)				X		
CD is unpredictable (**F**)				X		
Complexity increases when more parameters are used (**F**)				X		
Lack of structured scripts (**F**)				X		
Difficult to understand old scripts (**F**)				X		
TPL is more difficult to learn than VPL (**I, J, F**)				X		
Difficult to learn programming languages (**I, F**)	X			X		
Difficult to collect trustworthy and structured data (**C**)				X	X	
Difficult to interpret and maintain data (**F**)				X	X	
Stakeholders lack knowledge of CD (**I**)						X
Unappropriated business model (**J**)						X

5. DISCUSSION

In our analysis of categorizing challenges to the different MOKA stages – many challenges were connected to *Identify* and *Justify*, MOKA's first and second stages. Even though these two stages are not the focus in MOKA, there are still useful ideas to use with these challenges. We suggest that a designer that intends to use CD techniques could review our list of challenges and their corresponding MOKA part when planning the CD effort. Especially, for larger, resource intensive, automation efforts as simpler efforts are prone to be solved without MOKA's *Capture* and *Formalize*. Another more reactive way is to visit our list when challenges arise and use the MOKA instructions to mitigate the challenge.

It is important that MOKA is used iteratively instead of linearly, both in between separate phases and when cycling through all six phases of MOKA. Some challenges like "Difficult to estimate costs, risks, and benefits" could be difficult to tackle only within *Justify*, one phase of MOKA, although *Justify* is about estimating costs, risks, and benefits. An iteration through *Identify* and *Justify* could yield better insights into the intended CD application, planning and overall direction. Iteration through *Capture* and *Formalize* could result in a more predictable CD solution where knowledge can be reused in future projects. *Package* and *Activate* are the final phases that lead to the development and implementation of the CD application. In the next iteration, starting with Identify, the designer would have better insights to estimate costs, risks, and benefits of the CD application. Thus, a full cycle would more effectively mitigate the challenge "Difficult to estimate costs, risks, and benefits" than solely implementing the steps of *Justify*.

So, we can see that MOKA offers some general advice that might mitigate some of the presented CD challenges. Apart from MOKA, there are literature that discusses similar challenges on a general level (not specifically for CD), and we have chosen to divide those into the following sections: "*Lack of sufficient knowledge*", "*Planning for digitalization*", "C*hange management*", "C*ode readability*", "*Data management*" and "*Others*". Let us now discuss how these sources differ or complement MOKA. For details on our mitigation approach see Table 2.

5.1 Challenges related to lack of sufficient knowledge

There are studies that argue that designers need more advanced mathematical knowledge to keep up with the digital development (Sergeeva, 2020) targeting the challenge "*Architects lack sufficient mathematical knowledge*". For designers already working in the AEC industry, there are studies that have mapped the competence of designers and suggested mitigation strategies include involving practitioners within the AEC industry in training programs (Chen et. al, 2022) to mitigate challenges like "*Some designers lack programming knowledge*", "Lack competence to develop software" and "Difficult to learn programming languages". This could also mitigate "*Difficult to understand CD*". But, taking a course in CD would be even better for this challenge, as well as "Difficult to know if CD is the appropriate solution" and "*Few designers with CD competence*". MOKA connects to the challenge of having sufficient knowledge in the beginning of the *Identify* part where available resources in terms of for example people is investigated, here there is a possibility to at least map sufficient or not sufficient knowledge although not mitigating this lack of knowledge. MOKA also contributes to building up a knowledge-base of CD applications for future re-use of knowledge which would strengthen the lack of expertise in mathematics, programming and CD development overall. Advanced mathematical and programming knowledge need extensive training.

5.2 Challenges related to planning for digitalisation

We have not found any other literature on how to plan for the implementation of CD. There are some papers discussing digitalization as a broad concept, e.g. Samuelson and Stehn (2023) and Stoyanova (2020). When it comes to BIM there are some papers on how to mitigate implementation barriers e.g. Manzoor et al. (2021) where the challenge "*Lack of expertise*" which connects to our CD challenge "*Lack competence to develop software*" is mitigated simply by hiring a BIM expert while we, based on MOKA, reason what the designer can do by balancing the requirements of the CD idea with the limitations of the designer's expertise.

Regarding the challenge "*Uncertain if some scripts save time and/or provide additional value*" we have not found literature that specifically deals with this. MOKA have a checklist when automation is not suitable which could be used also for CD, but otherwise MOKA intends to mitigate such challenges on an overarching level. Stoyanova (2020) mentions "Developing a digitalisation strategy" but do not elaborate on how this is done.

Manzoor et al. (2021) suggest allocating parts of the budget for digitalisation to mitigate the "*High cost*" barrier which is similar to the *Justify* part of MOKA where costs are assessed. This also connects to the challenge "*Not enough time scheduled for developing scripts*" since budget allocations could mitigate this challenge. However, since this study takes the perspective of the designer our approach with MOKA complements the reviewed literature for these two challenges.

For the challenges "*Difficult to assess risks, costs and benefits of CD implementations*", *"VPL can't handle complexity like TPL"*, "*Lack computer power*", "*Some software tools are not made for CD*", "*Constrained by current software*" and "*IT-security process constrains new adoption of plugin-software*" no specific mitigation strategy was found in the reviewed literature. With MOKA we reason that all challenges in table 3 regarding "*Planning for digitalization"* primarily can be mitigated by choosing the right CD application to develop within the designer's limitations. As mentioned earlier for the challenge "*Difficult to assess risks, costs and benefits of CD implementation*" MOKA's entire life-cycle should be applied for effective mitigation. So, here our approach of using MOKA complements the lack of mitigation steps in the reviewed literature.

5.3 Challenges related to change management

The challenges that are on an overarching level: "*Fear of change*", "*Difficult to change existing processes*" and "*Opposition from colleagues*" connect to Kotter's (1995) change model while the challenges that are more specific to digitalization has connection to the elaborated change model that Mancini (2023) suggests. The elaborated change model includes repeated communication which could mitigate "*Unrealistic positive and negative expectations by the designer, colleagues and stakeholders*". Regarding "*Computational design process collides with existing design process*" Mancini (2023) suggests that Kotter's eight step "Anchoring new approaches in the culture" only is applied for larger projects, which also are deemed to have impact on the working culture. Mancini (2023) suggests an iterative and agile approach where small steps are changed at a time which possibly could make adoption of new ways of working more smooth connecting to the challenge "*General fear of being automated and replaced*". In the elaborated change model, it is also suggested to include stakeholder management, which could involve IT-security personnel, which is needed to mitigate the challenge "*IT-security process constrains new adoption of plugin-software*". Another elaboration is adding change coaches which could mitigate "*Lack support from management (during learning and using CD)*". MOKA has a proactive approach, to change in line with the Kotter step "Developing a vision and strategy", where plans to minimize change resistance are developed in the *Identify* and *Justify* parts. So MOKA do not offer ways to reactively deal with change management but could ideally reduce the risk for these challenges to rise if executed perfectly especially in the *Justify* phase. But, Mancini (2023) adds details on change management in digitalisation that could complement MOKA.

5.4 Challenges related to code readability

Although Fakhoury et al. (2019), Woodfield et al. (1984), and Kesler et al. (1984) outlines steps to increase code readability, they focus on TPL which is less accessible to designers in AEC due to the challenge "*TPL is more difficult to learn than VPL*". With MOKA we reason that designers could better plan and structure code for their CD implementation which address all challenges related to code readability in table 3. The challenge "*Complexity increases when more parameters are used*" is likely connected to the lack of readability due to how spaghetti-code is easily created on VPL-platforms like Grasshopper and Dynamo when the number of visual elements increase. We found no literature that explicitly address this challenge.

The challenges "*Difficult to learn programming languages*", and "*TPL is more difficult to learn than VPL*" are likely to persist in the future. We reason instead according to MOKA that designers should focus their time on developing CD that provide value within the limitations of the designer. For the challenge "*CD is unpredictable*" we reason that MOKA's process will make CD more predictable, and clear.

5.5 Challenges related to data management

Data management in the AEC industry is still in its infancy based on descriptions of Loyola (2018) and Halttula et al. (2020), two articles with comprehensive literature reviews of data management in the AEC industry. Furthermore, CD tools fail to meet the requirements for proper data management and information modelling according to Abrashami et al., (2014). Which relates to the two CD challenges "*Difficult to collect trustworthy and structured data*" and "*Difficult to interpret and maintain data*".

MOKA complements current research with a clear process in *Capture* and *Formalize* for challenges related to collection, structuring, interpretation and maintenance of data using the UML-standard among others. We believe that this also could benefit CD application development.

5.6 Other challenges

Bosch-Sijtsema et al. (2021) writes about a hype factor in AEC where new technologies go through four phases: confusion (1), excitement (2), experimentation (3), and integration (4). Lower knowledge and usage of a technology lead to a higher hype factor (Bosch-Sijtsema et al., 2021). For example, Bosch-Sijtsema et al. (2021) positions BIM in the integration phase while AI and automation are positioned in the excitement phase. The excitement phase represents technologies with high expectations that many have heard of even though few have used them (Bosch-Sijtsema et al., 2021). Which connects to two of our challenges: "Stakeholders lack knowledge of CD" and "Unrealistic positive and negative expectations by the designer, colleagues, and stakeholders". We reason with MOKA how the designer can introduce stakeholders to CD and include them in the development process.

Bosch-Sijtsema et al. (2021) also writes that new technologies in AEC are hindered by current business models and the industry's reluctance to change. This connects to the challenge "Unappropriated business model". We reason that even though MOKA suggest choosing an appropriate business model it is an unrealistic mitigation step for an individual designer to implement without collaborating with a business developer.

5.7 Wider discussion topics

5.7.1 Capture, Package & Activate

In table 1 only one challenge is connected to *Capture* and no challenges are connected to *Package* and *Activate*. The motivation to keep these are as explained earlier: MOKA is meant to be used iteratively through all phases and in between phases to increase mitigation effectiveness. For example, *Identify* applies a light version of *Capture* while *Formalize* relies on *Capture* to create the process model and product model as well as the MML structure. *Package* and *Activate* address the development and usage of the CD application, without them the MOKA-cycle becomes uncomplete.

5.7.2 Limitations of MOKA

Although this study has presented MOKA as a valuable methodology for mitigating CD challenges, MOKA could be an administrative burden depending on which CD challenges that need to be mitigated, and which CD application that have been chosen. Unlike *Identify* and *Justify*, which focus on choosing the right CD application to start developing, the designer might need more time to work with *Capture* and *Formalize* since data need to be collected, structured, and then produced into a product model and process model. Furthermore, *Capture* and *Formalize* lays the foundation for reusing knowledge in future CD applications, which is not as beneficial when the reuse of knowledge is less valuable, such as in cases where CD applications are developed for use in only one project or in simpler automation cases as noted by Kügler et al. (2023). Also, MOKA offers little guidance in how to use the developed automation application in the design process Verhaugen et al. (2012).

5.7.3 Generative AI & MOKA

With the recent year's increased usage of generative AI, such as ChatGPT, Copilot etc., it is likely that challenges related to coding will be less substantial. The positive impact with generative AI for learning coding is indicated in several studies, e.g. Garcia (2025) and Bucaioni et al. (2024). Generative AI could also impact the MOKA life-cycle positively. As mentioned previously MOKA could be an administrative burden which might be less of a challenge with Generative AI by offloading time-consuming administrative tasks such as those Kügler et al. (2023) describe as technically obsolete and therefore enable a more data-driven or automatic approach to using MOKA. The possibilities of generative AI need to be further investigated for this matter.

5.7.4 Computational design – a new field in research

Computational design (CD) is a young field in AEC where research is focused on technical aspects rather than the user's perspective (Saadi, 2024) being part of a design processes and dealing with challenges of CD. This is like design automation within AEC where Sandberg (2015) noted a focus on applications rather than processes.

Therefore, this study contributes to the current research on CD since we instead focus on the processes and how to mitigate CD challenges from the perspective of the designer.

5.7.5 Limitations of this study

This study proposes an approach, based on MOKA, on how CD challenges could be mitigated and thus our study is not an empirically proved mitigation strategy. The only empirical data is the CD challenges from Mikaelsson (2022), a qualitative study that interviewed 16 architects and engineers in the Swedish AEC industry. To our knowledge there is no other empirical research of CD challenges conducted more recently and as mentioned earlier, other research of CD tends to focus on technical aspects rather than challenges. Which is linked to the lack of other research of CD challenges. This study was also limited by resources, and an empirical study was considered too expensive.

Thus, the validation was instead conducted through a literature review in two stages by analysing and comparing other articles with our MOKA-approach (as described in the methodology section).

6. CONCLUSION

This section gives a summary to our research question and outlines future research. The study concludes that there are opportunities to enhance computational design practice in AEC if MOKA is applied iteratively during building design, regarding larger design automation efforts. Simpler automation efforts are prone to manage without MOKA's *Capture* and *Formalize* phases.

6.1 How can computational design challenges be mitigated?

In this paper we have categorised found challenges in Mikaelsson (2022) related to usage of computational design (CD). We have investigated how the MOKA methodology possibly could mitigate these challenges, and we argue that CD challenges could be mitigated by using MOKA's six phases iteratively where most challenges could be mitigated in MOKA's first and second phase, *Identify* and *Justify*. Overall, our presented MOKA mitigation strategy is about designers choosing the right CD application to develop within their limitations and which CD applications clients are willing to pay for. By choosing the right direction in the beginning most CD challenges could proactively be mitigated or prevented before they arise. Research of the literature fields "Lack of sufficient knowledge", "Planning for digitalization", "Change management", "Code readability", "Data management", and "Other" showed that our mitigation steps with MOKA complements current research even though some parts of the literature could be used to complement MOKA.

6.2 Future research

Future research should focus on investigating if our approach with MOKA actually mitigates computational design challenges for designers in architectural and engineering. Preferably, such studies should be conducted with designers by interviewing and practical testing in a real project. Furthermore, future research should investigate if the established computational design challenges in Mikaelsson (2022) are still experienced as challenges and/or if more challenges exist. It would also be of interest to study the application of MOKA with generative artificial intelligence to mitigate computational design challenges. Further research is also needed regarding how the other parts of MOKA, such as *Capture* and *Formalize*, could be used in more detail with computational design.

REFERENCES

Abrishami, S., Goulding, J., Rahimian, F.P., & Ganah, A. (2015). Virtual generative BIM workspace for maximising AEC conceptual design innovation. *Construction Innovation: Information, Process, Management*, *15*(1), 24–36. https://doi.org/10.1108/CI-07-2014-0036

Abrishami, S., Goulding, J.S., Rahimian, F.P., & Ganah, A. (2014). Integration of BIM and generative design to exploit AEC conceptual design innovation. *Journal of Information Technology in Construction*, *19*, 350–359. Available at: https://www.itcon.org/2014/21

Ackoff, R.L. (1989). From data to wisdom. *Journal of Applied Systems Analysis*, *16*, 3–9.

Barreiro, J., Martínez Pellitero, S., Cuesta, E., & Álvarez, B.J. (2010). Comparative synthesis between STEP and MOKA methodologies and new proposal for the scope of manufacturing and inspection processes. *Annals of DAAAM & Proceedings of the International DAAAM Symposium*, *21*(1).

Boissieu, A.D. (2022). Introduction to computational design: Subsets, challenges in practice and emerging roles. In *Industry 4.0 for the Built Environment: Methodologies, technologies, and practices*, 55–75. Springer. https://doi.org/10.1007/978-3-030-82430-3_3

Bosch-Sijtsema, P., Claeson-Jonsson, C., Johansson, M., & Roupe, M. (2021). The hype factor of digital technologies in AEC. *Construction Innovation*. https://doi.org/10.1108/CI-01-2020-0002

Bucaioni, A., Ekedahl, H., Helander, V., & Nguyen, P.T. (2024). Programming with ChatGPT: How far can we go? *Machine Learning with Applications*, 15, 100526. https://doi.org/10.1016/j.mlwa.2024.100526

Caetano, I., Santos, L., & Leitao, A. (2020). Computational design in architecture: Defining parametric, generative, and algorithmic design. *Frontiers of Architectural Research*, *9*(2), 287–300. https://doi.org/10.1016/j.foar.2019.12.008

Chen, K.W., Janssen, P., Aviv, D., Ninsalam, Y., & Meggers, F. (2022). A framework for considering the use of computational design technologies in the built environment design process. *ITcon*, *27*, 1010–1027. https://doi.org/10.36680/j.itcon.2022.049

Chen, X., Chang-Richards, A., Ling, F.Y.Y., Yiu, T.W., Pelosi, A., & Yang, N. (2025). Digital technologies in the AEC sector: a comparative study of digital competence among industry practitioners. *International Journal of Construction Management*, *25*(1), 63–76. https://doi.org/10.1080/15623599.2024.2304453

de Hoog, R., Martil, R., Wielinga, B., Taylor, R., Bright, C., & van de Velde, W. (1994). *The CommonKADS model set* (Report). University of Amsterdam.

Fakhoury, S., Roy, D., Hassan, A., & Arnaoudova, V. (2019). Improving source code readability: theory and practice. *IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, Montreal, Canada, 2–12. https://doi.org/10.1109/ICPC.2019.00014

Garcia, M.B. (2025). Teaching and learning computer programming using ChatGPT: a rapid review of literature amid the rise of generative ai technologies. *Education and Information Technologies*, *30*, 16721–16745. https://doi.org/10.1007/s10639-025-13452-5

Guida, G., & Tasso, C. (1994). Design and development of knowledge-based systems: from life cycle to methodology. John Wiley & Sons.

Halttula, H., Haapasalo, H., & Silvola, R. (2020). Managing data flows in infrastructure projects – the lifecycle process model. *ITcon*, *25*, 193–211. https://doi.org/10.36680/j.itcon.2020.012

Häußler, M., & Borrmann, A. (2021). Knowledge-based engineering in the context of railway design by integrating BIM, BPMN, DMN and MOKA. *ITcon*, *26*. https://doi.org/10.36680/j.itcon.2021.012

Kotter, J.P. (1995). Leading change: why transformation efforts fail. *Harvard Business Review*. https://doi.org/10.1109/EMR.2009.5235501

Kochański, Ł., & Borkowski, A. S. (2024). Automating the conceptual design of residential areas using visual and generative programming. *Journal of Engineering Design*, *35*(2), 195–216. https://doi.org/10.1080/09544828.2024.2303282

Kesler, T.E., Uram, R.B., Magareh-Abed, F., Fritzsche, A., Amport, C., & Dunsmore, H.E. (1984). The effect of indentation on program comprehension. *International Journal of Man-Machine Studies*, *21*, 415–428. https://doi.org/10.1016/S0020-7373(84)80068-1

Kügler, P., Dworschak, F., Schleich, B., & Wartzack, S. (2023). The evolution of knowledge-based engineering from a design research perspective: literature review 2012–2021. *Advanced Engineering Informatics*, *55*, 101892. https://doi.org/10.1016/j.aei.2023.101892

La Rocca, G. (2012). Knowledge-based engineering: between AI and CAD. Review of a language-based technology to support engineering design. *Advanced Engineering Informatics*, *26*(2), 159–179. https://doi.org/10.1016/j.aei.2012.02.002

Loyola, M. (2018). Big data in building design: a review. *ITcon*, *23*, 259–284. https://www.itcon.org/2018/13

Maierhofer, M., & Menges, A. (2019). Towards integrative design processes and computational design tools for adaptive architectural structures. *International Conference on Emerging Technologies in Architectural Design*, Toronto, Canada.

Mancini, L.A. (2023). *The role of change management in corporate digitalization: the relevance of Kotter's eight-step change model within IT/digitalization projects*. [Master thesis, ISCTE – Intsituto Universitário de Lisboa]. Repositório do ISCTE.

Manzoor, B., Othman, I., Gardezi, S.S.S., Altan, H., & Abdalla, S.B. (2021). BIM-based research framework for sustainable building projects: A strategy for mitigating BIM implementation barriers. *Applied Sciences*, *11*(12), 5397. https://doi.org/10.3390/app11125397

Mathern, A. (2021). *Addressing the complexity of sustainability-driven structural design: computational design, optimization, and decision making* [Doctoral dissertation, Chalmers University of Technology]. DiVA Portal. urn:nbn:se:trafikverket:diva-16345

Mikaelsson, R. (2022). *Computational Design in the AEC industry: Applications and Limitations* [Master thesis, Luleå University of Technology]. DiVA Portal. https://www.diva-portal.org/smash/get/diva2:1719725/FULLTEXT01.pdf

Mukkavaara, J. (2021). *Exploration and optimization of building design solutions using computational design* [Doctoral dissertation, Luleå University of Technology]. DiVA Portal. urn:nbn:se:ltu:diva-83636

Ma, W., Wang, X., Wang, J., Xiang, X., & Sun, J. (2021). Generative design in building information modelling (BIM): Approaches and requirements. *Sensors*, *21*(16), 5439. https://doi.org/10.3390/s21165439

Saadi, J. (2024). Generative design tools: Implications on design process, designer behaviour, and design outcomes. [Doctoral dissertation, Massachusetts Institute of Technology]. MIT DSpace. https://hdl.handle.net/1721.1/153664

Sandberg, M. (2015). Towards a knowledge-based engineering methodology for construction. *ICCREM 2015*, 1–8. https://doi.org/10.1061/9780784479377.001

Samuelson, O., and Stehn, L. (2023). Digital transformation in construction – a review. *ITcon*, *28*, 385–404. https://doi.org/10.36680/j.itcon.2023.020

Sergeeva, E.V. (2020). The importance of mathematics for future architects and civil engineers. *IOP Conference Series: Materials Science and Engineering*, *753*(5), 052024. https://doi.org/10.1088/1757-899X/753/5/052024

Schwaber, K. (1997). Scrum development process. In *Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings*, Austin, Texas, 117–134. Springer London. https://doi.org/10.36680/j.itcon.2023.02010.1007/978-1-4471-0947-1_11

Skarka, W. (2007). Application of MOKA methodology in generative model creation using CATIA. *Engineering Applications of Artificial Intelligence*, *20*(5), 677–690. https://doi.org/10.1016/j.engappai.2006.11.019

Stokes, M. (2001). Managing engineering knowledge – MOKA: Methodology for knowledge-based engineering. ASME Press.

Stoyanova, M. (2020). Good practices and recommendations for success in construction digitalization. *TEM Journal*, *9*(1), 42–47. https://doi.org/10.18421/TEM91-07

Turk Ž. (1991). Integration of existing programs using frames. *CIB Seminar: Computer Integrated Future*, 16–17 September, Eindhoven, Netherlands.

van der Velden, C., Bil, C., & Xu, X. (2012). Adaptable methodology for automation application development, *Advanced Engineering Informatics*, *26*(2), 231–250. https://doi.org/10.1016/j.aei.2012.02.007

Verhagen, W.J.C., Bermell-Garcia, P., van Dijk, R.E., & Curran, R. (2012). A critical review of knowledge-based engineering: An identification of research challenges, *Advanced Engineering Informatics*, *26*(1), 5–15. https://doi.org/10.1016/j.aei.2011.06.004

Woodfield, S.N., Dunsmore, H.E., & Shen, Y.Y. (1981). The effect of modularization and comments on program comprehension. *Proceedings of the International Conference on Software Engineering*, 215–223.

Zarei, Y. (2012). *The challenges of parametric design in architecture today: Mapping the design practice* [Doctoral dissertation, The University of Manchester]. The University of Manchester Research Explorer. https://pure.manchester.ac.uk/ws/portalfiles/portal/54523431/FULL_TEXT.pdf

Zhang, B., Mo, Y., & Li, B. (2025). Web-based computational design tools for architectural design studio: Enhancing pedagogical framework. *Nexus Network Journal*, *27*, 663–680. https://doi.org/10.1007/s00004-025-00826-y