

Technische Universität München

TUM School of Engineering and Design

Lehrstuhl für Computergestützte Modellierung und Simulation

A BIM-based Change Management Approach to Improve Decision-making in Building Design Revision

Master Thesis

for the Master of Science Degree in Civil Engineering

Autor: Lingyun Yan

Matriculation Number:



Supervised by: Prof. Dr.-Ing. André Borrmann

M. Sc. Jiabin Wu

Chair of Computational Modeling and Simulation

Date of Issue: 02. May 2024

Date of Submission: 30. September 2024

Acknowledgement

This thesis is a tribute to all the individuals who have made my academic journey at TUM over the past two years so meaningful. First and foremost, I express my deepest gratitude to the Chair of Computational Modeling and Simulation at TUM, led by Herr Prof. Dr. Ing. André Borrmann, for the invaluable support and guidance throughout my research. I am also profoundly grateful to my supervisor, Mr. Jiabin Wu, for his unwavering dedication, insightful discussions, and constant support from the very beginning. Your prompt responses and excellent suggestions were key to the completion of this thesis.

Finally, my heartfelt appreciation goes to my parents, family, and friends for their unwavering support throughout this journey. Thank you for always believing in me and standing by my side.

Abstract

Building design revisions are inevitable due to the reliance on experience and engineering knowledge throughout the design process. Additionally, Building Information Modeling (BIM)-based design is rich in information thanks to advanced technologies, incorporating not only geometric but also topological and semantic data. This leads to a significantly higher number of dependencies between building components. While this intricate web of dependencies enables a more accurate and detailed representation of a building, it also makes design revisions considerably more challenging. Maintaining the integrity of the dependency network is crucial to ensure the building can function as intended without losing key functionalities. As a result, initial changes or revisions can propagate through these dependencies, affecting other parts of a design. Change Propagation (CP) processes are traditionally handled manually, which is often labor-intensive and time-consuming. To address this, the thesis proposes a BIM-based Change Management (CM) framework that automates CP during building design revisions. As a prototype of this framework, a Design Revision Manager (DRM) was developed. The DRM aids decision-making by analyzing change propagation processes before design operations are executed. Furthermore, its versioning and search capabilities allow for tracking the current state of a design and quickly querying information about existing dependencies between components. Tested in real building design scenarios, the DRM prototype demonstrated that having access to component dependencies, CP results, and past revision documentation significantly enhances the decision-making process in building design revisions.

Keywords: Change Propagation (CP), Change Management System, Building Information Modeling (BIM), Building Design Revision, BIM-based Building Design, Engineering Database

Zusammenfassung

Gebäudedesign-Änderungen sind unvermeidlich, da Designprozesse stark auf Erfahrung und ingenieurtechnischem Wissen basieren. Darüber hinaus ist das auf Building Information Modeling (BIM) basierende Gebäudedesign aufgrund fortschrittlicher Technologien sehr informationsreich. Das bedeutet, dass nicht nur geometrische, sondern auch topologische und semantische Informationen in BIM-Modellen enthalten sein können, was zu einer deutlich größeren Anzahl von Abhängigkeiten zwischen den verschiedenen Bauteilen führt. Während ein solch komplexes Abhängigkeitsnetzwerk eine präzisere und genauere Darstellung eines Gebäudes ermöglicht, werden Designänderungen dadurch erheblich erschwert. Es ist notwendig, die Integrität des Abhängigkeitsnetzwerks aufrechtzuerhalten, um sicherzustellen, dass das Gebäude als Ganzes funktioniert, ohne dass gewünschte Funktionen verloren gehen. Folglich können anfängliche Änderungen oder Überarbeitungen durch diese Abhängigkeiten hindurchgreifen und andere Teile des Designs beeinflussen. Solche Prozesse der Änderungsweiterleitung (Change Propagation, CP) werden üblicherweise manuell bearbeitet, was oft mühsam und zeitaufwendig ist. In dieser Arbeit wird daher ein BIM-basiertes Change Management (CM)-Framework vorgeschlagen, das die CP-Prozesse bei Designänderungen automatisch unterstützt. Auf Basis dieses Frameworks wurde ein Design Revision Manager (DRM) entwickelt. Der DRM trägt zur Verbesserung der Entscheidungsfindung bei Designänderungen bei, indem er die Prozesse der Änderungsweiterleitung vor der Ausführung von Designoperationen analysiert. Darüber hinaus ermöglichen die Versionierungs- und Suchsysteme des DRM, den aktuellen Stand eines Designprodukts nachzuverfolgen und schnell Informationen über die aktuellen Abhängigkeiten zwischen Bauteilen abzufragen. Der Prototyp wurde in realen Gebäudedesign-Szenarien getestet und zeigte, dass der Zugriff auf die Abhängigkeiten zwischen den Bauteilen, die Ergebnisse der Änderungsweiterleitungsprozesse und die Dokumentation vergangener Designänderungen im Vorfeld maßgeblich zur Entscheidungsfindung beitragen.

Schlüsselwörter: Änderungsweiterleitung (CP), Änderungsmanagementsystem (CM), Building Information Modeling (BIM), Gebäudedesign-Überarbeitung, BIM-basiertes Gebäudedesign, Ingenieurdatenbank

Table of Contents

List of Abbreviations	VIII	
List of Tables	IX	
List of Figures	X	
1	Introduction	14
1.1	Research Scope	16
1.2	Thesis Structure	16
2	Related Work	18
2.1	Building Information Modeling	18
2.2	Building Design Process	20
2.3	Engineering Change	21
2.4	Change Propagation	23
2.5	Change Management	28
3	Methodology	31
3.1	Research Gap	31
3.2	Research Method	31
3.3	Topological Constraints	33
3.4	Operations for Building Design Revisions	35
3.5	Change Propagation Mechanism	36
3.5.1	Significance Matrix	36
3.5.2	Change Propagation Hierarchy	38
3.6	Database Schema	40
4	Prototype Implementation	45
4.1	Test Data	45
4.2	Implementation of Operations	46
4.2.1	User Input	47
4.2.2	Structural Wall Deletion	47
4.2.3	Structural Column Deletion	50

4.2.4	Stair Deletion	52
4.2.5	Structural Wall Addition.....	52
4.2.6	Structural Column Addition	55
4.2.7	Stair Addition	57
4.3	Implementation of Database Schema	60
4.4	Design Revision Manager (DRM)	64
5	Results and Discussion	67
5.1	Test Results of Design Revision Operations.....	67
5.1.1	Initial Test Model.....	67
5.1.2	Structural Wall Deletion	68
5.1.3	Structural Column Deletion	70
5.1.4	Stair Deletion & Addition	71
5.2	Test Results of Topological Constraint-based Commands	74
5.3	Key Findings	77
6	Conclusion	78
6.1	Contribution	79
6.2	Limitation	79
6.3	Future Work	80
	Bibliography	81

List of Abbreviations

BIM	Building Information Modelling
AEC	Architecture, Engineering, Construction
LOD	Level of Detail
ECs	Engineering Changes
CP	Change Propagation
CM	Change Management
CPA	Change Propagation Analysis
API	Application Programming Interface
GUI	Graphical User Interface
GUID	Globally Unique Identifiers
XAML	Extensible Application Markup Language
CAD	Computer-Aided Design
DSM	Design Structure Matrix
DRM	Design Revision Manager
SDK	Software Development Kit

List of Tables

Table 1: Topological constraints in BIM-based building design considered in this study 34

Table 2: Operations for building design revision 36

Table 3: Different constraints for different operations (based on topological constraints and the significance matrix) 37

Table 4: The number of building components (initial state of the test model) 67

Table 5: The number of building components (3rd version) 74

Table 6: The number of building components (5th version) 75

List of Figures

Figure 1: Building Information Modeling shifts planning effort and design decisions to earlier phases (MacLeamy 2004)	19
Figure 2: A system of reference planes and lines forms the basis for setting up a comprehensive parametric building design (figure from lecture material by Chuck Eastman, GeorgiaTech).....	20
Figure 3: The model of a generic change process from Jarratt et al (2004)	22
Figure 4: Ontology of design changes in BIM-based building design (Pilehchian et al. 2015).....	23
Figure 5: Change propagation as a cone. Hollow circles are constraints, with their state inside. Black small circles are entities. The constraint referenced by the arrow is infeasible, requiring iteration (Eastman et al. 1997)	24
Figure 6: Use cases for Change Propagation Analysis (Brahma et al. 2023)	25
Figure 7: DSM of a diesel engine used for CPA model input, showing mechanical linkages between subsystems from the viewpoints of four engineers. The position of the colored boxes within the cells refers to a particular engineer's marks (Brahma et al. 2023)	26
Figure 8: Dependency matrix by Pilehchian et al. (2015)	27
Figure 9: Core technologies of the knowledge-supported system developed by Ma et al. (2005)	28
Figure 10: Data loss in file conversion between different formats from Oh et al. (2015)	30
Figure 11: Proposed method	33
Figure 12: Represent dependencies between building components with low-level and topological constraints.....	35
Figure 13: Significance Matrix (indicates the existence of dependencies between two building components and controls the extent of change propagation)	37
Figure 14: Change Propagation Hierarchy	39
Figure 15: Example of Change Propagation Hierarchy	40
Figure 16: ER model of structural wall.....	41

Figure 17: ER model of architectural wall	42
Figure 18: ER model of architectural column.....	42
Figure 19: ER model of structural column.....	42
Figure 20: ER model of room.....	43
Figure 21: ER model of door.....	43
Figure 22: ER model of stair	43
Figure 23: ER model of revision	44
Figure 24: Floor plans of the test model (the floor plan on the left is the ground floor and the one on the right is the first floor)	45
Figure 25: Test model (3D view).....	46
Figure 26: Input file	47
Figure 27: Read critical conditions from input file	47
Figure 28: Constraints for Structural Wall Deletion.....	48
Figure 29: Structural Wall Deletion	48
Figure 30: Workflow of the method IsStandAloneStructuralWall	49
Figure 31: BoundingBoxXYZ class in Revit API	49
Figure 32: LocationCurve of wall	50
Figure 33: Constraints for Structural Column Deletion	51
Figure 34: Structural Column Deletion.....	51
Figure 35: Workflow of the method IsStandAloneStructuralColumn	51
Figure 36: Constraints for Stair Deletion.....	52
Figure 37: Stair Deletion	52
Figure 38: Constraints for Structural Wall Addition	53
Figure 39: Structural Wall Addition	54
Figure 40: Autodesk.Revit.DB.Create method.....	54
Figure 41: Workflow of the method FindOverlappingWall.....	55
Figure 42: Workflow of the method IsRoomAccessible.....	55
Figure 43: Constraints for Structural Column Addition.....	56
Figure 44: Structural Column Addition	56
Figure 45: Autodesk.Revit.Creation.Document.NewFamilyInstance	57

Figure 46: Workflow of the method FindOverlappingColumn	57
Figure 47: The implementation of the method NearbyColumns.....	57
Figure 48: Constraints for Stair Addition	58
Figure 49: Stair Addition	59
Figure 50: Methods for checking the accessibility of stairs	59
Figure 51: Implementation of the method CreateSingleStraightRunStair	59
Figure 52: Documents and collections in MongoDB	60
Figure 53: Data model of wall	61
Figure 54: Data model of column.....	61
Figure 55: Data model of door	62
Figure 56: Check the accessibility of stairs.....	62
Figure 57: Versioning system for design revision	63
Figure 58: Using multiple versioning properties simultaneously in queries.....	63
Figure 59: Track design revisions throughout the design process.....	64
Figure 60: Design Revision Manager (design operations)	64
Figure 61: Design Revision Manager (topological constraint-based commands)	65
Figure 62: Initial state (version 0) of test model	68
Figure 63: First operation – Structural Wall Deletion (the deleted wall resides in Level 1, the figure on the left is the original status, while the one on the right is the status after the DELETE operation)	68
Figure 64: 1 st version of test model (the left figure presents that the operation is documented in the Revisions collection, while the right one shows the updates of the deleted wall's properties)	69
Figure 65: Design Revision Manager – Structural Wall Deletion (pre-constraints are the dependencies of building components before the operation is performed).....	70
Figure 66: Second operation – Structural Column Deletion (the to-be-deleted structural column from different views, namely Level 1 in the left figure and Level 2 in the right one)	70
Figure 67: Third operation – Stair Deletion (the figure on the left is the original status, containing the to-be-deleted stair with an opening on top, the right one is the updated status after the DELETE operation)	71

Figure 68: 2nd version of test model (the left figure presents that the operation is documented in the Revisions collection, while the right one shows the updates of the deleted stair’s properties) 71

Figure 69: Fourth operation – Stair Addition (the left figure shows an intended insertion of a stair at a desired location by picking two points, the right one shows the updated status after adding the stair) 72

Figure 70: 3rd version of test model (the left figure presents that the operation is documented in the Revisions collection, while the right one shows the updates of the inserted stair’s properties) 73

Figure 71: Design Revision Manager – Stair Addition (change propagation results) 73

Figure 72: Fifth operation – Stair Addition 74

Figure 73: The latest state of test model (5th version of test model on the left and the corresponding database on the right)..... 75

Figure 74: Test results of topological constraint-based commands 76

Figure 75: Find the problematic dependency with GUID (the red marked GUID refers to the inaccessible stair found in Fig. 73, the blue marked GUID is the structural wall or column that blocks the exit or entrance of the stair) 76

1 Introduction

Building Information Modeling (BIM) is generally defined as a digital representation of the physical and functional characteristics of a built facility (NIBS 2015). Two key advantages of BIM-based building design are its parametric design capabilities and interoperability across different software platforms. BIM models typically contain extensive information that's relevant for civil engineering projects, including parametric representations of building components, non-physical objects, and even the structure of the project itself (Borrmann et al. 2018). In addition, the enrichment of information proves valuable for many downstream applications, such as structural simulations and calculations.

Building design processes are typically divided into three phases: pre-design, schematic design, detailed design, and construction (Borrmann et al. 2018). Implementing design revisions in the earlier stages requires significantly less effort than changes made in later phases. Given that building design heavily relies on engineering expertise (Cross 2021), revisions are often unavoidable. However, this challenge can be mitigated through BIM-based planning processes. For instance, clashes between different discipline-specific BIM models can be identified early by regularly federating these models. In this way, BIM-based design reduces risks and uncertainties, improving overall efficiency and productivity. By integrating BIM-based planning throughout all phases, design revisions can be managed more effectively, minimizing disruptions and ensuring a smoother design process.

Building design revisions are essentially Engineering Changes (ECs), defined as modifications made to released parts, drawings, or software (Jarratt et al. 2005). In the Architecture, Engineering, and Construction (AEC) industry, changes can occur throughout the entire lifecycle of a built facility. These revisions can stem from various factors, including coordination defects, updated requirements from stakeholders, or designer omissions (Cox et al. 1999). Although changes are easier to implement in earlier design stages, building design revisions generally require significant effort due to the complex process of Change Propagation (CP), where alterations can affect other parts of a design (Clarkson et al. 2004; Giffin et al. 2009). Furthermore, BIM-based designs often involve large-scale projects with thousands of components and detailed

semantic information. This means that not only the initial change but also its potential impact on related components must be carefully considered. Thoroughly analyzing the dependencies between building components is crucial, as more complex designs result in a much more sophisticated network of dependencies and therefore lead to significantly greater challenges in implementing design revisions.

Dependencies between building components are critical for BIM-based building design. Establishing appropriate dependencies in advance can significantly reduce laborious and time-consuming tasks. For example, two walls on different floors can be automatically aligned vertically by pre-establishing a relationship, such as applying a constraint to both components. In essence, a dependency between two building components means that they are mutually constrained. With the advancement of BIM technologies, a growing number of components can be stored within BIM models, creating the potential to establish more complex dependencies between them. However, this potential remains underutilized, as most BIM authoring tools primarily implement only low-level constraints. For example, while geometric constraints—such as ensuring the front and back sides of a wall share the same geometry—are standard features in most BIM authoring tools, more detailed information, such as the number of structural components linked to a specific element, is often unavailable.

Given the complexity of building design processes and the inevitability of design revisions, effective Change Management (CM) systems are urgently needed. Change management in Computer-Aided Design (CAD) focuses on representing design products as they evolve throughout the design process by storing data and organizing their dependency networks within databases. From a technical standpoint, a change management system can identify the path of Change Propagation (CP) and subsequently limit the scope within which CP can occur. Therefore, an effective change management approach can significantly reduce the negative impacts associated with building design revisions.

In the AEC industry, change management plays a crucial role in facilitating collaborative and concurrent design among multiple disciplines. T. Jeng and C. Eastman (1998) developed a database architecture to support collaboration by monitoring data consistency and CP. Ma et al. (2003) proposed an approach to investigate the effects of ECs by generating alternative scenarios of CP resulting from the same initial change

and evaluating their respective impacts. With the ability to assess CP, change management can enhance the optimization processes for resolving clashes. Additionally, change management systems typically perform Change Propagation Analysis (CPA) of proposed design revisions before any modifications are made, significantly aiding decision-making processes. This prior evaluation of CP also enables design automation to a certain extent, allowing multiple changes resulting from CP to be automatically applied to a design product, thus improving design efficiency. However, the CP capabilities within individual BIM authoring tools remain limited to low-level constraints, as mentioned earlier. Consequently, each design revision prompted by project-specific factors, such as updated requirements from stakeholders, necessitates a manual evaluation of its impacts before the revision can be implemented.

1.1 Research Scope

This research aims to develop a change management system that enhances decision-making for building design revisions within BIM tools. Specifically, this thesis seeks to answer the following questions:

- How can a change management system for BIM-based building design improve decision-making regarding design revisions?
- How can we identify the dependencies among building components, and how can these dependencies be integrated into the design operations in BIM authoring tools?

1.2 Thesis Structure

The rest of the thesis is structured as follows:

- Chapter 2, Related Work, conducts an in-depth literature review on topics such as Building Information Modeling (BIM), change management, Change Propagation Analysis (CPA) and their applications. This chapter establishes a solid theoretical background for the study.
- Chapter 3, Methodology, presents the proposed approach to address the research questions. The chapter proposes a framework of change management in BIM-based building design, supporting semi-automatic design revisions.
- Chapter 4, Prototype Implementation, illustrates the steps for the implementation of the proposed framework by using selected tools. In addition, a test case is designed to evaluate the functionalities of the prototype.

- Chapter 5, Results & Discussion, presents the test results and elaborates the key findings and limitations of the research.
- Chapter 6, Conclusion and Outlook, concludes the study by answering the proposed research questions and outlines recommendations for further research based on the results of this study.

2 Related Work

This chapter begins by reviewing the fundamental concepts of BIM and the advantages of BIM-based planning processes in building design, emphasizing the importance of shifting design decisions and changes to earlier phases. Next, it elaborates on the characteristics of building design processes by explaining why design changes occur and how they manifest. Using the concept of Change Propagation (CP) as a cone (Eastman et al. 1997), the chapter presents the essence of CP and highlights two critical issues that must be addressed to maintain the desired functionalities of a design product. Additionally, based on the generic change process proposed by Jarratt et al. (2004), the workflow of CPA is detailed, including the representation of input data, data population based on representation schemes, and various effective approaches for analyzing these representations. Finally, the chapter thoroughly reviews the application of CPA in supporting change management.

2.1 Building Information Modeling

The National Institute of Building Sciences (NIBS) defines Building Information Modeling (BIM) as a digital representation of a facility's physical and functional characteristics (NIBS 2015). BIM is an evolution of the product model concept (Eastman 1999 & Borrmann et al. 2009). Its two technological foundations are parametric design and interoperability among multiple BIM software products, facilitating data exchange between different vendors (Eastman 2011). In the AEC industry, BIM typically refers to the process of creating a building facility model and managing it throughout its lifecycle, from conceptual design to deconstruction (Borrmann et al. 2018). A BIM model consists of parametric objects representing building components at a defined Level of Detail (LOD), as well as non-physical objects, such as zones, spaces, or the project structure itself (Borrmann et al. 2018). These objects encompass both geometric and non-geometric properties, including functional, semantic, and topological information (Eastman 2011 & Wong et al. 2010). For example, costs can serve as a functional attribute, while information regarding intersections and connectivity represents semantic properties, and an object's perpendicularity or adjacency conveys topological information.

Although the development of 3D CAD systems began in the 1970s, traditional 2D CAD systems remained the dominant design approach in the AEC industry for many years

(Volk et al. 2014). The application of BIM technologies in pilot projects to support building design emerged even later, in the early 2000s (Penttilä et al. 2007). The conventional building design process is often laborious, time-consuming, and prone to errors, such as the manual evaluation of consistency among various 2D technical drawings and the need to re-enter information extracted from these drawings for downstream applications, including calculations and simulations (Borrmann et al. 2018).

With the adoption of BIM technologies, numerous benefits throughout a built facility's entire lifecycle—encompassing pre-construction, design, construction and fabrication, and post-construction—become achievable (Eastman 2011). In particular, during the design phase, repetitive and labor-intensive tasks can be automated, leading to increased productivity. Such tasks may include updating the BIM model whenever design changes occur, generating accurate 2D drawings directly from the BIM model (Kim et al. 2016), and federating or integrating BIM models from multiple design disciplines (Beach et al. 2017).

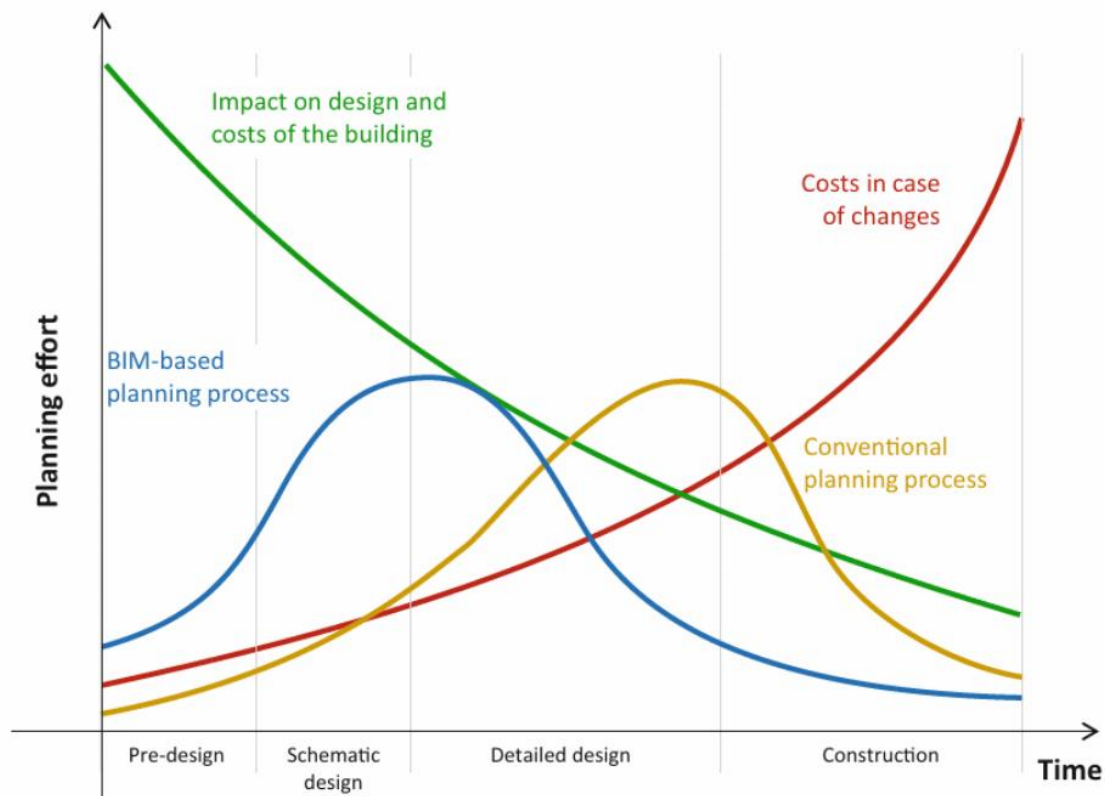


Figure 1: Building Information Modeling shifts planning effort and design decisions to earlier phases (MacLeamy 2004)

Prior to the construction phase, building design can be divided into three phases: pre-design, schematic design, and detailed design. Implementing potential design changes in the earlier phases requires significantly less effort since changes initiated late in the

process impact more stakeholders (Jarratt et al. 2011). However, conventional design workflows often commit changes during the detailed design phase and frequently during the construction phase, leading to higher costs. By employing a modern BIM-based planning workflow, design decisions and changes can be shifted to the earlier phases (Figure 1), resulting in significantly reduced costs and improved building performance (Borrmann et al. 2018).

2.2 Building Design Process

The essence of engineering design lies in identifying and implementing solutions to engineering problems. Throughout the design process, practitioners rely heavily on prior experience, general guidelines, and "rules of thumb" (Cross 2021). However, these heuristic methods do not guarantee the success of the design product. Among the various types of engineering design tasks, building design is particularly complex due to its large scale and the necessity for coordination among multiple design disciplines. Changes made in the later stages of building design are often cost-intensive, and certain modifications may not be feasible due to existing constraints (Keller et al. 2005). Typically, the building design process seeks to balance various competing parameters, each subject to specific constraints (Machairas et al. 2014).

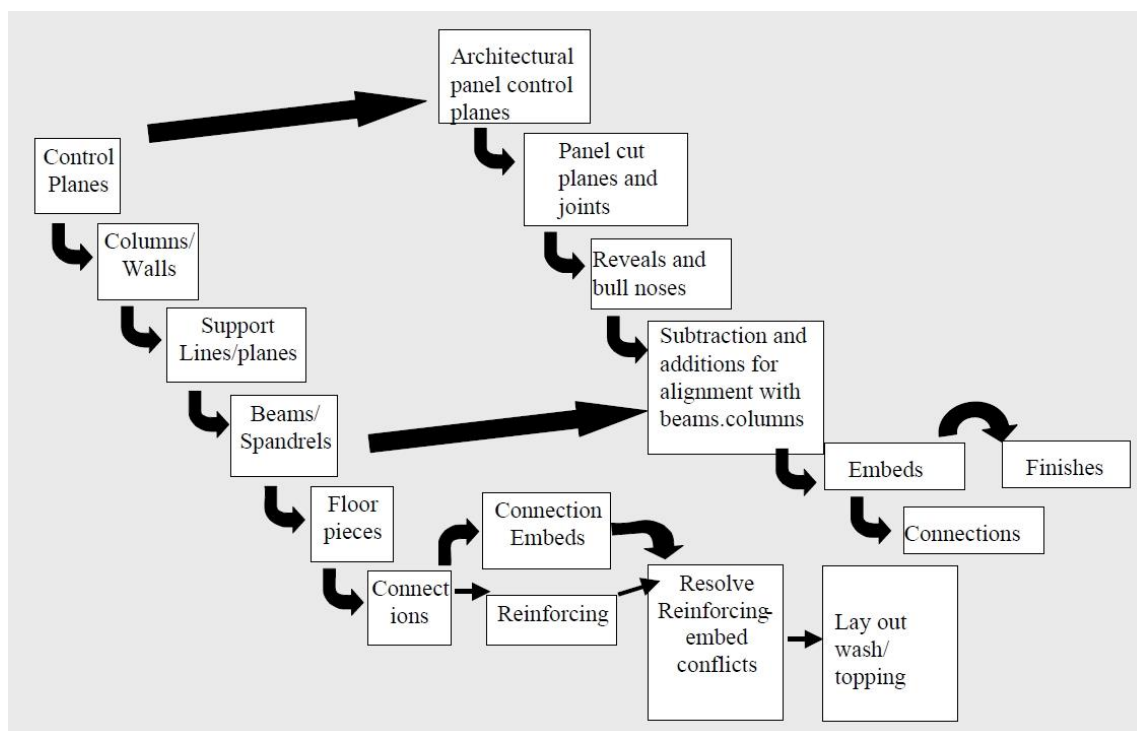


Figure 2: A system of reference planes and lines forms the basis for setting up a comprehensive parametric building design (figure from lecture material by Chuck Eastman, GeorgiaTech)

In addition to the designer, numerous stakeholders participate in building design processes (Borrmann et al. 2018). Design decisions are significantly influenced by these stakeholders, whose opinions can differ and evolve over time. Sten de Wit et al. (2002) noted that the evolution of a building design consists of a series of design decisions, each informed by insights from various domain experts. From this perspective, multiple potential re-evaluations and design revisions become necessary to enhance the design (Koh et al. 2012), rectify design flaws, or adapt to updated requirements from stakeholders (Ahmed et al. 2012).

2.3 Engineering Change

Engineering Change (EC) is referred to as an alteration made to parts, drawings or software that have already been released during the product design process and life cycle, without considering the scale or the type of change (Jarratt et al. 2011 & Jarratt et al. 2005). In the AEC industry, changes are described as modifications, differences, subtractions, additions, or exclusions of work (Antill, 1990). Potential changes can occur throughout the entire design process and have an impact on the product, e.g. BIM model. Typical causes of change in construction projects could be designer's omission and coordination defects in tender documents, updated employer's requirements, or new information regarding site conditions (Cox et al. 1999). Engineering changes that can trigger a chain of changes, are categorized as either emergent (originating from the product itself, e.g. errors) or being initiated, e.g. by stakeholders' new requirements (Eckert et al. 2004). Jarratt et al. (2005) proposed a model to describe and formalize a generic change process (Figure 3), which comprises three phases. The before-approval phase includes raising a change request and identifying possible solutions and their risk assessment. The second stage is to select and approve one of the possible solutions. Thereafter, the implementation of the selected solution and the review of a particular change process are incorporated into the final phase.

Based on the observation of numerous examples of building design changes from case studies, Pilehchian et al. (2015) developed the following BIM-based ontology of design changes, so that changes can be structurally and hierarchically organized, and their impacts can be kept traceable, as illustrated in Figure 4. Building design changes are categorized into three classes. The object-oriented class represents changes in terms of geometry, position, or specifications of building components, while the adaptation-

oriented characteristics are relevant for automatic change propagation, comprising various dependencies between components and aiming at continuous data modification. The third class is integration-oriented and, therefore, is responsible for merging data from multiple sub-models.

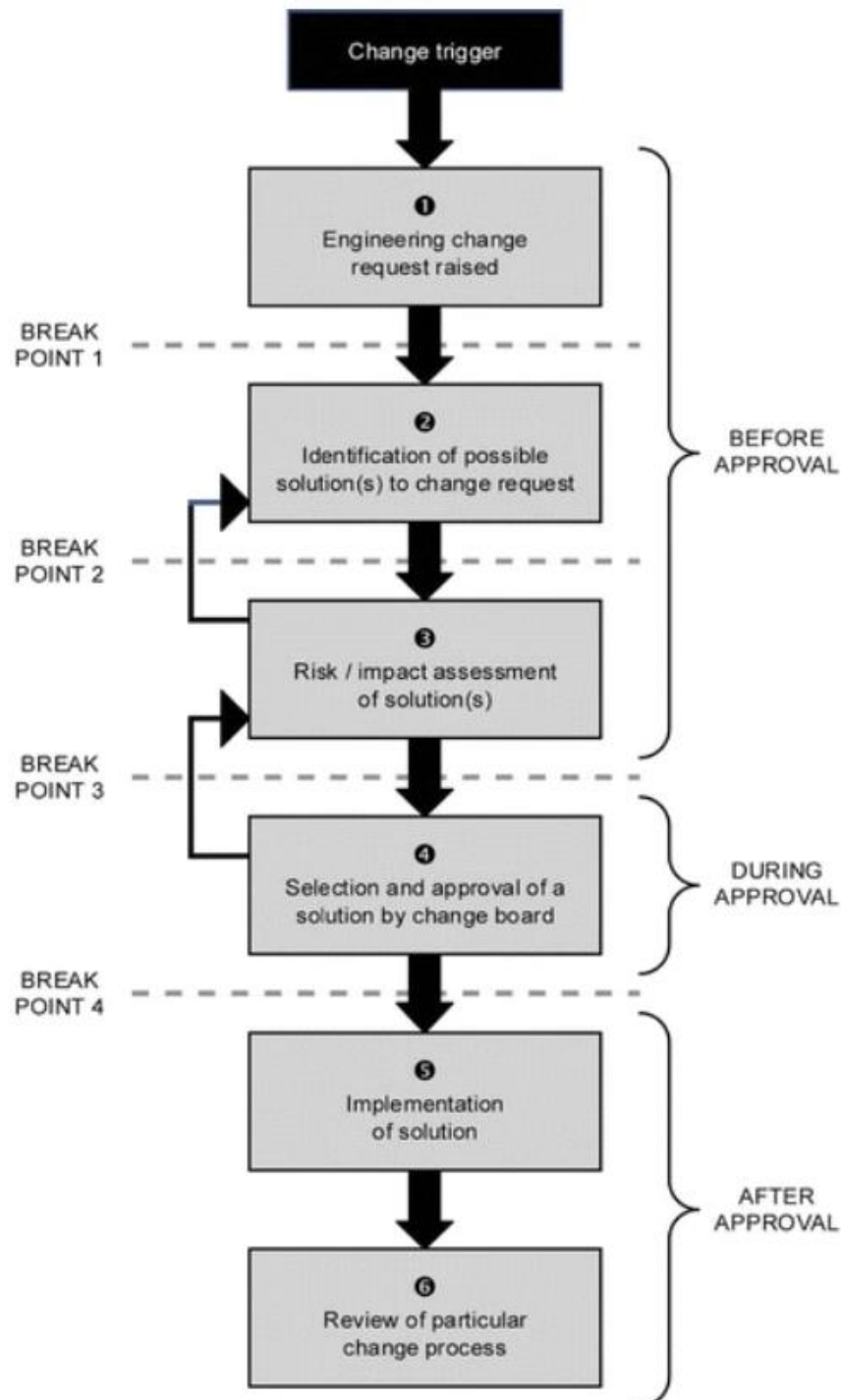


Figure 3: The model of a generic change process from Jarratt et al (2004)

Classes & Sub-classes	Facets: Description/Example
Object-oriented	
Change type	
Addition (ADD)	<i>Creating a new component</i>
Deletion (DEL)	<i>Deleting an existing component</i>
Modification (MOD)	<i>Modification in attributes of an existing component</i>
Recreate (REC)	<i>Deleting a component then adding a new one with similar attributes</i>
Merge (MRG)	<i>Combining two or more components to create a new component</i>
Split (SPL)	<i>Dividing a component into two or more components</i>
Changed component attributes	
Geometry	Shape (SHP): <i>cubic, cylindrical, rectangular, plate</i> Dimensions (DIM): <i>shape, length, width, thickness, diameter, slope</i>
Position	Coordinates (CRD): <i>X, Y, Z</i> Orientation (ORN): <i>Rx, Ry, Rz</i>
Specification	Material (MAT): <i>concrete, mild steel, galvanized steel</i> Elements (ELM): <i>stud, rebar: size, shape, arrangement</i> Semantic properties (PRP): <i>fire-rating, acoustic, water proof</i>
Adaptation oriented	
Dependencies between components	
Spatial dependencies	Connected to (CNT): <i>column and floors, main and secondary ducts</i> Adjacent to (ADT): <i>duct and adjacent pipes, duct and ceiling</i> Supported by (SPB): <i>duct and steel hangers</i> Surrounded by (SRB): <i>duct and false ceiling/plenum area</i>
Analytical dependencies	Structural (STR): <i>sleeves size and arrangement of rebar</i> Architectural (ARC): <i>room functionality and exposed duct</i> Mechanical (MEC): <i>size and location of air supply duct</i> Electrical (ELC): <i>size of cable tray and motor power</i> Operational (OPR): <i>clearance around a pipe</i>
Propagation of changes	
Level of propagation	Extensive: <i>substantial effects on many components</i> Regional: <i>affect several adjacent components</i> Local: <i>minimal effect on other components</i>
Type of dependencies	Intra-model: <i>between components with same LOD</i> Inter-model: <i>between components with different LOD</i>
Integration oriented	
Change timing	
Conceptual design	<i>During early decision making about the primary design aspects</i>
Basic design	<i>During early stages of the design but prior to the full extended design</i>
Detail design	<i>During the detailed design but prior to any procurement/construction</i>
Procurement	<i>After purchase order but prior to fabrication</i>
Fabrication	<i>After fabrication but prior to erection</i>
Construction	<i>After commence of construction</i>
Change impact	
Cost	<i>The impacts of the change on the project cost</i>
Time schedule	<i>The impacts of the change on the project time schedule</i>
Client's objective	<i>The impacts of the change on the client's objectives</i>

Figure 4: Ontology of design changes in BIM-based building design (Pilehchian et al. 2015)

2.4 Change Propagation

ECs that occur in one aspect of the design can result in changes to other parts (Clarkson et al. 2004 & Giffin et al. 2009), since all parts of the design must work together as a whole to achieve the desired functionalities of the product. For example, in building design, a change in a building's overall height typically results in changes in floor-to-floor height and some other vertical dimensions. Similarly, Mirshekarlou (2012) described this phenomenon as a cumulative or ripple effect of a change. During the design process, components of complex products that were considered finished are still subject to design changes (Huang et al. 1999). Due to these knock-on changes (Keller et al. 2005), Change Propagation (CP) is one key aspect in the building design process.

To be more specific, CP is initially defined by the changed attribute values, but thereafter proceeds by invalidating constraints that accessed the modified data or that relied on a constraint that is now invalidated (Eastman et al. 1997). Constraints here are typically referred to as the integrity rules on each part of the design, e.g., dependencies to other parts, requirements derived from design codes or directly from the involved stakeholders etc. Keller et al. (2005) classified CP into two categories: direct propagation, which appears in components directly connected to the modified component, and indirect propagation.

Two essential problems regarding CP were proposed by Randy et al. (1990). The first is to disambiguate the path of CP since different sequences in which the propagation takes place result in different design products. The second is to limit the scope of propagation. Similarly, Eastman et al. (1997) described and formalized CP as a cone. The path of propagation, along with a set of pre-defined transaction protocols for identifying what data and constraints should be invalidated by which kind of initial changes, is graphically illustrated (Figure 5). Namely, each design change propagates along a set of components linked by constraints, diversely into multiple directions. The complexity of the cone increases significantly as the number of components and constraints in the design becomes larger. Such a complicated network of CP should be handled properly so that a design product's semantic integrity, which is a term used in the database field to designate the constraints that data must satisfy to have a meaningful correspondence with reality (Eastman et al. 1997), can be maintained.

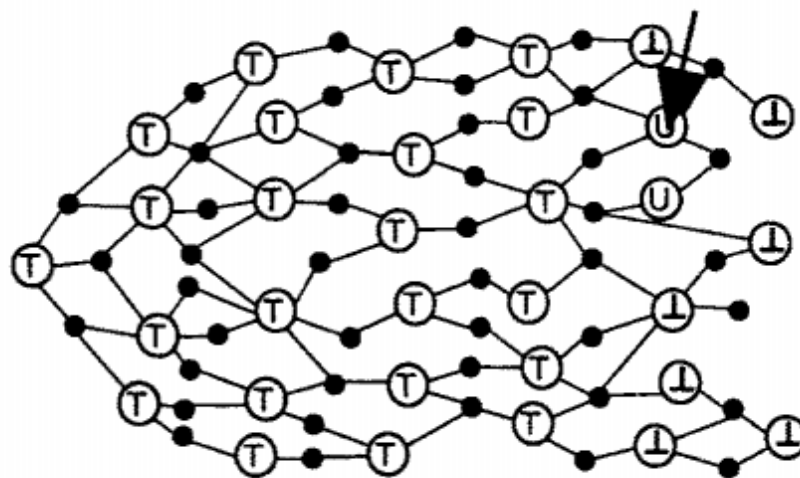


Figure 5: Change propagation as a cone. Hollow circles are constraints, with their state inside. Black small circles are entities. The constraint referenced by the arrow is infeasible, requiring iteration (Eastman et al. 1997)

When a design comprises many parts that are tightly integrated and correlated, and knowledge of the design is distributed among various domain experts, e.g. building design (Ahmad et al. 2013), it's typically difficult to identify possible propagations across disciplinary boundaries (Reddi et al. 2009). For this reason, many researchers proposed models for the purpose of performing Change Propagation Analysis (CPA), which can support the process of change management by generating alternatives for implementing an engineering change, evaluating impacts of a proposed change on various aspects, coordinating change activities, and finally improving the design quality (Brahma et al. 2023), as illustrated in Figure 6. Generating a wide range of possible solutions can improve the likelihood of finding a good one (Pahl et al. 2013). The impact of a proposed change determines whether the initial change should be implemented or not. The workflow of CPA includes representing the input data used for CPA, populating the representations with data, and evaluating the representations (Brahma et al. 2023).

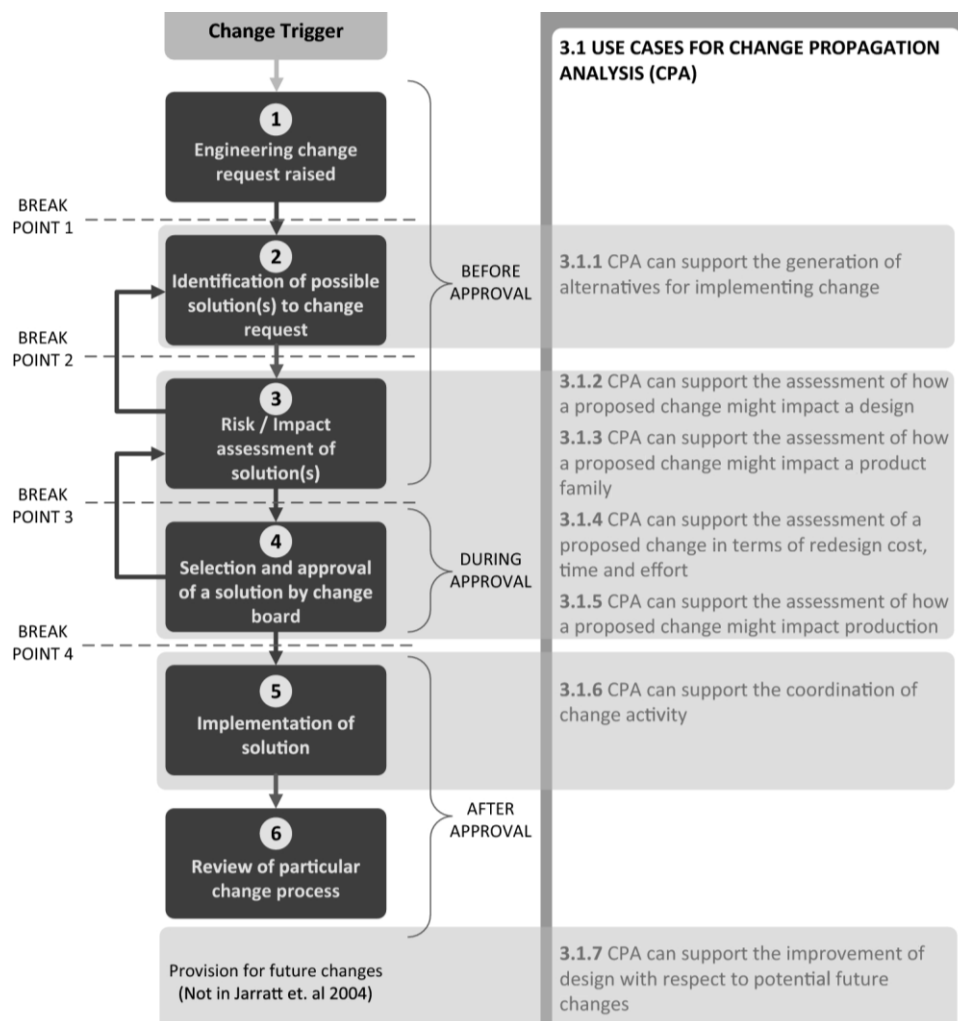


Figure 6: Use cases for Change Propagation Analysis (Brahma et al. 2023)

Design Structure Matrix (DSM), as illustrated in Figure 7, offers a very compact visual representation of data for CPA. It is typically applied either for displaying the relationships between discrete components, through which design changes can propagate, or to represent the likelihood and impact values of changes (Keller et al. 2005). In other words, DSM is a network of dependencies through which change can propagate. Researchers supplemented the schema of DSM by adding extra information into the dependency cells. Ma et al. (2003) included information regarding energy, material, etc., for each dependency, while Rutka et al. (2006) indicated levels and types of changes that are allowed to propagate through each dependency. Lama Adel Saoud et al. (2017) developed an approach to visually represent the process of predicting changes through the integration of BIM with a parameter-based DSM.

Mechanical Links (either static or dynamic)	Cyl Head	Cyl Block	Piston	Conn Rod	Crankshaft	Adapter Plate	Flywheel Housing	Flywheel	Starter Motor	Fan Drive	Sump	Oil Fillers & Breathers	Oil Fillers & Coolers	Crank Pulley	Coolant Pump	Fan & Extension	Alternators & Bracket	Belt Driven Auxiliary	Gear Driven Auxiliary	Balancer	Turbochargers	Intake	Exhaust	Fuel Filters	Starting Aids	Lifting Eyes	
Cyl Head	■	■																									
Cyl Block	■	■	■																								
Piston		■	■	■																							
Conn Rod			■	■	■																						
Crankshaft				■	■	■																					
Adapter Plate					■	■	■																				
Flywheel Housing						■	■	■																			
Flywheel							■	■	■																		
Starter Motor								■	■																		
Fan Drive									■	■																	
Sump										■	■																
Oil Fillers & Breathers											■	■															
Oil Fillers & Coolers												■	■														
Crank Pulley													■	■													
Coolant Pump														■	■												
Fan & Extensions															■	■											
Alternators & Brackets																■	■										
Belt Driven Auxiliary																	■	■									
Gear Driven Auxiliary																		■	■								
Balancer																			■	■							
Turbocharger																				■	■						
Intake																					■	■					
Exhaust																						■	■				
Fuel Filter																							■	■			
Starting Aid																								■	■		
Lifting Eyes																										■	■

Figure 7: DSM of a diesel engine used for CPA model input, showing mechanical linkages between subsystems from the viewpoints of four engineers. The position of the colored boxes within the cells refers to a particular engineer's marks (Brahma et al. 2023)

Input data for CPA can be generated from design analysis, workshops, historical data, etc. In building design, CAD data is mostly available for CPA. An algorithm that acquires constraints between topology faces from CAD models was developed by Yin et al. (2016). The acquired data was then used to create a relationship matrix for CPA.

Masmoudi et al. (2017) discussed the approaches for extracting dependencies between dimensions by adjusting those dimensions in CAD software and how geometric constraints result in changes in other dimensions. Chen et al. (2017) proposed a novel change feature-based approach to predict the impact of the current design changes. The approach detects changed features by comparing CAD files, and the features are compared to past changes for CPA.

There are various approaches for CPA. Reddi and Monn (2009) applied probabilistic methods to evaluate the dependency network. They considered the type of changes and the likelihood of propagation for each dependency in the DSM. Monte Carlo Simulation can be applied to simulate individual changes as they propagate step by step (Wynn et al. 2014 & Li et al. 2012). Matrix operations and calculations over cells are also applicable. For example, Pilehchian et al. (2015) proposed a graph-based approach and the concept of dependency matrix for realizing automated CP in BIM-based project delivery processes. The dependency matrix, as illustrated in Figure 8, is defined to use 0 and 1 to indicate the effect of changes, e.g. a 0 value at the position [1,2] indicates that the change of attribute q of component 1 has no impact on the attribute p of component 2. Various types of interdependencies among design changes were investigated.

$$D = \begin{bmatrix} |D11| & \dots & |D1n| \\ \vdots & |Dij| & \vdots \\ |Dn1| & \dots & |Dnn| \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{bmatrix}$$

$$|Dij| = \begin{bmatrix} d11 & \dots & d1m \\ \vdots & dpq & \vdots \\ dm1 & \dots & dmm \end{bmatrix} d_{pq}; \quad d_{pq} = \begin{cases} 1 : & \text{if change in attribute } p \text{ of component } i \\ & \text{affects attribute } q \text{ of component } j \\ 0 : & \text{if change in attribute } p \text{ of component } i \\ & \text{does not affect attribute } q \text{ of component } j \end{cases}$$

Figure 8: Dependency matrix by Pilehchian et al. (2015)

With the help of CPA, insights regarding possible impacts caused by a proposed change can be analyzed for change management, which typically supports the decision-making process when design changes occur. A knowledge-supported system was proposed by Ma et al. (2003), as illustrated in Figure 9. One part of the system is the

integrated design information model that represents the complete design data and the corresponding relationships. The other part is the intelligent change impact analysis engine, which can generate change plans, quantify the impact of engineering changes, and evaluate alternative change scenarios. Valeh (2017) developed an automated model named BIM-Change for calculating and visualizing ripple effects on the project's duration, that are caused by changes initiated by owners.

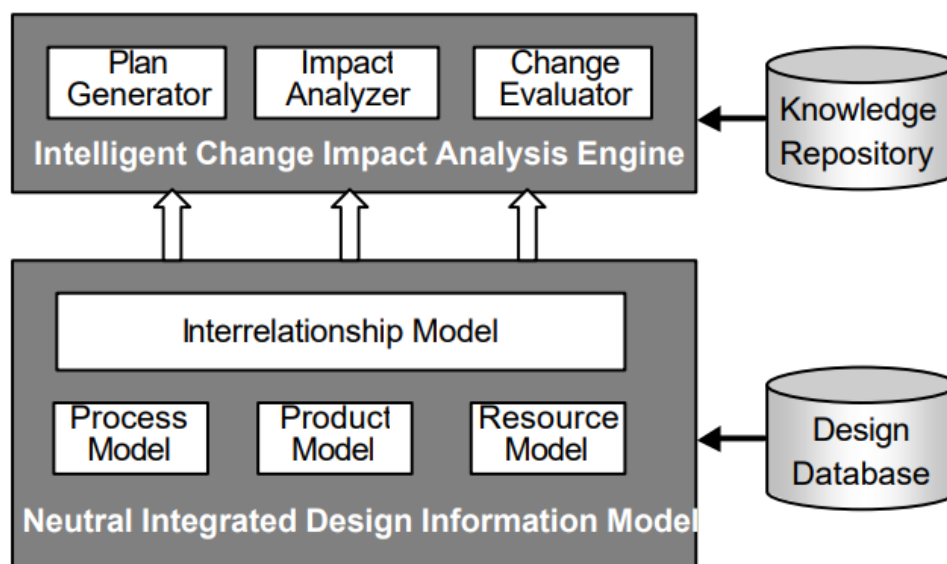


Figure 9: Core technologies of the knowledge-supported system developed by Ma et al. (2005)

2.5 Change Management

Voropajev (1997) defined Change Management (CM) as an integral process related to all project internal and external factors, influencing project changes; to possible change forecast; to identification of already occurred changes; to planning preventive impacts; to coordination of changes across the entire project. CPA is an effective approach for managing changes in engineering design. Digital twins as an emerging information technology provide opportunities for the development of new techniques for managing change in complex projects, such as infrastructure, new energy and resource projects (Whyte and Jennifer et al. 2024).

Change management in terms of computer-aided design is essentially a matter of managing the semantic integrity of the database of a design product when design changes are initiated. For most CAD software, various transaction protocols and integrity rules aiming at maintaining the semantic integrity are embedded and implemented in the software database. Namely, simple CPA is already taken into account in 2D or 3D modelling processes. In the context of building design, most low-level constraints,

e.g. alignment, levels, or direct interference between components (geometric properties), are already included and implemented, e.g. in BIM authoring tools. Because they are applicable for most engineering design tasks.

In BIM-based building design, change management systems are primarily implemented to facilitate concurrent design and enhance collaboration and coordination among multiple design disciplines or software applications. This involves managing interactions and data exchanges between different databases. The reason is the inevitable data loss in file conversion between different formats (Figure 10). A comprehensive product design repository is typically sought to address all relevant aspects of a design within a specific context. Jacobsen et al. (1997) explored information management requirements and established general criteria for collaboration and concurrency control in creative engineering design. While Jeng and Eastman (1998) proposed a new database architecture for design collaboration, emphasizing consistency monitoring, change propagation, and the structure and application of rules to support design processes, the concept of “Engineering Design Knowledge Repository” was proposed by Regli et al. (2000) to aid collaboration and archival processes for distributed design and manufacturing teams by collecting and storing public-domain engineering data. Some BIM authoring tools, such as Revit, have achieved collaborative design. Nevertheless, CAD systems and interoperability between different CAD software often focus on general rules applicable to most systems and design disciplines.

Project-dependent and discipline-dependent topological constraints derived from expert knowledge and good practices are complex and too specific, such as the guarantee of the accessibility of stairs by adding the necessary openings to the relevant slabs. Therefore, they need to be implemented individually for a CAD system in a specific design context. Topological constraints can only be imposed incrementally on the data manually by the designer, since building design is a complicated and cross-disciplinary process involving many stakeholders, whose opinions and decisions on the design vary all the time. It is not feasible to account for all topological constraints and satisfy the requirements of all relevant parties at the outset. Similar to geometric constraints, these constraints can also be implemented and stored in a design database. Szykman et al. (2000) pointed out the necessity and importance of a design repository in engineering design. In addition to storing design information, design repositories are rather designed for retrieval and reuse of design knowledge by using sophisticated methods

that are not available in conventional database management systems, e.g., querying components that satisfy required functions. Moreover, Murdock et al. (1997) attempted to develop a framework in terms of information modeling for supporting the creation of design repositories. The key aspects are the representation of form, function, and behavior of artifacts, which are referred to as individual components. What's more, Xue et al. (2006) introduced a revolutionary design database model describing design requirements and rules developed at different stages. The representation in the database comprises both geometric and non-geometric descriptions. Only differences between design descriptions at different design stages are recorded for propagation and consistency control. Rather than a full representation of a design product, Zahedi et al. (2021 & 2022) proposed the concept of design episodes, which capture different bits and pieces of a design product and encapsulate them in various episodes by utilizing storytelling techniques (Martin et al., 2003). Similarly, a partial representation of a design product can be applied for change management to significantly reduce the amount of effort.


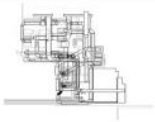

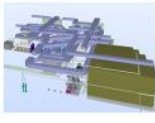
Modeling and conversion (1)	Revit MEP modeling	Revit→IFC→Revit
		
Data loss analysis (1)	Shape Visualization Location Layer Property System Loss of object	Recognition of 2D shapes only due to loss of view tree data Loss of view and color data Loss of object location and grid data Loss of object layer Loss of object property data Loss of MEP system data 222 out of 1031 objects preserved (78.8% lost)
Modeling and conversion (2)	Revit MEP modeling	Revit→IFC→Solibri
		
Data loss analysis (2)	Shape Visualization Location Layer Property System Loss of object	Good overall recognition of object shapes in IFC file Loss of color data Loss of object location and grid data Good overall recognition of layers as IFC data Good overall recognition of property data Good overall recognition of MEP system data 1,078 out of 1,031 objects preserved (0.2% increased)

Figure 10: Data loss in file conversion between different formats from Oh et al. (2015)

3 Methodology

3.1 Research Gap

BIM-based building design is a complex process where design operations can lead to significant redesigns involving numerous building components and their dependencies. These dependencies can be classified into two categories: low-level constraints and topological constraints. While most low-level constraints, such as geometrical constraints, have already been implemented in BIM authoring tools, topological constraints derived from engineering knowledge are often overlooked. This is primarily because topological constraints are project-specific and can only be defined and implemented incrementally throughout the design process. Nevertheless, they are essential for ensuring that the design product maintains a meaningful correspondence with reality.

This thesis aims to develop a BIM-based approach to support building design revisions by incorporating change propagation processes, which typically depend on topological constraints, into design operations. The key challenges include defining the necessary topological constraints and integrating them with design operations to enable prior evaluation of the change propagation processes for intended design revisions. This information can help determine whether to proceed with a given operation. Additionally, another challenge is to devise a method for tracking design revisions throughout the design process, as accumulated experience can help prevent similar mistakes in future design tasks.

3.2 Research Method

To address these challenges, a BIM-based change management framework consisting of two parts was proposed. The first part focuses on integrating change propagation processes into design operations within BIM authoring tools. A well-defined change propagation mechanism is essential for this integration, as it addresses two core problems: controlling the direction and extent of change propagation. Specifically, a set of topological constraints was established based on disciplinary knowledge to determine the direction of change propagation, given that changes can only propagate through existing dependencies. Additionally, a significance matrix, informed by the concept of DSM and expert knowledge, was proposed to manage the scope of propagation. This

matrix indicates whether dependencies exist between building components and identifies which ones hold greater significance. Consequently, change propagation between two components is possible only if at least one dependency exists between them. Overall, a change propagation mechanism should be developed within a specific design context to align with the requirements of the design project. With such a mechanism in place, change propagation can be effectively integrated into BIM-based design operations. Moreover, this mechanism was also utilized to design the documentation system in the second part, as it needs to record change propagation processes to track design revisions.

The Design Revision Manager (DRM), resulting from this integration, serves as the change management system. It can execute design operations that not only delete or add individual building components but also automatically address change propagation processes. Design revisions or operations performed throughout the design process are exported to a developed database. The documentation system is crucial because past revisions, or accumulated experience, can help designers avoid making similar mistakes in future tasks. Furthermore, by querying data from the design revision database, the DRM provides essential information regarding topological constraints and the effects of change propagation processes prior to executing intended operations. In summary, the availability of past revisions, current topological constraints, and potential change propagation results can significantly enhance decision-making in BIM-based building design revision. The underlying philosophy is that informed decisions can be made when the consequences of each choice are known in advance.

The core components of the integration phase—specifically, the development of the change propagation mechanism based on expert knowledge, which includes the definition of topological constraints and the corresponding significance matrix—are elaborated in Chapter 3.3 and 3.5.1, respectively. Additionally, the design operations considered in this thesis are summarized in Chapter 3.4. The database schema, based on the proposed change propagation mechanism, is illustrated with a set of Entity-Relationship models in Chapter 3.6.

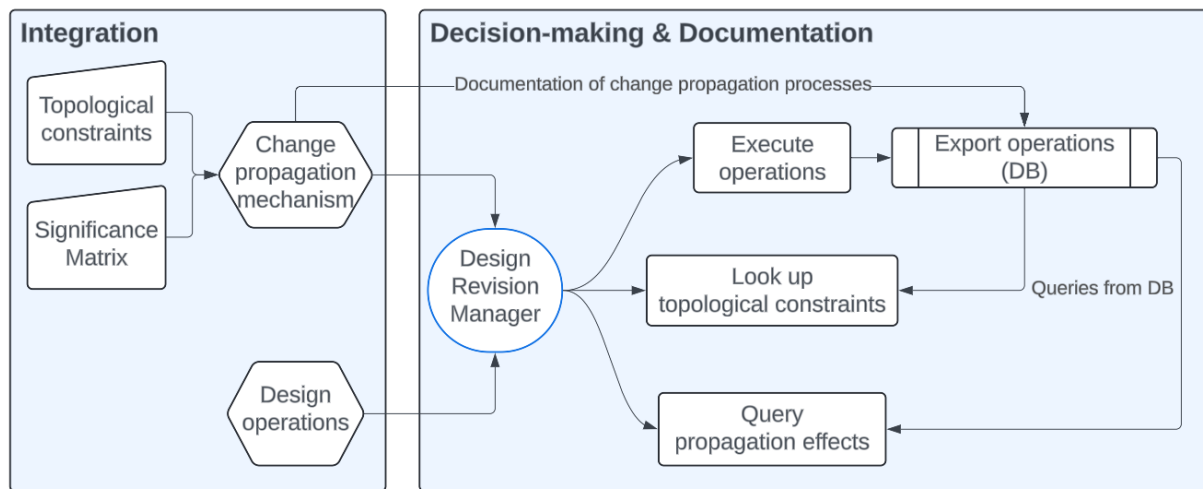


Figure 11: Proposed method

3.3 Topological Constraints

In engineering databases, semantic integrity refers to the meaningful correspondence between data and reality (Ullman 1988). To maintain the semantic integrity of a design product, specific integrity rules must be applied to relevant data and met by the end of the design phase. Traditionally, adding and evaluating these rules during the building design process is a manual and time-consuming task. As the number of rules increases, re-evaluations initiated by design revisions become increasingly complex, as initial changes can propagate through a chain of dependencies defined by these integrity rules. In the context of computer aided design, integrity rules—including geometric constraints, unit definitions, and construction practices—can be embedded within CAD software. Thus, BIM-based building design simultaneously generates a data set and a corresponding set of integrity rules, ensuring that the building is feasible for construction upon design completion (Eastman et al. 1997).

In this paper, integrity rules are defined as constraints arising from dependencies between building components, which must be incrementally imposed throughout the design process to ensure the design product maintains its semantic integrity by the end of the design phase. Constraints in BIM models can be classified into geometric and non-geometric (semantic and topological) rules. This research categorizes dependencies into low-level constraints and topological constraints (Figure 12).

Low-level constraints are those inherently implemented in most CAD software. For example, dimensional or geometrical constraints are commonly embedded in nearly all CAD systems. These low-level constraints are applicable across various engineering

design disciplines. In BIM authoring tools, geometric constraints form the basis for families—flexible geometric models that can be quickly adapted to meet varying boundary conditions (Borrmann et al. 2018). Consequently, these cross-disciplinary low-level constraints enhance design efficiency and significantly reduce unintended design errors, such as those related to modifying dimensions or positioning and aligning building components. This distinction underscores how computer-aided design differs from traditional building design workflows, which predominantly rely on paper and pencil.

In contrast, topological constraints are more project-specific and discipline-dependent, requiring expert knowledge. Given that building design is a large-scale and complex task involving numerous participants from various domains, identifying and implementing all topological constraints at any given stage is simply impractical. In this paper, topological constraints are abstracted and classified into two types: structural constraints and architectural constraints.

Table 1: Topological constraints in BIM-based building design considered in this study

Structural constraints		Connected to (CON)
		Supported by (SUP)
Architectural constraints	Spatial dependency	Room/enclosed area (ROO)
		Surrounded by (SUR)
		Clashed by (CLA)
	Accessibility	Clearance area in front doors, stairs' exit/entrance (CLE) Floor opening for Stairs (OPE)

Structural constraints primarily focus on structural components, verifying whether a structural element is connected to (CON) or supported by (SUP) other structural components. In contrast, architectural constraints encompass all building elements, including non-physical objects. Architectural constraints are further divided into two subcategories: spatial layout and accessibility. Regarding spatial layout, three specific constraints are considered: the first, ROO, checks if a given element forms the boundary of a room or enclosed area; the other two, CLA and SUR, are designed to identify clashes and undesirable neighboring building components. In terms of accessibility, two relevant constraints include the clearance area directly in front of doors and stairs (CLE) and the necessary opening above stairs (OPE).

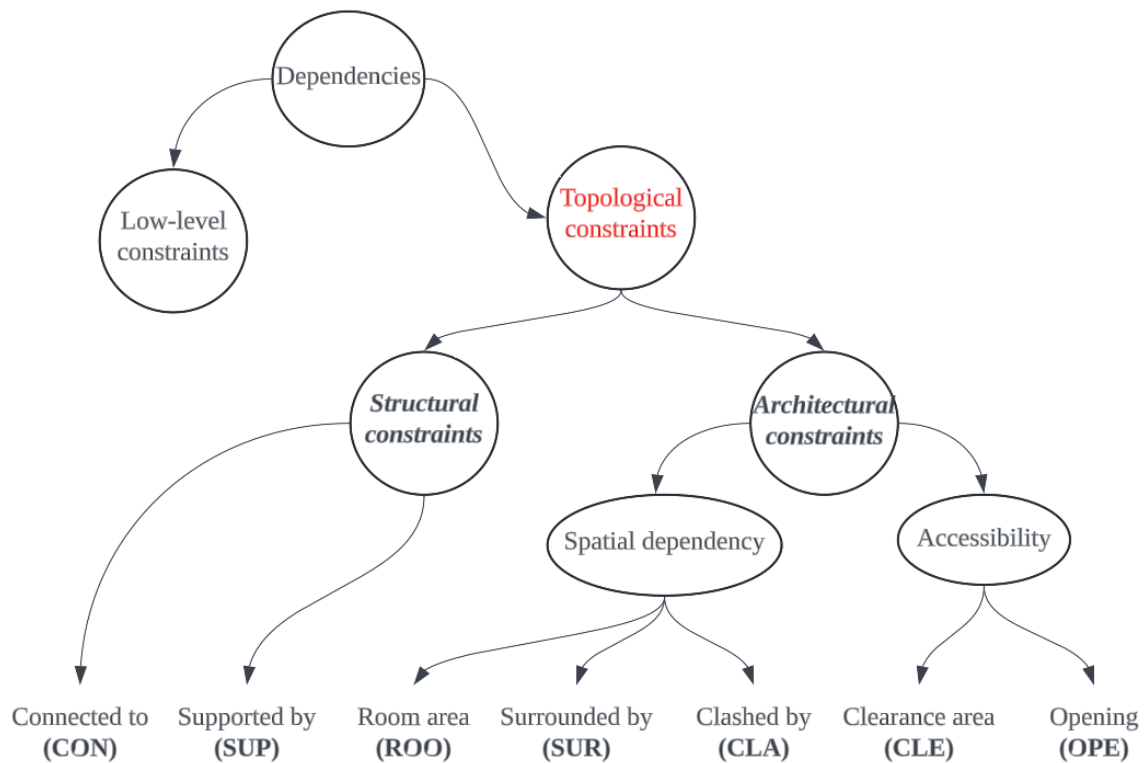


Figure 12: Represent dependencies between building components with low-level and topological constraints

3.4 Operations for Building Design Revisions

Eastman et al. (1997) defined operations as mechanisms for applying constraints to data by adding, deleting, or modifying building elements or their attributes, which results in a new design state and functional dependencies as side effects. In this research, building design revisions involve performing such operations on design data. This study focuses exclusively on element-level operations, such as adding and deleting components. Changes in geometric properties are particularly significant because they are more likely to propagate to related components, thereby having a greater impact on the overall building structure. For instance, the consequences of altering the thermal properties of a wall are considerably less critical than those of deleting a wall in terms of architectural and structural design. As such, modifications to semantic properties are not considered in this study.

This research primarily addresses two types of operations for building design revisions using BIM authoring tools: adding and deleting components. These operations serve as the foundation for more complex design operations, such as moving, which combines both adding and deleting. In the early stages of BIM-based building design, building components with large quantities are essential for three-dimensional modeling, as

they form the primary building structure and are crucial for cost estimation. Sub-systems, such as HVAC, typically rely heavily on the main building skeleton. Therefore, this study focuses on design revisions involving walls, columns, and stairs—components that significantly affect aspects like the main building structure, spatial layout, and accessibility.

In this paper, we consider only structural walls and structural columns for these operations due to their complexity and comprehensive nature, as they encompass more constraints than their architectural counterparts. In total, this research implements six operations (Table 2): deleting structural walls, structural columns, and stairs, as well as adding structural walls, structural columns, and stairs.

Table 2: Operations for building design revision

		Building Components		
		Structural Wall	Structural Column	Stair
Operations	Delete	x	x	x
	Add	x	x	x

note: x means the design operation is applied to the corresponding building component

3.5 Change Propagation Mechanism

3.5.1 Significance Matrix

As discussed in Chapter 2.4, Design Structure Matrix (DSM) effectively displays the relationships or interdependencies between discrete engineering parts or components, illustrating how design changes can propagate. It also represents both the likelihood and impact values of these changes (Keller et al. 2005). Building on the DSM, a significance matrix (Figure 13) is introduced to illustrate the connections and relative importance of building components. There are three possible relationships between two components: no relation, equal significance, or one being more important than the other.

Change propagation can only occur when dependencies exist. For example, the significance matrix indicates that structural walls are generally more important than architectural walls, while structural walls hold equal significance among themselves. This implies that structural walls take precedence over architectural walls during operations. For instance, if a new structural wall conflicts with an existing architectural wall, the architectural wall should be deleted. Conversely, if a conflict arises between two struc-

tural walls during an operation, further confirmation may be necessary before proceeding. Specifically, inserting a new structural wall that overlaps with an existing one will require an approval from designer: the operation will continue if the response is affirmative, or be canceled if it is negative.

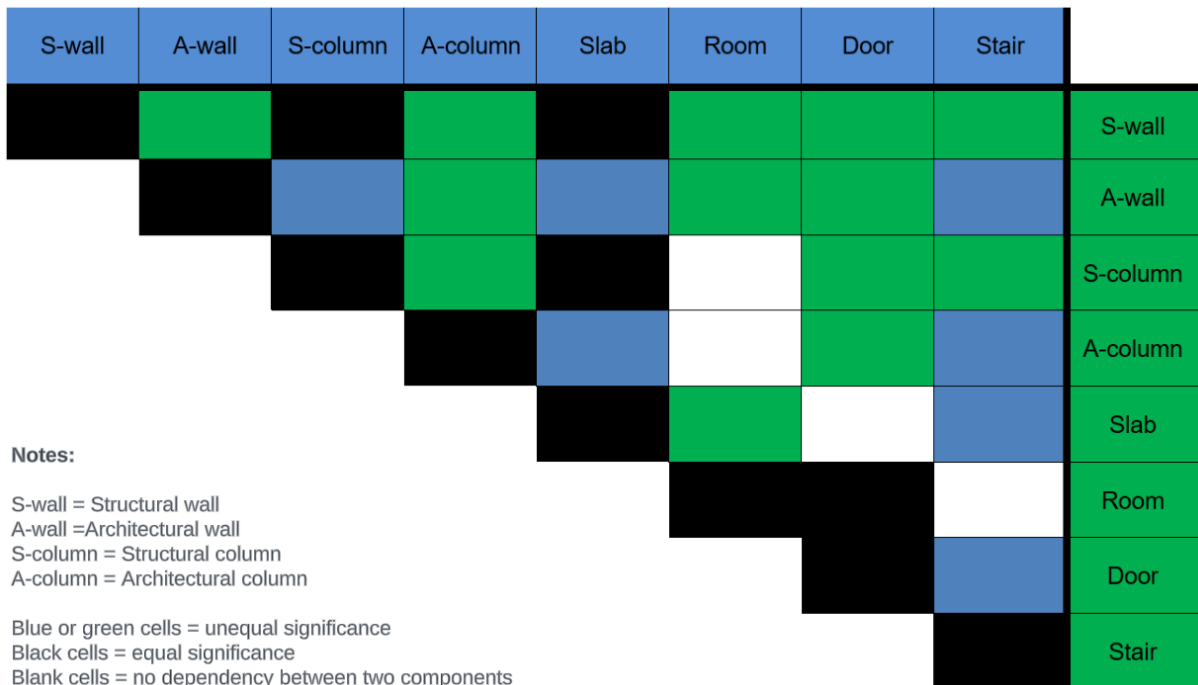


Figure 13: Significance Matrix (indicates the existence of dependencies between two building components and controls the extent of change propagation)

Table 3: Different constraints for different operations (based on topological constraints and the significance matrix)

		S-wall	A-wall	S-column	A-column	Slab	Room	Door	Stair
S-wall	Add	CON CLA	CLA	CON		SUP	ROO	CLA CLE	CLA CLE
	Delete	CON		CON			ROO		
S-column	Add	CON		CON CLA SUR	CLA SUR	SUP		CLA CLE	CLA CLE
	Delete	CON		CON					
Stair	Add	CLA CLE	CLA CLE	CLA CLE	CLA CLE	SUP OPE		CLE	CLA
	Delete					OPE			

Notes: S stands for structural, A stands for architectural

Additionally, a summary table (Table 3) is defined to represent the dependencies and topological constraints between various building components for different design revision operations, as outlined in Table 1 and Table 2, and illustrated in Figure 13. This table reflects that constraints are only partially considered for each operation, as different constraints are activated depending on the specific operation and building components involved. For example, two relevant constraints (CON and CLA) must be checked between two structural walls during an ADD operation, while only one constraint (CON) needs to be examined for a DELETE operation, specifically to determine if any structural walls are connected to the one being deleted.

3.5.2 Change Propagation Hierarchy

Design changes can lead to modifications in related building components, and improper management of change propagation can result in extensive redesign or even design failure. To address integrity violations during change propagation, two critical aspects must be considered. First, the path of change propagation needs to be identified, which has already been effectively addressed in Table 3.

Changes can only propagate along constraints that establish dependencies between discrete building components. Second, it is essential to define the scope of change propagation to enable automatic propagation. According to Eastman's theory (Figure 5), which conceptualizes change propagation as a cone, an initial change will continue to propagate as long as there are existing dependencies, as illustrated in Figure 14; this implies that change propagation can potentially be infinite.

In this paper, change propagation is generally limited to four levels, as the number of constraints considered is finite, preventing a highly complex dependency network. The naming principle for different levels is as follows: "No-Propagation" signifies that a change has no impact on other components, while the level of change propagation is denoted by incremental numbers. The "First-Propagation" occurs when changes affect directly connected building components. As detailed in Table 3, this can be achieved by simply checking the corresponding constraints during an operation on a specific component.

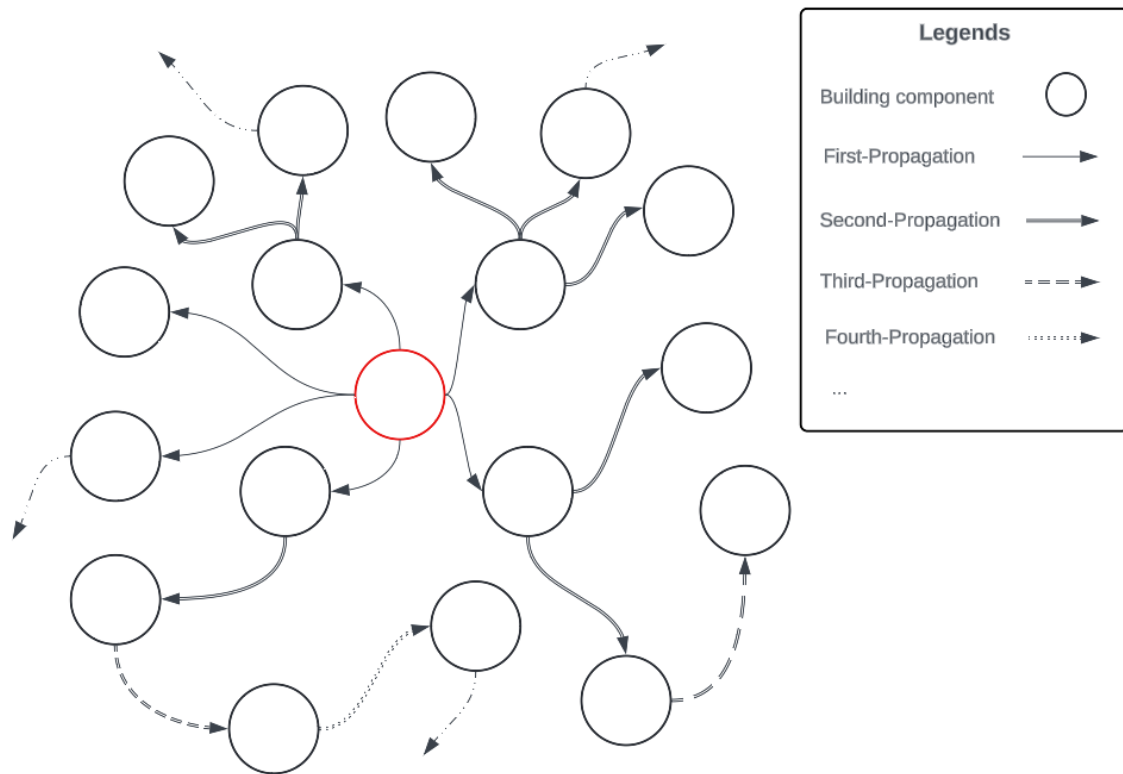


Figure 14: Change Propagation Hierarchy

The significance matrix (Figure 13) not only illustrates the connections between components but also indicates the potential for further propagation, namely Second-Propagation and Third-Propagation. Typically, propagation is feasible only if a dependency exists between two components. Furthermore, changes can only propagate to components of equal or lesser significance, which helps clearly define and limit the scope of change propagation. Second-Propagation concludes reached once it has checked the constraints on the first set of indirectly connected components. For instance, as shown in Figure 15, deleting a structural wall requires identifying its connected structural walls (the CON constraint), resulting in First-Propagation. Subsequently, it must be determined whether the identified connected structural walls are structurally stand-alone. At this stage, in addition to the directly connected structural walls, the first level of indirectly connected structural walls is evaluated, achieving Second-Propagation. Finally, Third-Propagation involves identifying affected and invalidated rooms, which must be deleted if the corresponding structural walls are stand-alone and have been removed in previous propagation stages. It is important to note that while First-Propagation can result in either DELETE or ADD operations, only deletion is considered in any subsequent propagations.

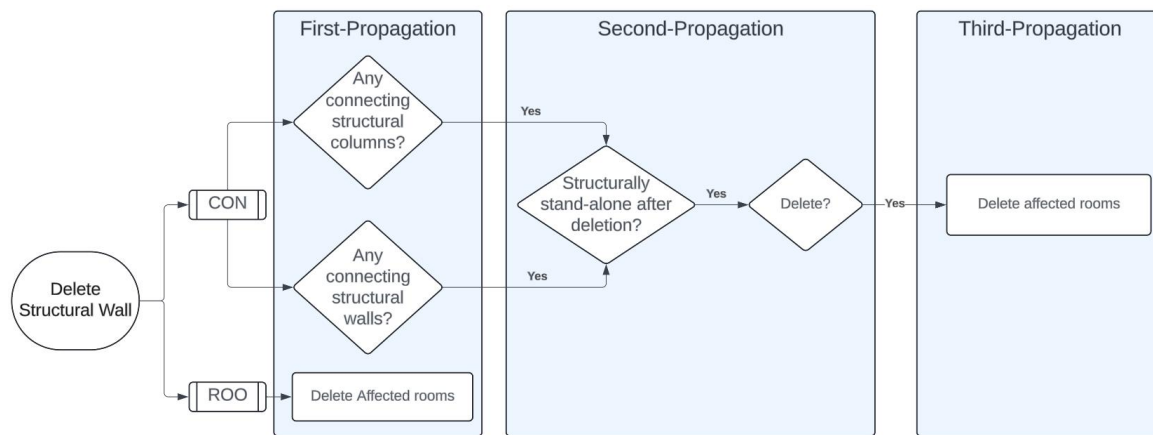


Figure 15: Example of Change Propagation Hierarchy

3.6 Database Schema

Default design operations in BIM authoring tools primarily handle common, cross-disciplinary low-level constraints in engineering design, as discussed in Chapter 3.3. These built-in commands facilitate modifications to documents through transactions, which ensure database consistency by managing changes. Each transaction results in a new data state that represents the design product. A key advantage of transactions is their ability to be rolled back after successful execution, allowing users to access both the previous state and the latest version of the design. The proposed operations can similarly be implemented using transactions. However, access to historical transactions is limited, as the database of BIM authoring tools only partially grants access. Transactions remain valid only during an active session, meaning all records are lost once the current document is closed. Understanding historical design changes is crucial for further improvements, highlighting the need for a dedicated database to track and document transactions throughout the design process.

The database schema is specifically developed for documenting and managing building design revisions initiated by the operations proposed in Chapter 3.4. Consequently, the corresponding data model must accurately represent the state of the design and its evolution over time. Representing a design state necessitates information about both the identities of building components involved in the project and the active dependencies among them. Based on Table 1, Figure 13, and Table 3, eight Entity-Relationship (ER) models are proposed. Seven ER models, illustrated in Figures 15 to 21, represent the identities and active topological constraints of seven categories of building components, reflecting the design state of each. Additionally, an additional

ER model named Revision (Figure 23) is designed to record executed operations, detailing when and how design revisions are performed and their side effects.

Unnecessary relationships are omitted for clarity. For instance, architectural walls share the same relationships as structural walls (Figure 16). However, according to the significance matrix (Figure 13), architectural walls can only propagate changes to components of equal or lesser significance, including architectural walls, architectural columns, rooms, and doors. Furthermore, operations cannot be directly performed on architectural walls; they can only be deleted during the First-Propagation and then propagate further changes. Therefore, the active constraints for architectural walls should include only rooms and doors. Additionally, only relationships derived from topological constraints are illustrated in the ER models, omitting properties of entities to simplify representation, as the relationships are already complex. In this research, two properties are utilized: the identifier and the version number of building components. The identifier serves as a reference for modeling dependencies and relationships between components, akin to a foreign key in a relational database, while the version number aids in tracking and recording design revisions. More details can be found in the software implementation section (Chapter 4).

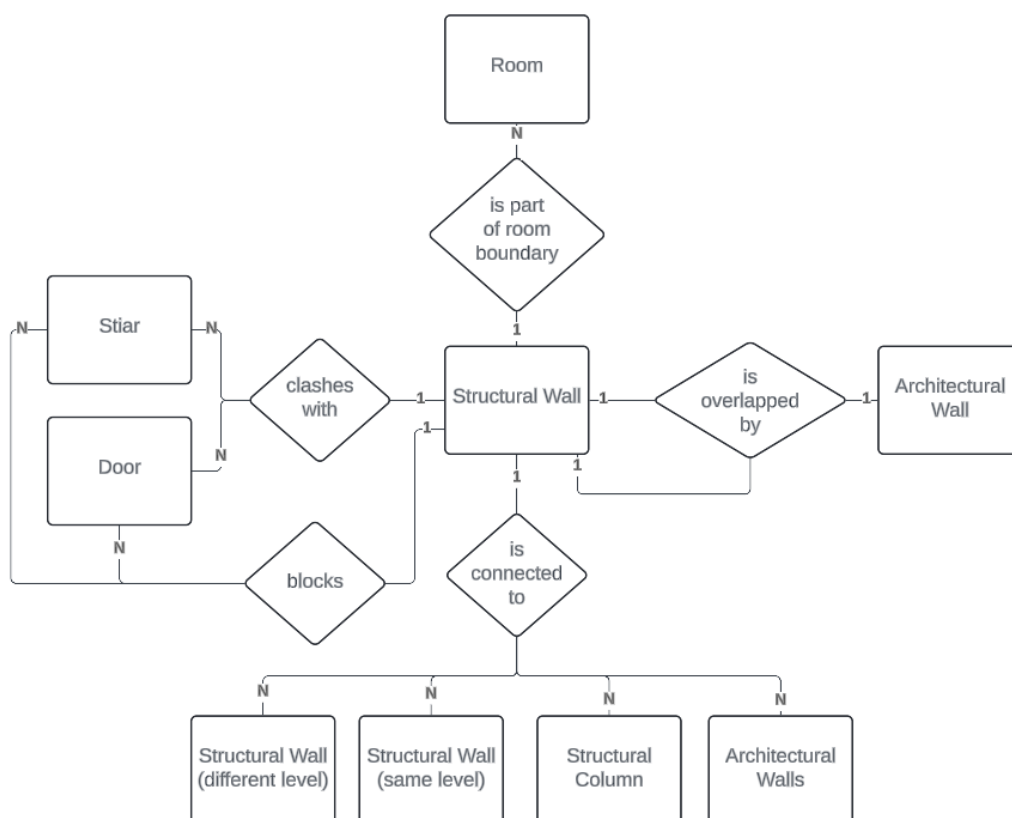


Figure 16: ER model of structural wall



Figure 17: ER model of architectural wall

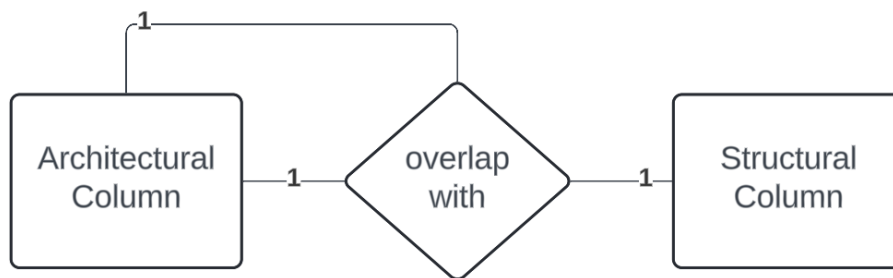


Figure 18: ER model of architectural column

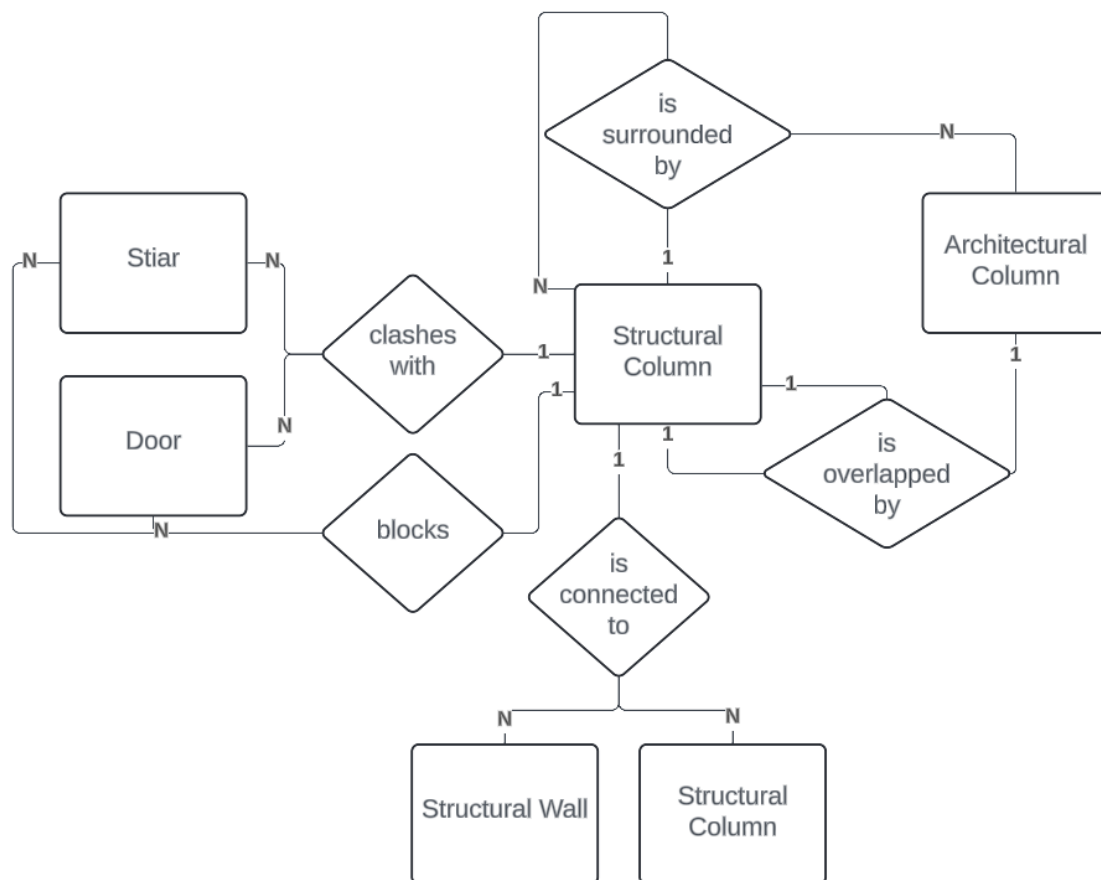


Figure 19: ER model of structural column

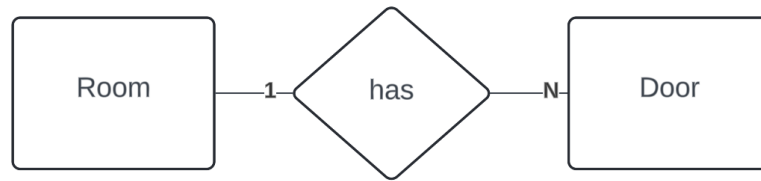


Figure 20: ER model of room

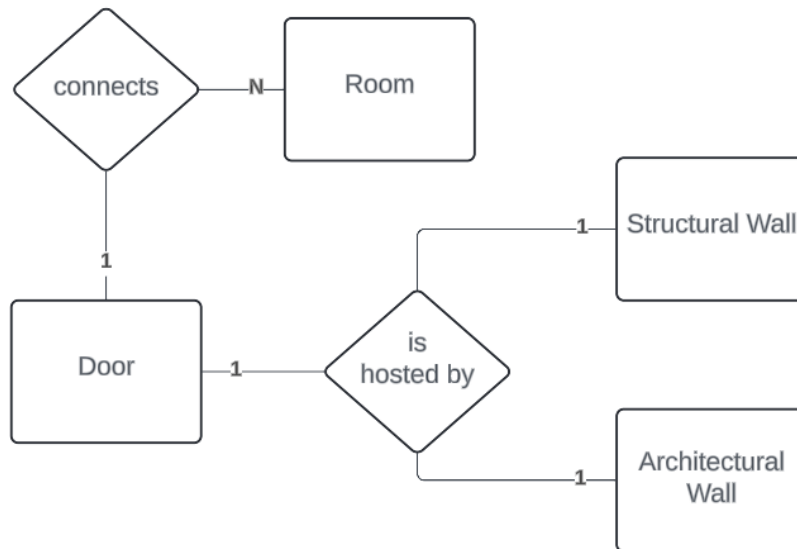


Figure 21: ER model of door

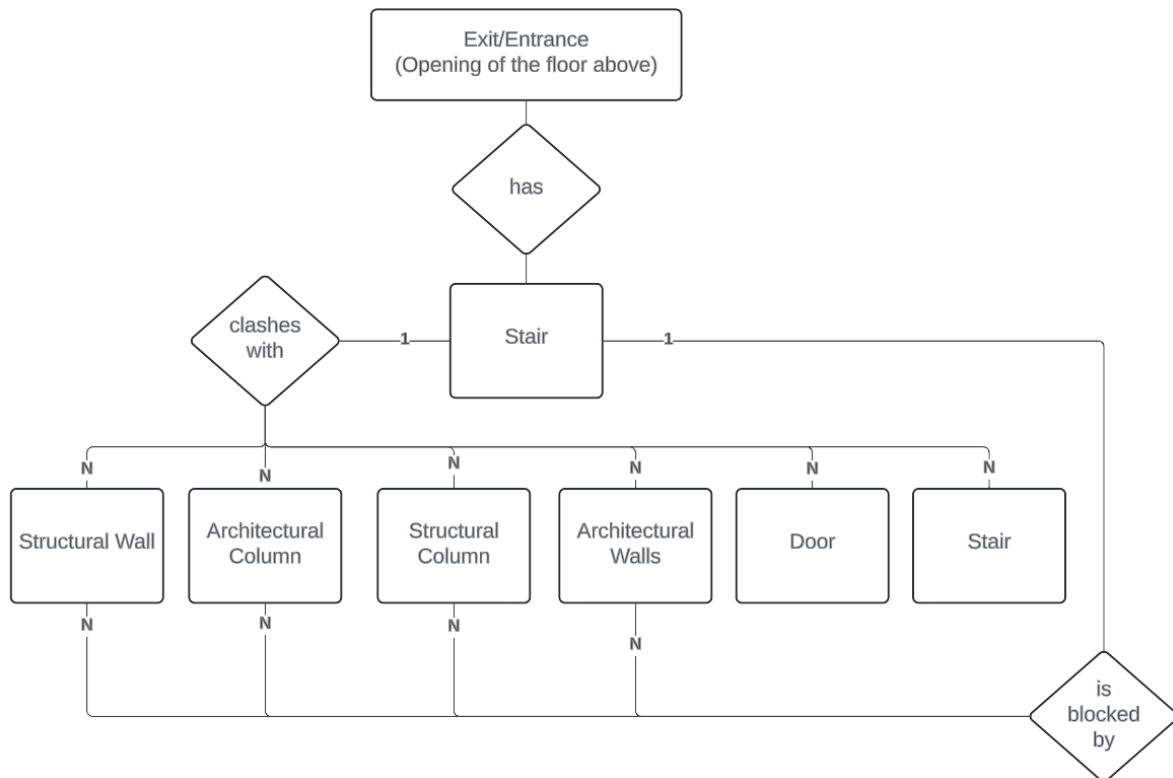


Figure 22: ER model of stair

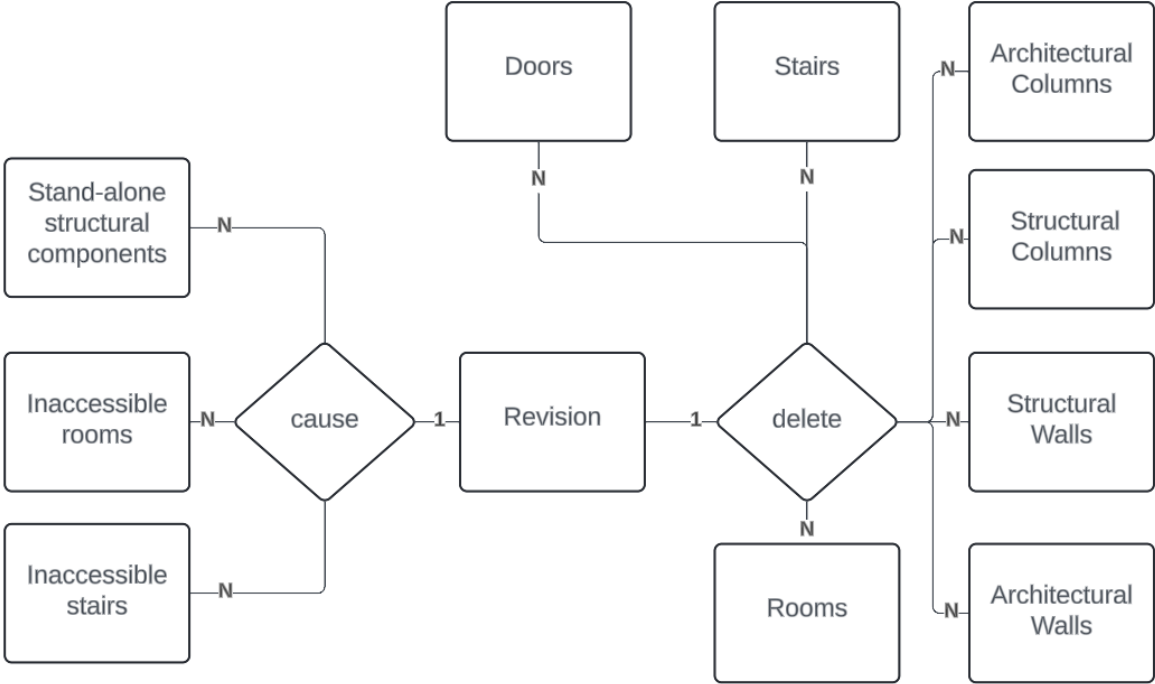


Figure 23: ER model of revision

4 Prototype Implementation

4.1 Test Data

The design operations defined in Chapter 3.4 can only be executed on a model that adheres to established design norms, codes, and best practices in building design. In the early stages of design, floor plans serve as the primary documents. Any alterations to these floor plans can lead to significant differences in both quantity and cost. For this research, a floor plan that includes structural and architectural walls, columns, stairs, slabs, and doors is used for modeling, as changes to these elements can impact the overall plan. Additionally, rooms, though non-physical entities, are also modeled due to the ROO constraint. The test model was developed using Autodesk Revit 2023.

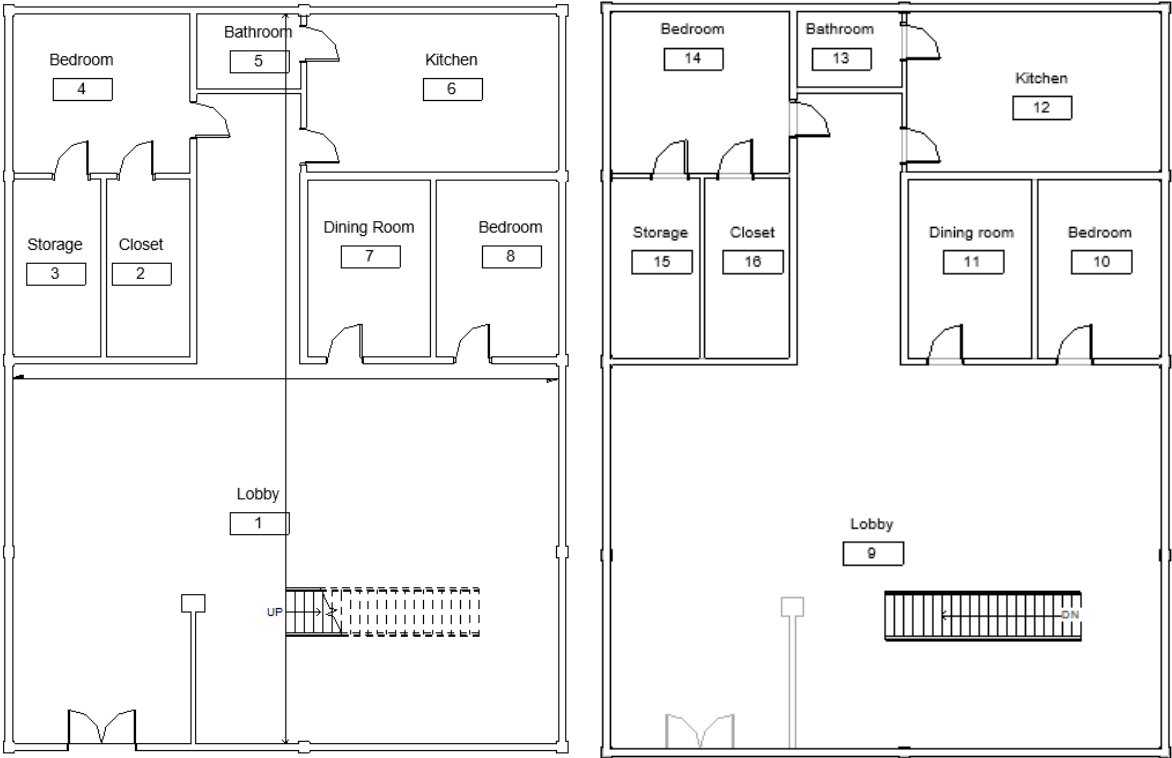


Figure 24: Floor plans of the test model
(the floor plan on the left is the ground floor and the one on the right is the first floor)

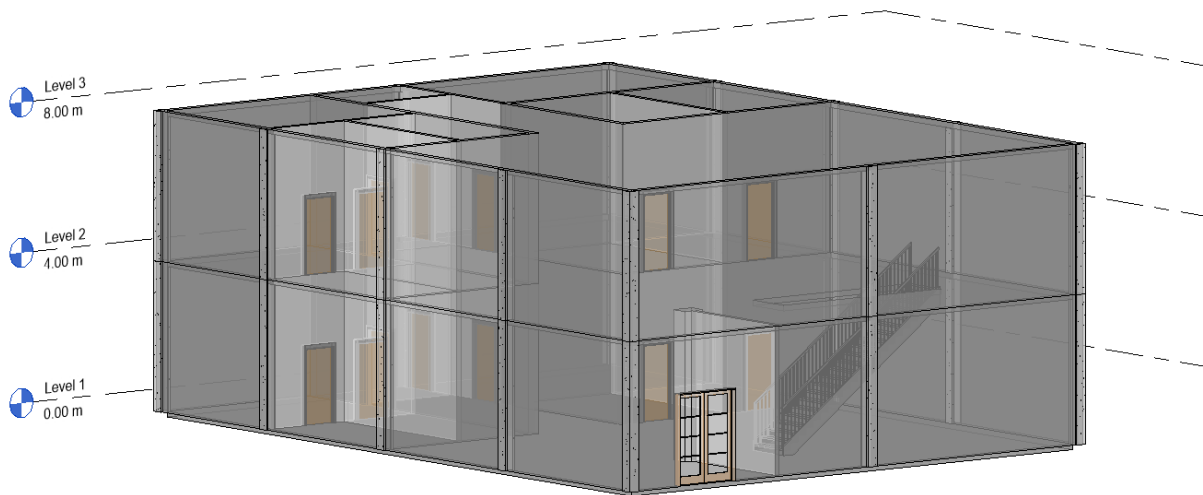


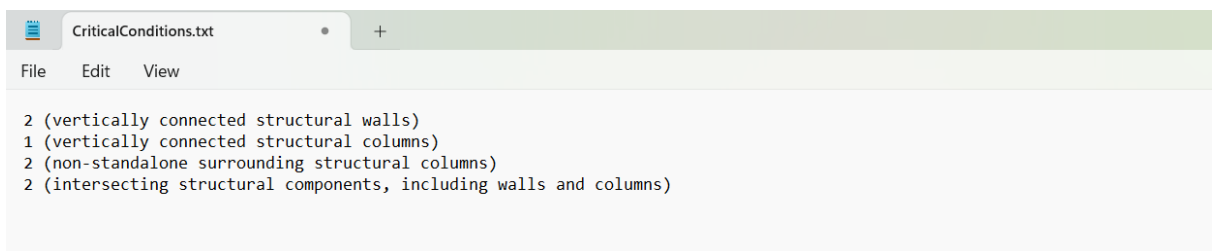
Figure 25: Test model (3D view)

4.2 Implementation of Operations

Structural walls, structural columns and stairs are considered as subjects, on which the operations of design revision are to be performed, namely delete and add operations, while the other building components in the BIM model take part only in the subsequent change propagation processes. The workload of the implementation of operations can be divided into two parts. The first part is to implement the simple delete and add functionalities for the three components without any consideration of topological constraints via the Application Programming Interface (API) of Autodesk Revit 2023. Further is to incorporate topological constraints into the commands, resulting in change propagation to a desired extent. The implementation of propagation is based on Figure 13 and Table 3. The active constraints indicate the potential directions of propagation, whereas the significance matrix (Figure 13) gives information regarding the possibility of propagating a change, namely changes are allowed to solely propagate to less or at least the same important building components. The following is the implementation of the six proposed operations. A text description and a flow-chart are provided for each of the commands to elaborate the logic and its full functionalities. Red fonts in the illustration of active constraints indicate how far the given constraint propagates. Additionally, this research considers only rational operations and design revisions, which are to be performed under the assumption that the given BIM model is well constructed by following codes and good practices in building design. Furthermore, Revit Lookup (Tammik et al. 2023), which is an interactive Revit RFA and RVT project database exploration tool to view and navigate BIM element parameters, properties and relationships, is most often used to help with the implementation.

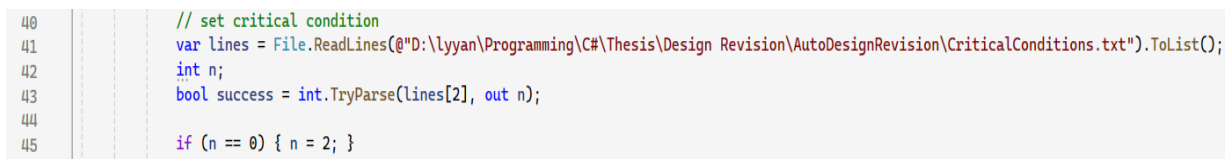
4.2.1 User Input

There are four critical conditions for the following six operations, namely the number of vertically connected structural walls for deleting structural wall, the number of vertically connected structural columns for deleting structural column, the number of non-standalone surrounding structural columns for adding structural column, and the number of intersecting structural components (walls and columns) for adding stair. Critical conditions are responsible for determining whether to continue with the current operation or not. The numbers for the conditions are stored in a text file, whose content can be easily read (Figure 26). Instead of hard-coded conditions, the user can set the critical conditions manually as desired with the help of this input file. More flexibility is achieved.



```
2 (vertically connected structural walls)
1 (vertically connected structural columns)
2 (non-standalone surrounding structural columns)
2 (intersecting structural components, including walls and columns)
```

Figure 26: Input file



```
40 // set critical condition
41 var lines = File.ReadLines(@"D:\lyyan\Programming\C#\Thesis\Design Revision\AutoDesignRevision\CriticalConditions.txt").ToList();
42 int n;
43 bool success = int.TryParse(lines[2], out n);
44
45 if (n == 0) { n = 2; }
```

Figure 27: Read critical conditions from input file

4.2.2 Structural Wall Deletion

According to the significance matrix, changes caused by operations on structural walls can propagate to all the eight components. However, only the evaluation of the structural constraint (CON) and the spatial layout constraint (ROO) is relevant for deleting structural walls (Table 3) in First-Propagation. CON refers to both horizontally and vertically connected structural walls and structural columns. The connected structural components can result in Second-Propagation. Namely, connected structural walls from other stories and any other connected structural components, which appear to be structurally stand-alone after executing the deleting command, are taken into consideration. In addition, structural walls deleted in Second-Propagation can trigger Third-Propagation due to their related rooms. The number of vertically connected structural walls, which is required as user input, are viewed as critical condition, since violating

this constraint can result in termination of the current operation. Moreover, the deletion of the structurally stand-alone structural components in Second-Propagation requires confirmation. All rooms related to the walls, which are to be deleted during the operation (First-Propagation and Third-Propagation), are to be deleted by default as well.

As illustrated in Figure 28, before deleting the structural wall, it needs to be checked if there are connecting structural components, resulting in First-Propagation. Furthermore, the connecting structural walls need to be checked if they are structurally stand-alone components, which then results in Second-Propagation. Thereafter, deleting some walls in the Second-Propagation can lead to the deletion of some rooms, resulting in Third-Propagation. For ROO constraints, only First-Propagation is possible since rooms cannot propagate changes further.

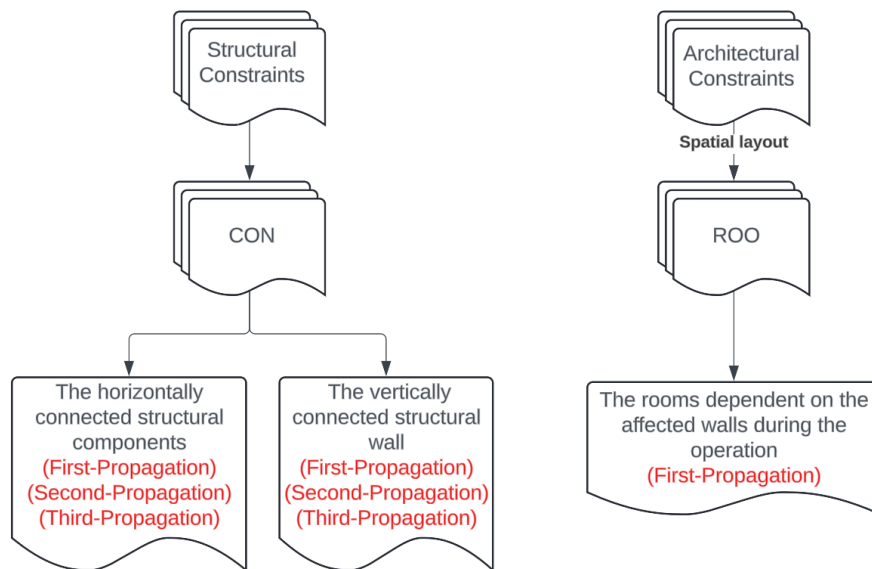


Figure 28: Constraints for Structural Wall Deletion

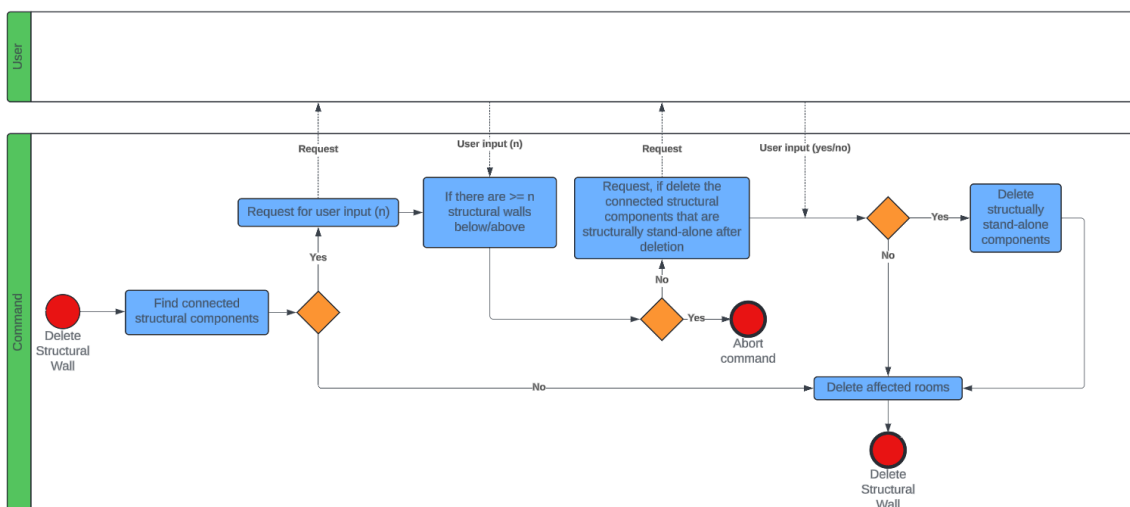


Figure 29: Structural Wall Deletion

To examine if a structural wall is structurally stand-alone, the class LocationCurve in the Revit API is of importance, as illustrated in Figure 32 using Revit Lookup. The location curves of structural walls below or above should have the same direction, and the length of these walls should be approximately the same, if the walls are to be recognized and classified as structurally connected. A method called IsStandAloneStructuralWall is implemented accordingly. The logic of the function is illustrated in Figure 30. In the examination, structural walls from levels both above and below are considered, whereas only structural columns from the same level are relevant. Namely, the walls from the same level are considered for finding intersecting walls, while walls from the other levels are for finding walls at the same location in terms of the x-y plane. Moreover, detection of intersecting columns is implemented by using the get_BoundingBox and the BoundingBoxIntersectsFilter methods from the Revit API, which is basically to check if there's an intersection between the bounding boxes (Figure 31) of the given elements.

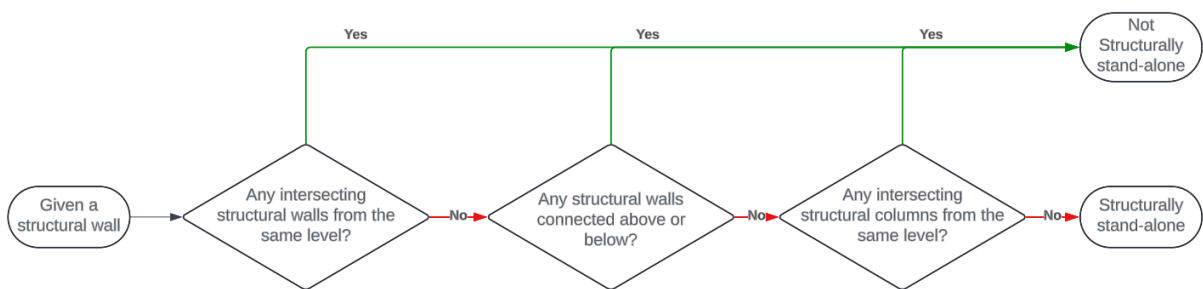


Figure 30: Workflow of the method IsStandAloneStructuralWall

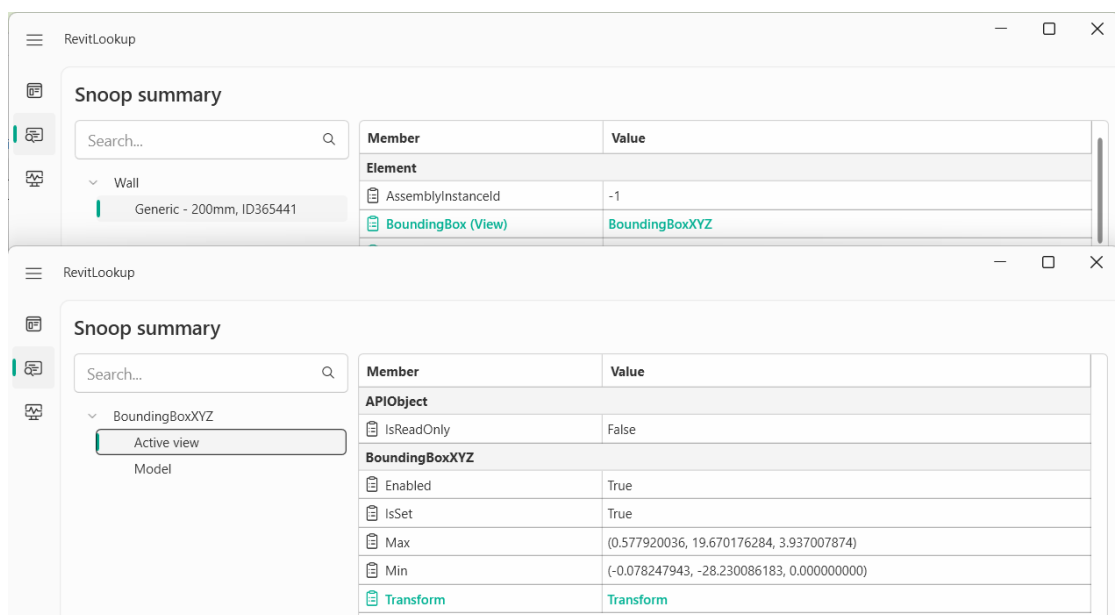


Figure 31: BoundingBoxXYZ class in Revit API

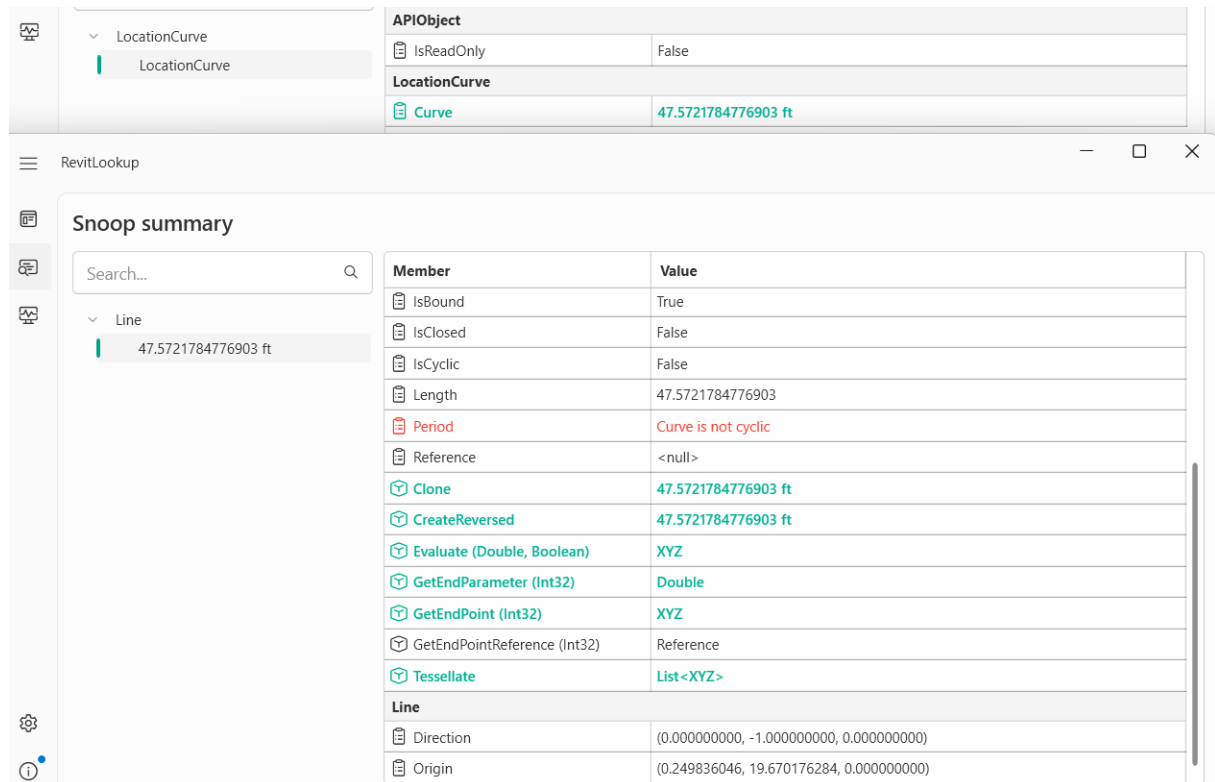


Figure 32: LocationCurve of wall

4.2.3 Structural Column Deletion

Similarly, structural columns can propagate changes to all other building components as well. However, rooms are not considered, since they are not dependent on columns. Structural connections can trigger Second-Propagation and therefore, are important for deleting a structural column (Table 3). Connected structural walls can trigger Third-Propagation due to relations to rooms. Vertical connections between structural columns (CON) are significant. Namely, the number of structural columns, which are at the same location point in terms of the x-y plane, but from different levels, is the critical condition for this operation. User input is required to set this number and to decide if to delete the structurally stand-alone structural components as well. Deleting the rooms dependent on the affected walls (ROO) during the operation is only possible in the Third-Propagation, since columns and rooms are not directly related.

As illustrated in Figure 33, First-Propagation can be achieved by checking if the to-be deleted structural column is structurally stand-alone. Following is the Second-Propagation, which is to check if there are further structural connections to the connected structural components found in the First-Propagation. The connecting Structural walls can lead to Third-Propagation because some rooms might be removed due to the deletion of walls in the Second-Propagation.

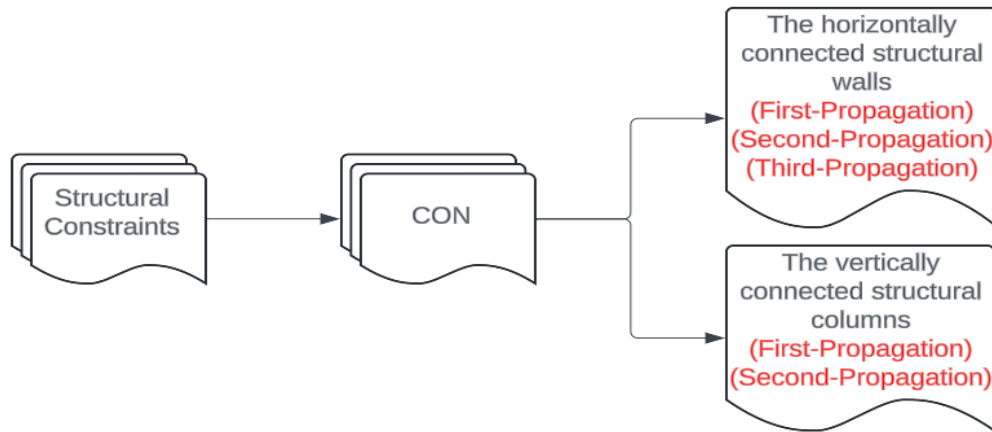


Figure 33: Constraints for Structural Column Deletion

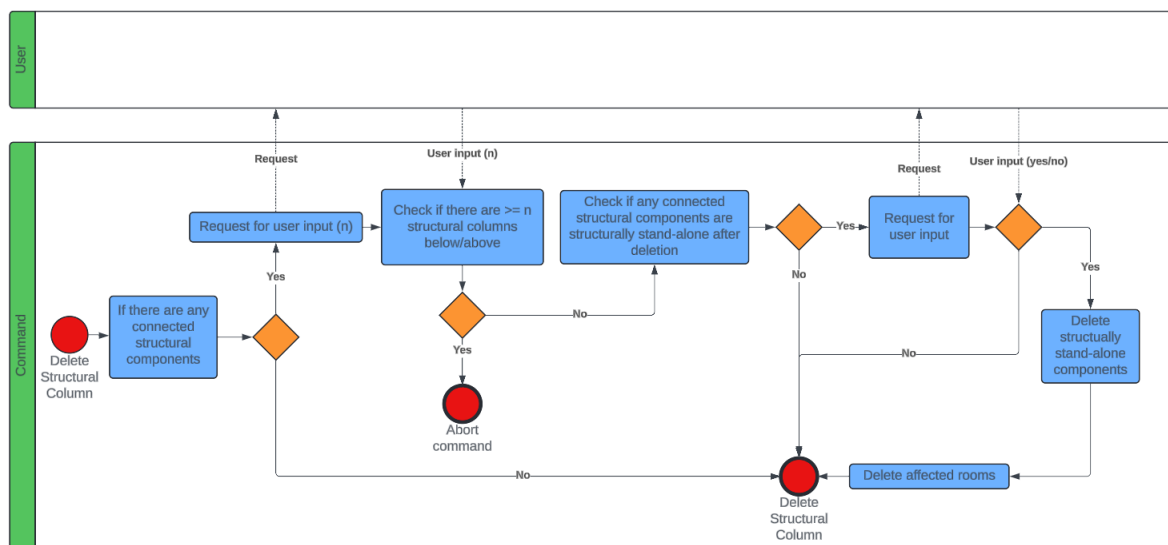


Figure 34: Structural Column Deletion

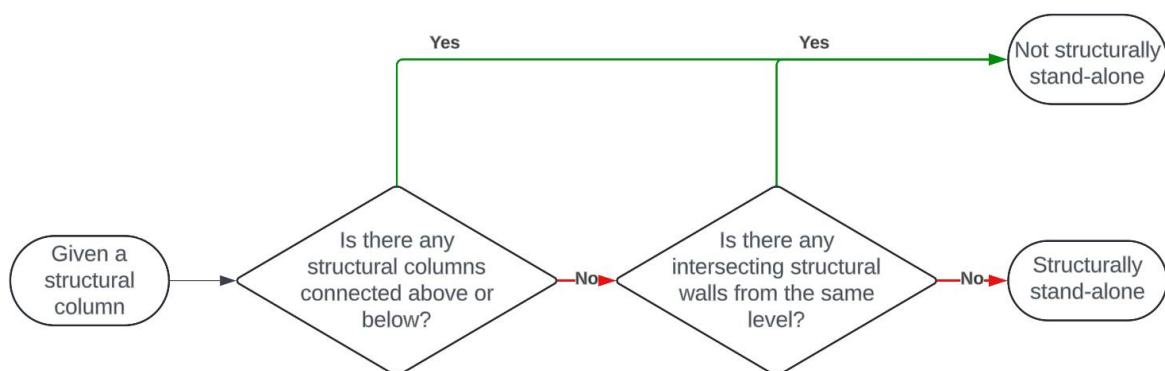


Figure 35: Workflow of the method IsStandAloneStructuralColumn

Structural columns below and above can be found by comparing the location points. Revit API provides a class called LocationPoint for accessing the location point of a column, which is similar to the LocationCurve. If the location points stay the same, two columns are at the same location in terms of the x-y plane. Similar to structural wall, a

method named `IsStandAloneStructuralColumn` is implemented as well, which takes both structural columns from different levels and structural walls only from the same level into consideration (Figure 35).

4.2.4 Stair Deletion

Stairs can only propagate changes mostly to architectural components (Figure 13). In terms of deleting stairs, the accessibility constraint (OPE) is considered, which is to basically delete the opening of the slab right above the deleted stair. There could be openings at the base level of the deleted stair as well. Though, openings from the base level are usually meant for the stair below.

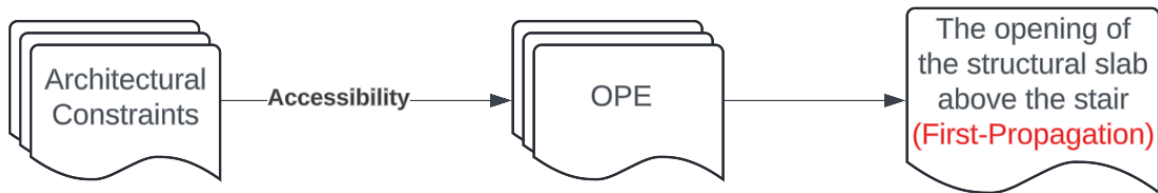


Figure 36: Constraints for Stair Deletion

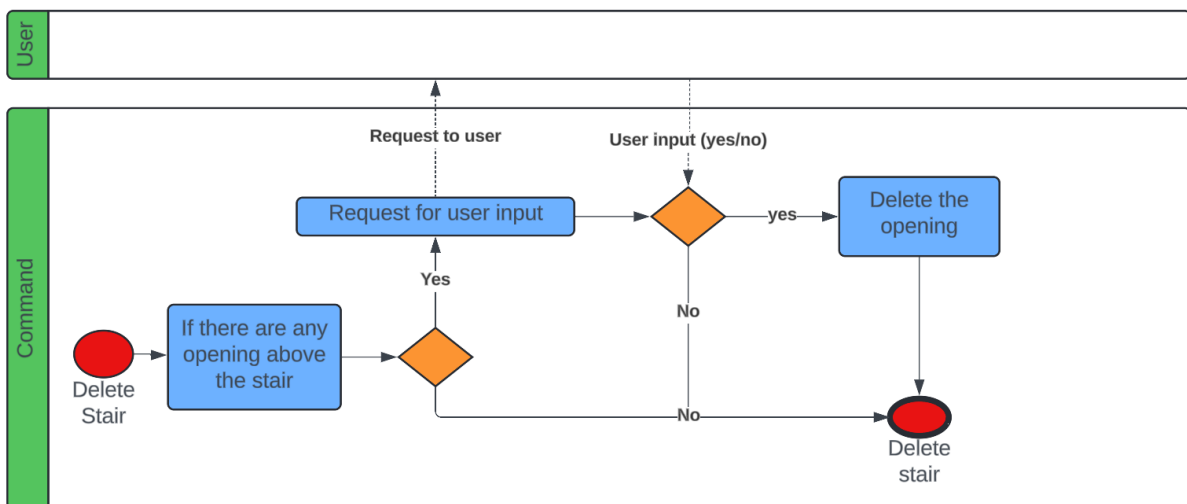


Figure 37: Stair Deletion

4.2.5 Structural Wall Addition

Compared to the delete operation, adding a structural wall requires an evaluation of more active constraints, namely CON, CLA, CLE, ROO, and SUP. Since a wall is created in Autodesk Revit in such a way that it has both the top level and the base level constraints, a wall is not necessarily supported by (SUP) a structural slab anymore. Besides, there are no rules of thumb for the SUR constraint. Namely, SUR and SUP

are not considered in the implementation. The constraints, namely CON and CLA, are viewed as critical conditions here. User input is required to decide if to cancel the operation if the new structural wall is either a stand-alone building component or a structurally stand-alone structural component (CON). Additionally, an overlapping structural wall (CLA), which is not structurally stand-alone (Second-Propagation), can result in cancelation of the current operation. An auto-alignment of the inserted wall is performed if there is at least one vertically connected structural wall (CON). Other components, which directly clash with the inserted wall, are to be deleted as well. In terms of the CLE constraint, the accessibility of rooms and stairs are checked. Namely, only rooms that have at least one door, or at least one of the boundary lines is not a wall, are viewed as accessible. Namely, deleting clashing doors can lead to Second-Propagation. For stairs, a clearance area of at least one square meter in front of the exit or entrance should be guaranteed, meaning that the inserted wall should not be in this clearance area. Moreover, ROO is implemented to delete all the rooms dependent on the affected or modified walls during the operation.

Different from DELETE operations, CON constraints in ADD operations can only result in First-Propagation, meaning that it only needs to be checked if the inserted structural wall is structurally stand-alone. In addition, stairs and rooms do not propagate changes further and therefore can only cause First-Propagation. Doors, however, can have impacts on rooms (Second-Propagation), which means that some rooms might become inaccessible if their doors are deleted. Moreover, First-Propagation can be achieved by searching for overlapping walls. Though, only overlapping structural walls can lead to Second-Propagation since they need to be checked if structurally independent.

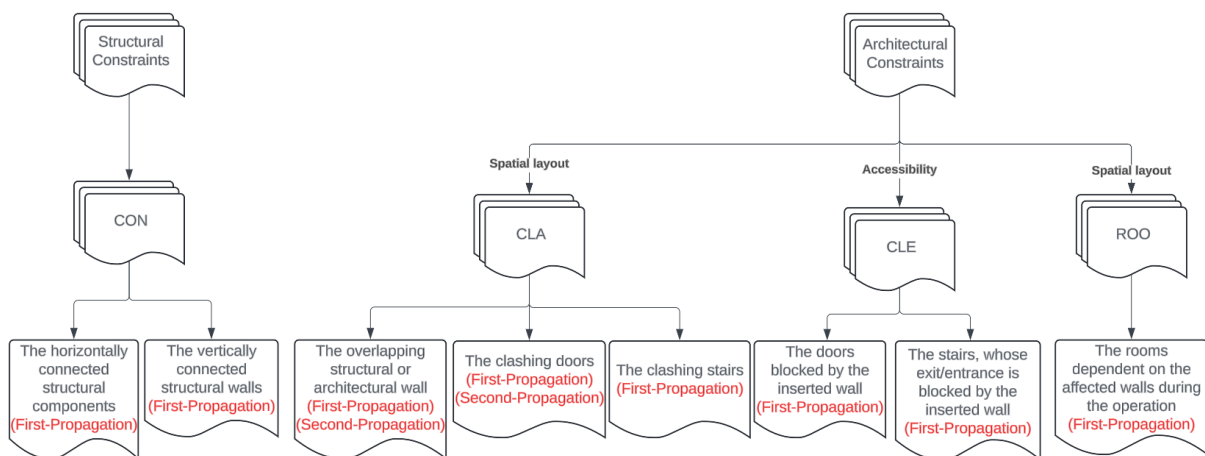


Figure 38: Constraints for Structural Wall Addition

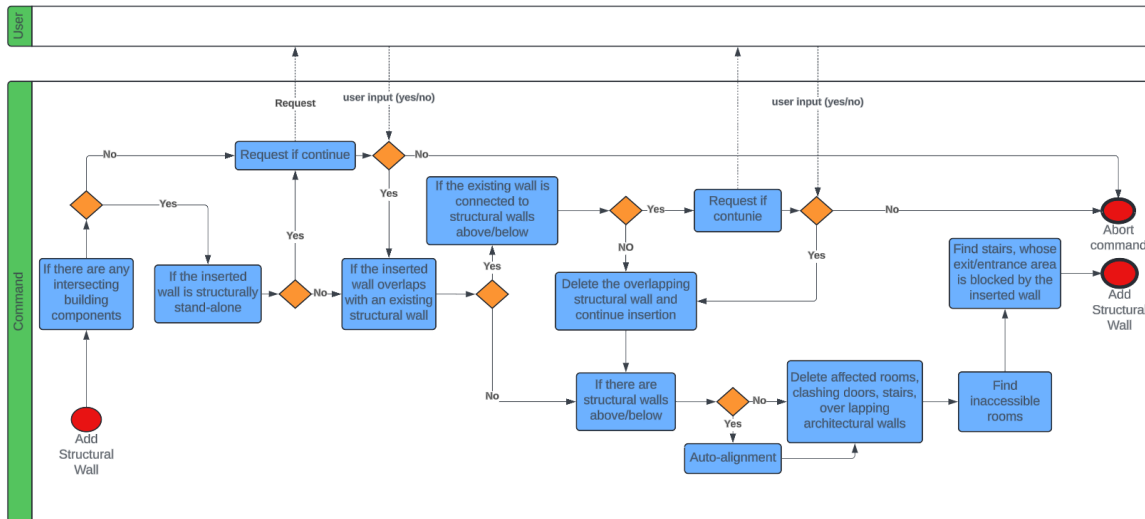


Figure 39: Structural Wall Addition

The creation of a wall instance is based on the method Create (Figure 40), which lies in the DB namespace of Revit API. The method requires a location curve and the base level along with the height of the wall as parameters. Therefore, this operation is implemented in such a way that the user is allowed to create a wall by simply picking two points in a floor plan. The picked two points are automatically aligned and the resulting location curve is always parallel to the x-axis or the y-axis. Besides, a newly inserted structural wall is automatically aligned to the connected structural walls from levels below or above. Furthermore, an overlapping wall can be identified through a comparison of the location curves and the levels in which the walls reside (Figure 41). The method SameLocWalls, which is to find the walls with the same location curve but from different levels, is also used in the implementation of IsStandAloneStructuralWall. Clashing components can be identified by checking if there's an intersection between the bounding boxes of two building components. In terms of inaccessible rooms, a method called IsRoomAccessible (Figure 42) is implemented based on the GetBoundarySegments method of the Autodesk.Revit.DB.Architecture.Room class. Given a room, it can tell if the room is accessible or not. Namely, a room is accessible if it has at least one door or at least one of the room boundary lines is a room separation line (a built-in category in Autodesk Revit) instead of a solid wall instance.

```

163     try
164     {
165         structWall = Autodesk.Revit.DB.Wall.Create(doc, geoLine, type.Id, currentLevel.Id, height, 0.0, true, true);
166         doc.Regenerate();
167     }
168     catch (Exception ex)
169     {
170         throw ex;
171     }
    
```

Figure 40: Autodesk.Revit.DB.Create method

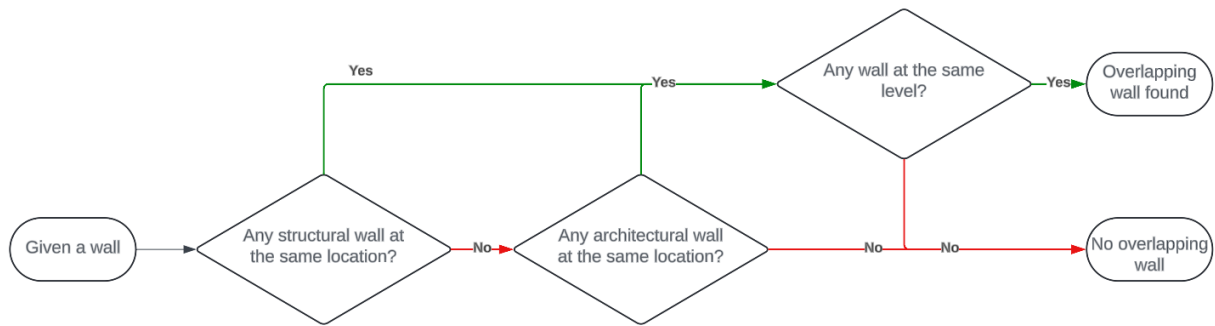


Figure 41: Workflow of the method FindOverlappingWall

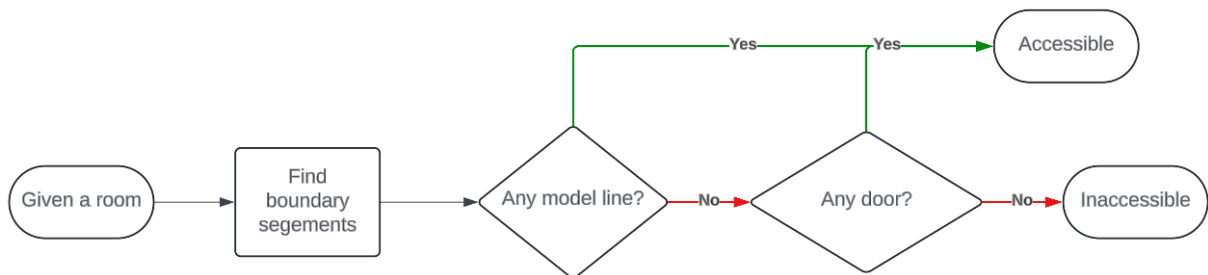


Figure 42: Workflow of the method IsRoomAccessible

4.2.6 Structural Column Addition

The following five constraints are considered for change propagation processes induced by adding a structural column, namely CON, CLA, SUR, SUP, CLE. The existence of surrounding structural columns within 1 meter (SUR), including structural columns from levels below and above, can lead to Second-Propagation and termination of the current operation. One of the critical conditions is still to check if the newly inserted structural column is structurally stand-alone (CON). The procedure in terms of handling overlapping structural columns is similar to that of handling overlapping structural walls. Therefore, overlapping structural columns can result in Second-Propagation as well. What’s more, the operation deletes surrounding stand-alone structural columns (Second-Propagation) and any other clashing or surrounding architectural components, including overlapping (CLA) and nearby architectural columns (SUR), clashing doors and stairs (CLA), and stairs, whose clearance area is blocked by the newly inserted structural column (CLE). The accessibility of the rooms, which are related to the deleted doors, is checked in Second-Propagation as well. Since columns and walls are automatically joined and connected in Autodesk Revit, the corresponding clashing constraint CLA can be omitted. In addition, SUP is resolved by the top level and base level constraint.

Similar to structural walls, CON constraints and stairs here are also only for First-Propagation. The evaluation of overlapping structural columns is based on the same principle as that of overlapping structural walls, involving First- and Second-Propagation. There is no big difference in terms of evaluating the propagation levels of doors as well. What's more, the surrounding columns need to be considered as First-Propagation. Especially for the surrounding structural columns, their further structural connections need to be evaluated in the Second-Propagation.

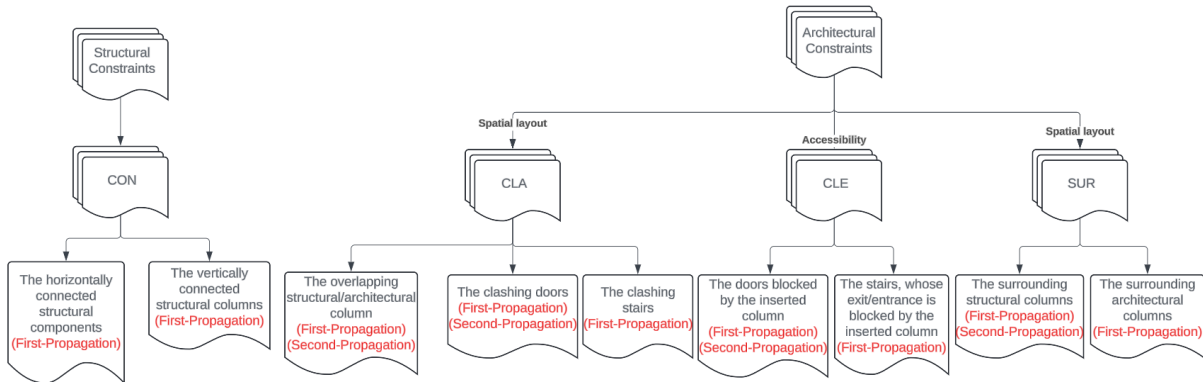


Figure 43: Constraints for Structural Column Addition

The creation of a structural column is implemented based on the method `Autodesk.Revit.Creation.Document.NewFamilyInstance` (Figure 45), which requires a location point and a base level of the column as parameters. Not only to find the overlapping columns at the same location point (Figure 46), but this operation takes also surrounding columns into consideration. When the distance between the location points of two columns are less than one meter, the two columns can be identified as mutually surrounding (NearbyColumns in Figure 47).

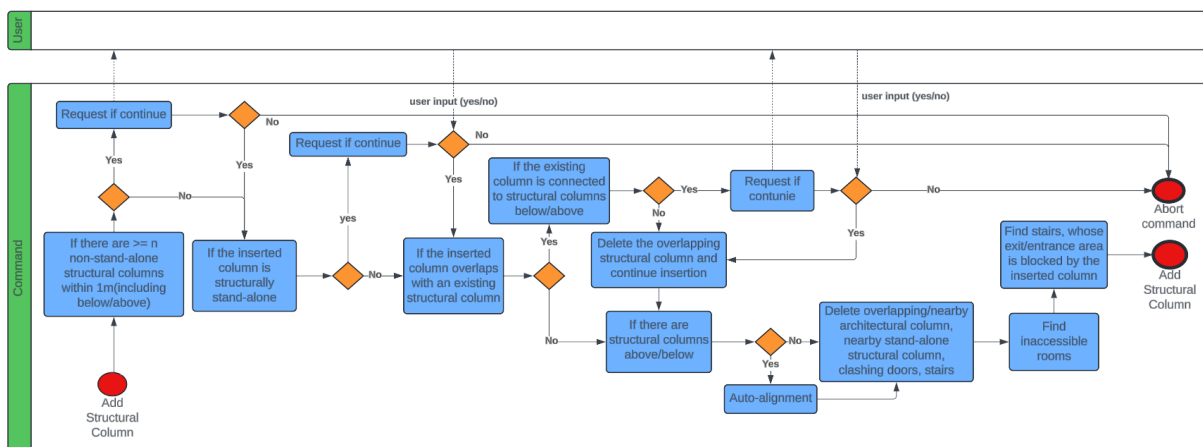


Figure 44: Structural Column Addition


```

57 #region Insert structural column by picking a point, auto alignment, delete clashing & nearby columns within 1m
58 if (type != null)
59 {
60     try
61     {
62         var tx = new Transaction(doc, "Add Column");
63
64         tx.Start();
65
66         // insert a structural column by selecting a point
67         var p = uidoc.Selection.PickPoint();
68         column = doc.Create
69             .NewFamilyInstance(p, type, doc.ActiveView.GenLevel, StructuralType.Column);
70         doc.Regenerate();
71     }
    }

```

Figure 45: Autodesk.Revit.Creation.Document.NewFamilyInstance

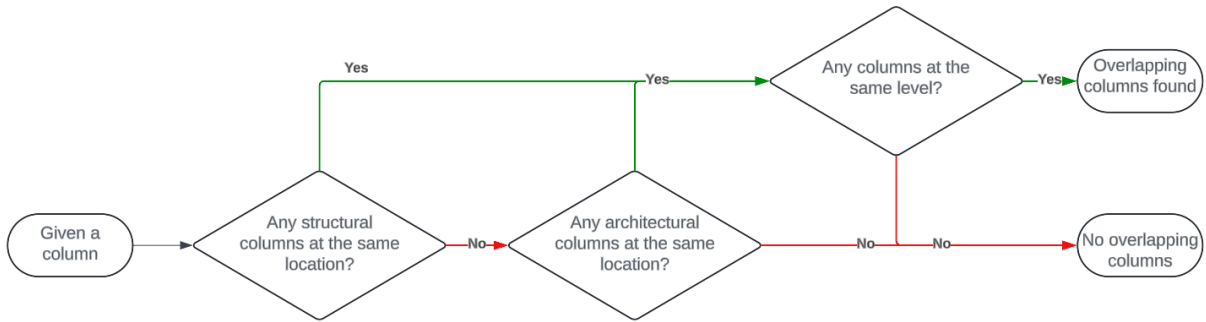


Figure 46: Workflow of the method FindOverlappingColumn

```

238 /// <summary>
239 /// Given a column, find surrounding structural and architectural columns
240 /// </summary>
241 /// <param name="column">The given struct column.</param>
242 /// <remarks>Nearby columns includes columns at the same location point within 33 cm,
243 /// and surrounding columns within 3.3 feet (ca. 1 meter) </remarks>
244 /// <returns>Ids of (structColumnsSameLoc, structColumnsNearby, archiColumnsNearby, structColumnsAbove, structColumnsBelow)</returns>
245 7 references
246 public static (List<ElementId>, List<ElementId>, List<ElementId>, List<ElementId>, List<ElementId>) NearbyColumns(Element column)
247 {
248     if (column.Category.BuiltInCategory != BuiltInCategory.OST_StructuralColumns &&
249         column.Category.BuiltInCategory != BuiltInCategory.OST_Columns)
250     { throw new Exception("Not a column."); }
251
252     var structColumnsSameLoc = new List<ElementId>();
253     var structColumnsNearby = new List<ElementId>();
254     var structColumnsAbove = new List<ElementId>();
255     var structColumnsBelow = new List<ElementId>();
256
257     var archiColumnsNearby = new List<ElementId>();
258
259     var doc = column.Document;
260
261     var structColumns = new FilteredElementCollector(doc)
262         .OfClass(typeof(FamilyInstance))
263         .WhereElementIsNotElementType();

```

Figure 47: The implementation of the method NearbyColumns

4.2.7 Stair Addition

For adding a stair, the clearance area (CLE) and the clashing (CLA) components are relevant dependencies. According to the significance matrix, structural components are relatively more significant than stairs, whereas stairs are of greater importance than architectural components. Therefore, clashing structural walls or structural columns can result in Second-Propagation and termination of the operation. Clashing structural walls have the potential to trigger Third-Propagation due to their related rooms. On the contrary, clashing architectural columns, walls, doors, and stairs are to be deleted directly, causing only First-Propagation. Though, deleting architectural walls and doors can lead to Second-Propagation as well, since the deleted walls can be a part of room

boundaries. All affected rooms during the operation are to be deleted. OPE is a constraint specific for stairs. Namely, an opening above the newly inserted stair is to be created after the insertion operation. It guarantees the accessibility of stairs.

First-Propagation applies to overlapping stairs, architectural columns and openings of slabs, since they do not propagate changes further. Because of the structural connections, structural walls can always propagate one level further than the architectural ones. Take a clashing structural wall for an example, detecting the clashing wall itself is the result of First-Propagation. Then this clashing structural wall needs to be checked if it's structurally stand-alone, resulting in Second-Propagation. If the clashing wall is structurally stand-alone and is deleted at the end, some rooms that are dependent on it also need to be deleted, which is Third-Propagation. In addition, doors can propagate changes to rooms, which makes Second-Propagation possible.

This operation allows the user to create a stair by picking two points (Figure 51), which can result in a location line with a direction, namely the center line of a stair. Though, the length of the stair run is 17 feet by default, which is suitable for a floor-to-floor height of 4 meters. Additionally, the default width of stair run is ca. 1.2 meters. The implementation is based on the StairsRun.CreateStraightRun method. In addition, clashing components can be identified by checking if there's an intersection between the bounding boxes of components. Based on the concept of bounding box, two methods are implemented to guarantee the accessibility of stairs, as illustrated in Figure 50.

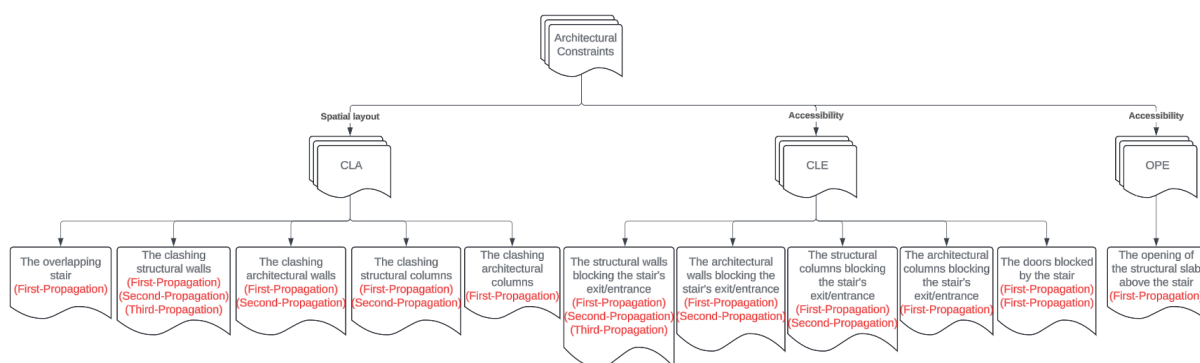


Figure 48: Constraints for Stair Addition

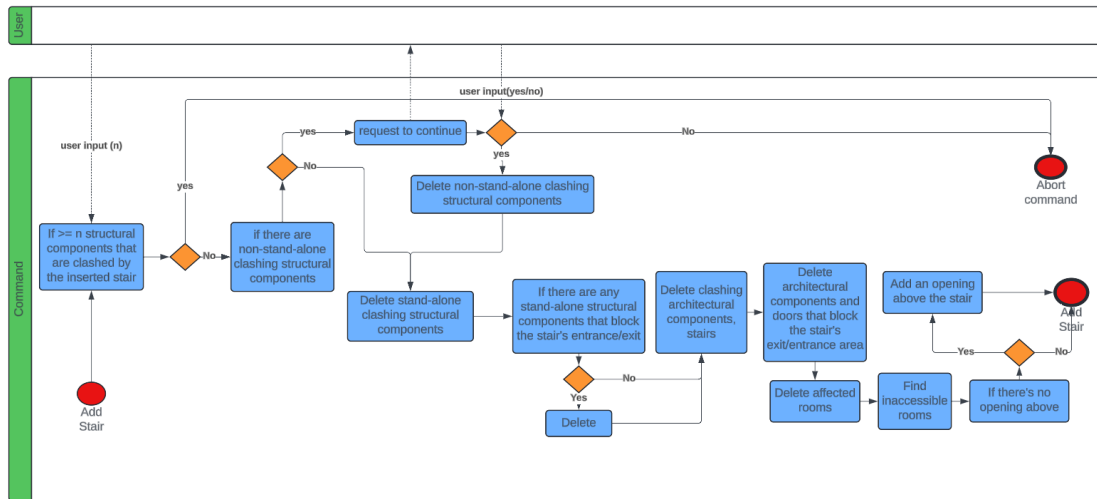


Figure 49: Stair Addition

```

399  /// <summary>
400  /// Given a stair, find the columns that block the exit/entrance of the given stair
401  /// </summary>
402  /// <param name="stair">The given stair</param>
403  /// <param name="structural">structural or architectural</param>
404  /// <param name="enlargement"></param>
405  /// <remarks>The exit/entrance area = enlargement * enlargement, right in front of the stair.
406  /// The enlargement is by default 3.3 feet. If structural = 1, return structural columns</remarks>
407  /// <returns>A list of column ids</returns>
408  /// 4 references
409  public static List<ElementId> FindColumnsBlockingStairExit(Element stair, int structural = 1, double enlargement = 3.3)
410  {
411  }
412
413  /// <summary>
414  /// Given a stair, find the walls that block the exit/entrance of the given stair
415  /// </summary>
416  /// <param name="stair">The given stair</param>
417  /// <param name="structural">structural or architectural</param>
418  /// <param name="enlargement"></param>
419  /// <remarks>The exit/entrance area = enlargement * enlargement, right in front of the stair.
420  /// The enlargement is by default 3.3 feet. If structural = 1, return structural walls</remarks>
421  /// <returns>A list of wall ids</returns>
422  /// 4 references
423  public static List<ElementId> FindWallsBlockingStairExit(Element stair, int structural = 1, double enlargement = 3.3)
424  {
425  }

```

Figure 50: Methods for checking the accessibility of stairs

```

632  stairsTrans.Start();
633
634  // Add a straight run
635  Line locationLine = null;
636
637  if (Math.Abs(pt1.X - pt2.X) > Math.Abs(pt1.Y - pt2.Y))
638  {
639      if (pt1.X - pt2.X < 0)
640      {
641          locationLine = Line.
642              CreateBound(new XYZ(pt1.X, pt1.Y, levelBottom.Elevation),
643                          new XYZ(pt1.X + runLength, pt1.Y, levelBottom.Elevation));
644      }
645      else
646      {
647          locationLine = Line.
648              CreateBound(new XYZ(pt1.X, pt1.Y, levelBottom.Elevation),
649                          new XYZ(pt1.X - runLength, pt1.Y, levelBottom.Elevation));
650      }
651  }
652  else
653  {
654      if (pt1.Y - pt2.Y < 0)
655      {
656          locationLine = Line.
657              CreateBound(new XYZ(pt1.X, pt1.Y, levelBottom.Elevation),
658                          new XYZ(pt1.X, pt1.Y + runLength, levelBottom.Elevation));
659      }
660      else
661      {
662          locationLine = Line.
663              CreateBound(new XYZ(pt1.X, pt1.Y, levelBottom.Elevation),
664                          new XYZ(pt1.X, pt1.Y - runLength, levelBottom.Elevation));
665      }
666  }
667
668  StairsRun run = StairsRun.CreateStraightRun(document, newStairsId, locationLine, StairsRunJustification.Center);
669  run.ActualRunWidth = 4.0;
670
671  stairsTrans.Commit();
672

```

Figure 51: Implementation of the method CreateSingleStraightRunStair

4.3 Implementation of Database Schema

This section is to describe the database schema proposed in chapter 3.6, aiming at recording design revision operations throughout the entire design process. The tools used for the implementation are Microsoft C# (.Net framework), Revit API 2023, and MongoDB, which uses a flexible way of storing data called documents and collections. A document can represent a single object, e.g. wall, door, column and so on, whereas a collection is to represent a set of objects. As illustrated in Figure 52, six collections are involved in the database for design revision, corresponding to the eight ER-diagrams in chapter 3.6, since walls and columns can represent both the structural and non-structural ones by simply adding an attribute to indicate whether a component is structural or not (Figure 53 & Figure 54). Each document in MongoDB must be assigned with a unique identifier. In this research, we use the Globally Unique Identifier (GUID) of building components obtained from the Revit model to initialize the identifier of documents (BsonId in Figure 53 & Figure 54 & Figure 55). GUIDs are used as reference to build relationships as well.

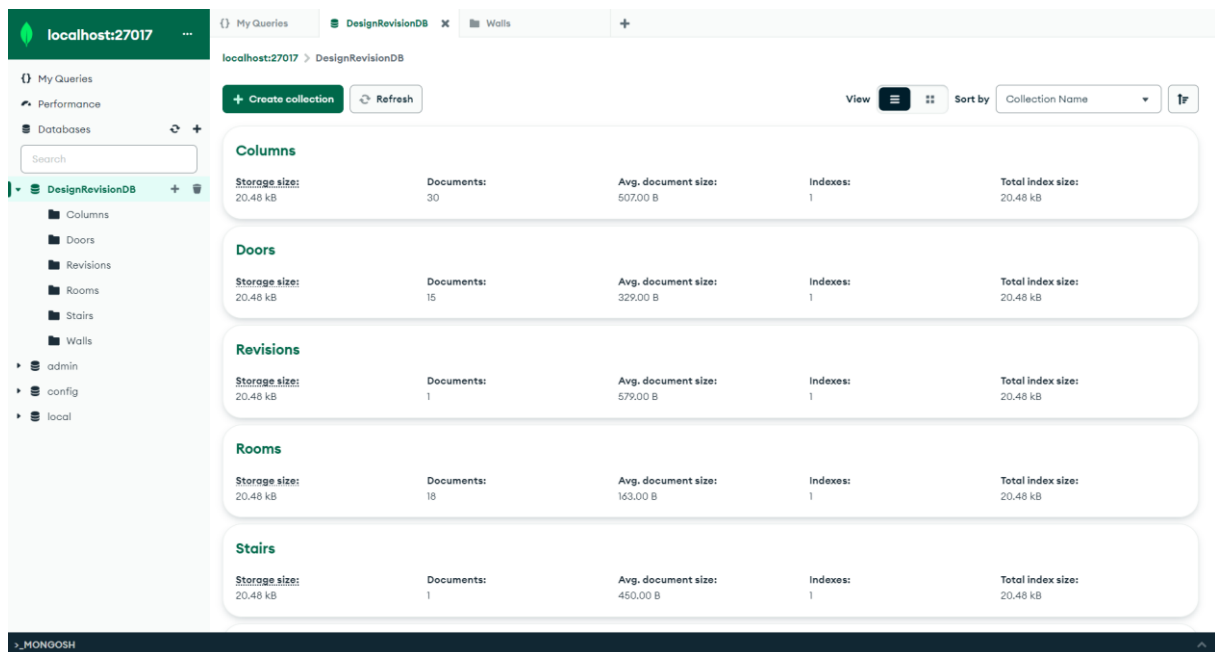


Figure 52: Documents and collections in MongoDB

Based on the relationship that a room can have multiple doors (Figure 20), we assign a property to the data model of room to indicate whether a room is accessible or not, which can be simply checked by the method IsRoomAccessible (Figure 42). Namely, a room is accessible if it has at least one door or at least one of the room boundary lines is a room separation line instead of a wall. The data model of door is assigned

with a property to store the rooms that it connects (Figure 55). Besides, it stores the GUID of its host wall as well. In terms of the data model of stair, a property is similarly assigned to indicate the accessibility of stairs, which is typically based on the methods in Figure 50. Moreover, it takes the opening above the stair into consideration as well (Figure 56). Regarding the implementation of the revision data model (Figure 59) stays the same as described in Figure 23.

```

35 #region Fields
36
37     13 references
38     public int Version { get; set; }
39     11 references
40     public bool IsDeleted { get; set; }
41     9 references
42     public bool IsStructural { get; set; }
43     5 references
44     public string OverlappingWall { get; set; }
45     7 references
46     public List<string> VerticallyConnectedStructWalls { get; set; }
47     7 references
48     public List<string> HorizontallyConnectedStructWalls { get; set; }
49     7 references
50     public List<string> ConnectedStructuralColumns { get; set; }
51     5 references
52     public List<string> ClashingStairs { get; set; }
53     5 references
54     public List<string> DependentRooms { get; set; }
55     5 references
56     public List<string> BlockedDoors { get; set; }
57     5 references
58     public List<string> BlockedStairs { get; set; }
59     5 references
60     public List<string> HorizontallyConnectedArchiWalls { get; set; }
61     4 references
62     public string FamilyType { get; set; }
63     4 references
64     public string Level { get; set; }
65
66     [BsonId]
67     17 references
68     public string Id { get; set; }
69     20 references
70     public List<int> RevisionCycle { get; set; }
71
72 #endregion

```

Figure 53: Data model of wall

```

25 #region Fields
26
27     13 references
28     public int Version { get; set; }
29     11 references
30     public bool IsDeleted { get; set; }
31     10 references
32     public bool IsStructural { get; set; }
33     7 references
34     public List<string> ConnectedStructuralColumns { get; set; }
35     5 references
36     public List<string> SurroundingStructuralColumns { get; set; }
37     5 references
38     public string OverlappingColumn { get; set; }
39     7 references
40     public List<string> ConnectedStructuralWalls { get; set; }
41     5 references
42     public List<string> ClashingDoors { get; set; }
43     5 references
44     public List<string> ClashingStairs { get; set; }
45     5 references
46     public List<string> BlockedStairs { get; set; }
47     5 references
48     public List<string> SurroundingArchiColumns { get; set; }
49     4 references
50     public string FamilyType { get; set; }
51     4 references
52     public string Level { get; set; }
53     23 references
54     public List<int> RevisionCycle { get; set; }
55
56     [BsonId]
57     17 references
58     public string Id { get; set; }
59
60 #endregion

```

Figure 54: Data model of column

```

35 #region Fields
36     7 references
37     public int Version { get; set; }
38     4 references
39     public bool IsDeleted { get; set; }
40     2 references
41     public List<string> FromToRooms { get; set; }
42     1 reference
43     public string HostWall { get; set; }
44     2 references
45     public string FamilyType { get; set; }
46     1 reference
47     public string Mark { get; set; }
48     2 references
49     public string Level { get; set; }
50
51     [BsonId]
52     4 references
53     public string Id { get; set; }
54     6 references
55     public List<int> RevisionCycle { get; set; }
56 #endregion

```

Figure 55: Data model of door

```

if ((blockingStructuralWallsAndColumns != null && blockingStructuralWallsAndColumns.Count > 0) ||
    (blockingArchitecturalWallsAndColumns != null && blockingArchitecturalWallsAndColumns.Count > 0) ||
    (topOpeningLines.Count < 3))
{
    isAccessible = false;
}
else
    isAccessible = true;

```

Figure 56: Check the accessibility of stairs

To implement a versioning system, a version number, whose initial value is set to 0 if no operations have been performed yet, is assigned to each building component at the point of its creation, representing the current version of the design. The current version number automatically increments itself by one when an operation is performed. For newly inserted components, the version number is synchronized with the current version of the database, while the version number of deleted items stops updating itself anymore. For example, the current version number for all existing components in the database should be five, if five operations have already been performed. Now, if a new wall needs to be inserted, the version of the wall should be six, meaning that this wall was created in the sixth operation. In addition, all other existing components in the database should increase their version numbers to six. Upon both creation and deletion, the version number of a component is to be recorded in a different property, namely the RevisionCycle property. The version property and the RevisionCycle property together can serve as a versioning system (Figure 57). Moreover, an additional property is assigned to components, indicating whether a component is currently deleted or not. The three versioning properties are often used in the implementation simultaneously as a means of double check, as illustrated in Figure 58. For instance, if a component is to be deleted in the upcoming operation, the database first increments

its version number by one and then marks it as deleted and thereafter, adds the version number to its RevisionCycle property, whereas no updates are possible anymore if it is already marked as deleted before an operation.

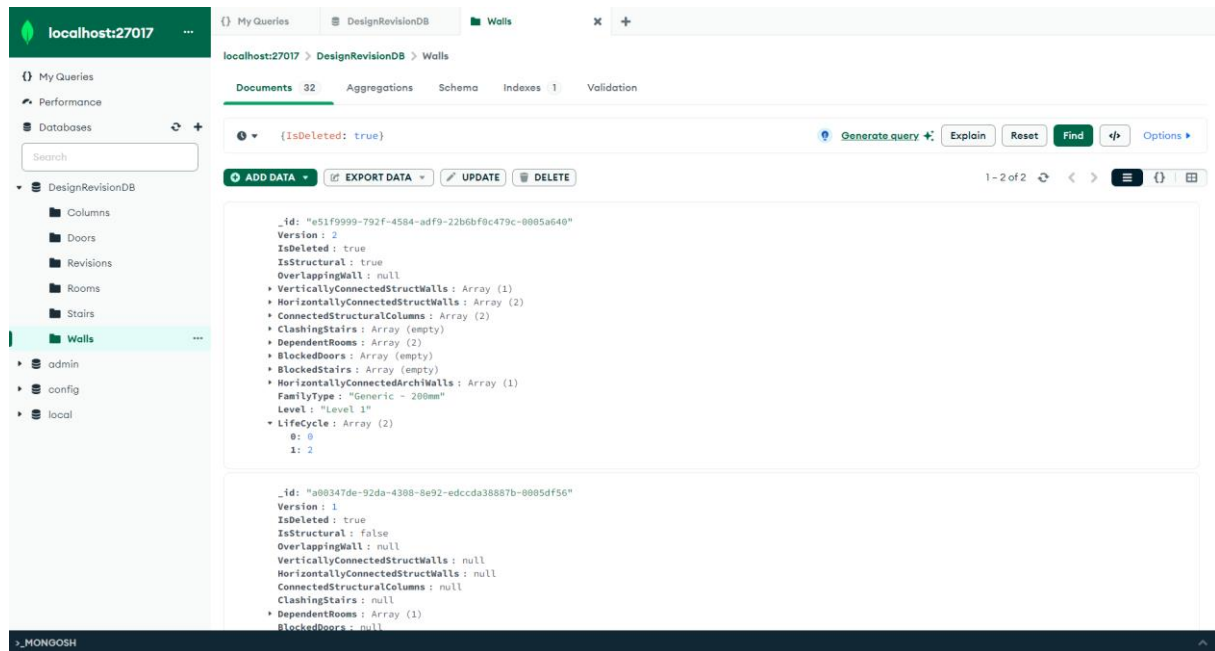


Figure 57: Versioning system for design revision

Based on the versioning system, it is now possible to keep track of design revisions. As illustrated in Figure 59, the identifier of documents in the revisions collection is still the GUID of building components from Revit. Namely, the identifier indicates the subject, on which the corresponding operation was performed. Additionally, the version property tells when the operation was conducted. The affected components in a revision are also easy to track with their stored GUIDs as reference.

```

52     var deletedStructWalls = wallCol.Find(wallFilter.And(
53         wallFilter.Eq(w => w.IsDeleted, true),
54         wallFilter.Eq(w => w.IsStructural, true),
55         wallFilter.SizeGt(w => w.RevisionCycle, 1),
56         wallFilter.Eq(w => w.Version, currentVersion),
57         wallFilter.Ne(w => w.Id, ComponentId))).ToList();
58
59     var deletedArchiWalls = wallCol.Find(wallFilter.And(
60         wallFilter.Eq(w => w.IsDeleted, true),
61         wallFilter.Eq(w => w.IsStructural, false),
62         wallFilter.SizeGt(w => w.RevisionCycle, 1),
63         wallFilter.Eq(w => w.Version, currentVersion),
64         wallFilter.Ne(w => w.Id, ComponentId))).ToList();
65
66     var deletedStructColumns = colCol.Find(columnFilter.And(
67         columnFilter.Eq(c => c.IsDeleted, true),
68         columnFilter.Eq(c => c.IsStructural, true),
69         columnFilter.SizeGt(c => c.RevisionCycle, 1),
70         columnFilter.Eq(c => c.Version, currentVersion),
71         columnFilter.Ne(c => c.Id, ComponentId))).ToList();
72
73     var deletedArchiColumns = colCol.Find(columnFilter.And(
74         columnFilter.Eq(c => c.IsDeleted, true),
75         columnFilter.Eq(c => c.IsStructural, false),
76         columnFilter.SizeGt(c => c.RevisionCycle, 1),
77         columnFilter.Eq(c => c.Version, currentVersion),

```

Figure 58: Using multiple versioning properties simultaneously in queries

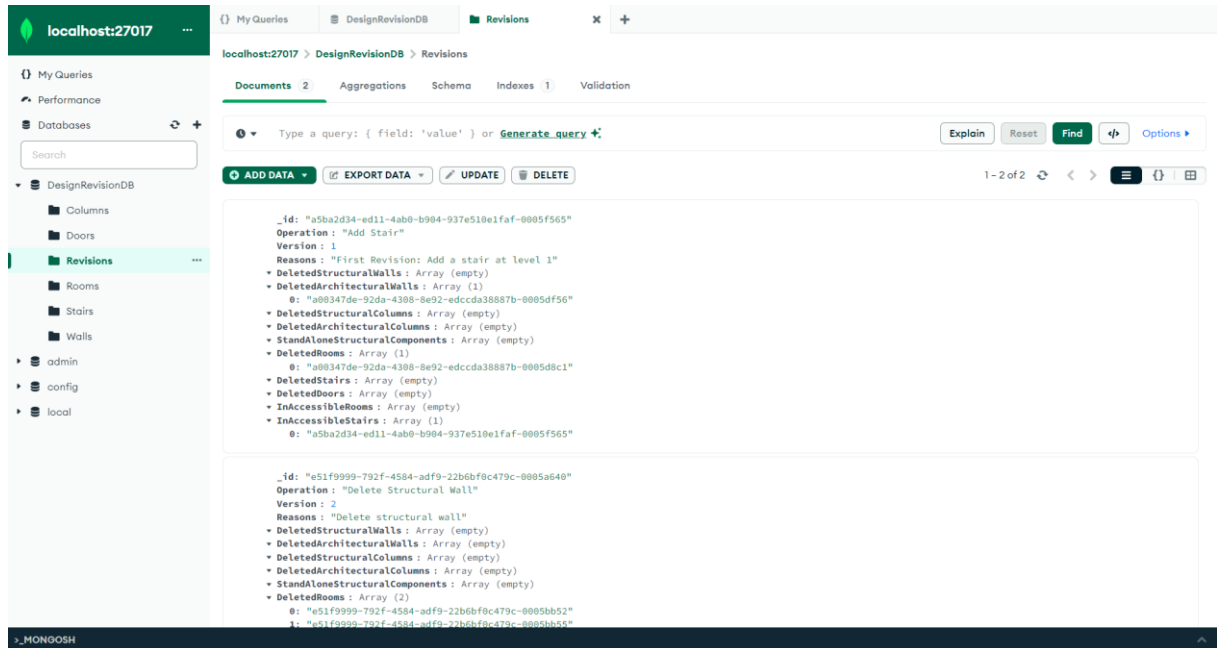


Figure 59: Track design revisions throughout the design process

4.4 Design Revision Manager (DRM)

A graphical user interface (GUI)-based Design Revision Manager (DRM) is implemented to realize a better interaction with the user while executing operations. The implementation was based on XAML, which is a declarative markup language used for creating user interface (UI) for .Net applications.

Design Revision Operations

Delete structural wall	Delete stair	Delete structural column
Add structural wall	Add stair	Add structural column

Element Properties:

Unique ID: **65d6fb67-5633-428f-9a1f-1f816b2d3ba5-0005a2ee**

Name: **Generic - 200mm** Revision Cycle:

Base Level: **Level 1** Top Level:

Reasons for selected operations:

First Op - Delete structural wall

Exit
Continue
Set Reasons

Change Propagation Results based on queries:

Operation: **Delete Structural Wall**

Version: **1**

Component ID: **65d6fb67-5633-428f-9a1f-1f816b2d3ba5-0005a2ee**

Deleted Structural Walls:	0	Deleted Structural Columns:	0
Deleted Architectural Walls:	0	Deleted Architectural Columns:	0
Stand-alone Structural Components:	0	Deleted Rooms:	3
Deleted Stairs:	0	Deleted Doors:	0

The number of stairs connecting the current level to the upper level:

Look up constraints:

Pre-Constraints	Post-Constraints
Current Version: 0	Current Version: 1
IsDeleted: False	IsDeleted: True
IsStructural: True	IsStructural:
IsAccessible:	IsAccessible:
Connected Struct Columns: 1	Connected Struct Columns:
Horizontally Connected Struct Walls: 1	Horizontally Connected Struct Walls:
Vertically Connected Struct Walls: 1	Vertically Connected Struct Walls:
Horizontally Connected Archi Walls: 1	Horizontally Connected Archi Walls:
Surrounding Struct Columns:	Surrounding Struct Columns:
Surrounding Archi Columns:	Surrounding Archi Columns:
Dependent Rooms: 3	Dependent Rooms:
Overlapping Wall:	Overlapping Wall:
Overlapping Column:	Overlapping Column:
Clashing Struct Walls and Columns:	Clashing Struct Walls and Columns:
Clashing Archi Walls and Columns:	Clashing Archi Walls and Columns:
Clashing Doors: 0	Clashing Doors:
Clashing Stairs: 0	Clashing Stairs:
Blocked Stairs: 0	Blocked Stairs:
Blocking Struct Walls and Columns:	Blocking Struct Walls and Columns:
Blocking Archi Walls and Columns:	Blocking Archi Walls and Columns:
FromToRooms:	FromToRooms:

Quick Commands based on queries of topological constraints from DB:

Structurally stand-alone components

Inaccessible stairs (clearance area of 1 square meter in front of the stair's entrance/exit is not guaranteed)

Inaccessible rooms (A room without any exit/entrance)

Figure 60: Design Revision Manager (design operations)

As illustrated in Figure 60, there are six gray buttons for performing revisions, corresponding to the six operations in Chapter 4.2. Before a command can be executed, the properties of the element and its pre-constraints, which are typically the relationships to other building components, are to be displayed, so that the user can gain knowledge of relevant dependencies before making modifications on the current BIM model. These relationships are based on the ER models proposed in Chapter 3.6. After the execution of an operation, the post-constraints of the component are to be displayed. Nevertheless, post-constraints only make sense for performing add-operations, since deleted components do not have constraints anymore. In addition, the results, namely the side effects of change propagation processes, will be shown as well. Moreover, it's allowed to set the reasons for the performed operation, which are to be written in the corresponding revision in the database.

The screenshot displays the 'Design Revision Operations' interface. At the top, there are six buttons for operations: 'Delete structural wall', 'Delete stair', 'Delete structural column', 'Add structural wall', 'Add stair', and 'Add structural column'. Below these are 'Element Properties' for a 'Stair' component, including its Unique ID, Name, Base Level, LifeCycle, and Top Level. A section for 'Reasons for selected operations' contains three buttons: 'Exit', 'Continue', and 'Set Reasons'. The 'Change Propagation Results based on queries:' section shows the operation and version details, along with a list of affected components like Deleted Structural Walls, Deleted Architectural Walls, Stand-alone Structural Components, Deleted Stairs, Deleted Rooms, Deleted Doors, Deleted Structural Columns, Deleted Architectural Columns, Deleted Rooms, and Deleted Doors. A 'Quick Commands based on queries of topological constraints from DB:' section is highlighted with a red box, listing three commands: 'Structurally stand-alone components' with a long ID, 'Inaccessible stairs (clearance area of 1 square meter in front of the stair's entrance/exit is not guaranteed)' with a shorter ID, and 'Inaccessible rooms (A room without any exit/entrance)'. On the right side, there are 'Pre-Constraints' and 'Post-Constraints' tables for the selected element, with a 'Look up constraints:' field at the top.

Figure 61: Design Revision Manager (topological constraint-based commands)

The DRM provides another three topological constraint-based commands, which are implemented through querying the relevant data from the design revision database. In other words, it allows a quick search for components which satisfy certain topological constraints. The commands are meant for providing information that's relevant for the design process, namely more of such commands can be implemented for various pur-

poses and projects. As illustrated in Figure 61, the structurally stand-alone components, inaccessible rooms and inaccessible stairs can be quickly found just with one click. These topological constraints are often easily overlooked since they need to be checked manually and BIM models from real projects are complex. With the found GUIDs, one can use the command “Look up constraints” to find the constraints of the corresponding problematic component. In addition, it’s also convenient to locate the component in a complex BIM model with its GUID. All in all, with this prototype, designers can find the components with “ill” topological constraints swiftly at any time. Besides, topological constraints of components can be looked up with their GUIDs.

5 Results and Discussion

5.1 Test Results of Design Revision Operations

This chapter presents the test results for both the proposed change management framework and the various functionalities of the developed prototype. Multiple ADD and DELETE operations were conducted, and their outcomes were evaluated to confirm that the Design Revision Manager (DRM) supports decision-making in building design revisions. Additionally, the complete workflow for each operation tested is illustrated with accompanying test data. Furthermore, we examined the effectiveness of the proposed versioning system.

5.1.1 Initial Test Model

The prerequisite for running the prototype is a BIM model, as stated before in Chapter 4.1. Since there's no existing database for the current BIM project, the prototype first creates a new database in MongoDB locally based on the database schema proposed previously and then exports the data of relevant building components from the BIM model to the database. The first time when the prototype is fired up, five collections are to be created, corresponding to the five categories of building components involved in change propagation, namely columns, walls, stairs, doors, and rooms. For columns and walls, both structural and architectural ones are considered. Nevertheless, no operations have been performed till this point yet. Besides, every document from the collections, namely every building component with its topological constraints as attributes, has the same version number of 0, as illustrated in both Figure 62 and Table 4. It is namely the initial state of the BIM model for testing, based on which several operations are to be performed in the following steps.

Table 4: The number of building components (initial state of the test model)

	Structural wall	Architectural wall	Structural column	Architectural column	Door	Room	Stair
Version 0	19	11	24	1	15	16	1

My Queries
Performance
Databases
DesignRevisionDB
Columns
Doors
Rooms
Stairs
Walls
admin
config
local

Documents 25 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

ADD DATA EXPORT DATA UPDATE DELETE

```
{
  "_id": "fa0694c5-d6db-4e05-a205-beb7092add32-0005654d"
  "Version": 0
  "IsDeleted": false
  "IsStructural": true
  "ConnectedStructuralColumns": Array (1)
  "SurroundingStructuralColumns": Array (empty)
  "OverlappingColumn": null
  "ConnectedStructuralWalls": Array (2)
  "ClashingDoors": Array (empty)
  "ClashingStairs": Array (empty)
  "BlockedStairs": Array (empty)
  "SurroundingArchiColumns": Array (empty)
  "FamilyType": "12 x 12"
  "Level": "Level 1"
  "RevisionCycle": Array (1)
}
```

```
{
  "_id": "fa0694c5-d6db-4e05-a205-beb7092add32-00056639"
  "Version": 0
  "IsDeleted": false
  "IsStructural": true
  "ConnectedStructuralColumns": Array (1)
  "SurroundingStructuralColumns": Array (empty)
  "OverlappingColumn": null
  "ConnectedStructuralWalls": Array (1)
  "ClashingDoors": Array (empty)
  "ClashingStairs": Array (empty)
  "BlockedStairs": Array (empty)
  "SurroundingArchiColumns": Array (empty)
  "FamilyType": "12 x 12"
}
```

Figure 62: Initial state (version 0) of test model

5.1.2 Structural Wall Deletion

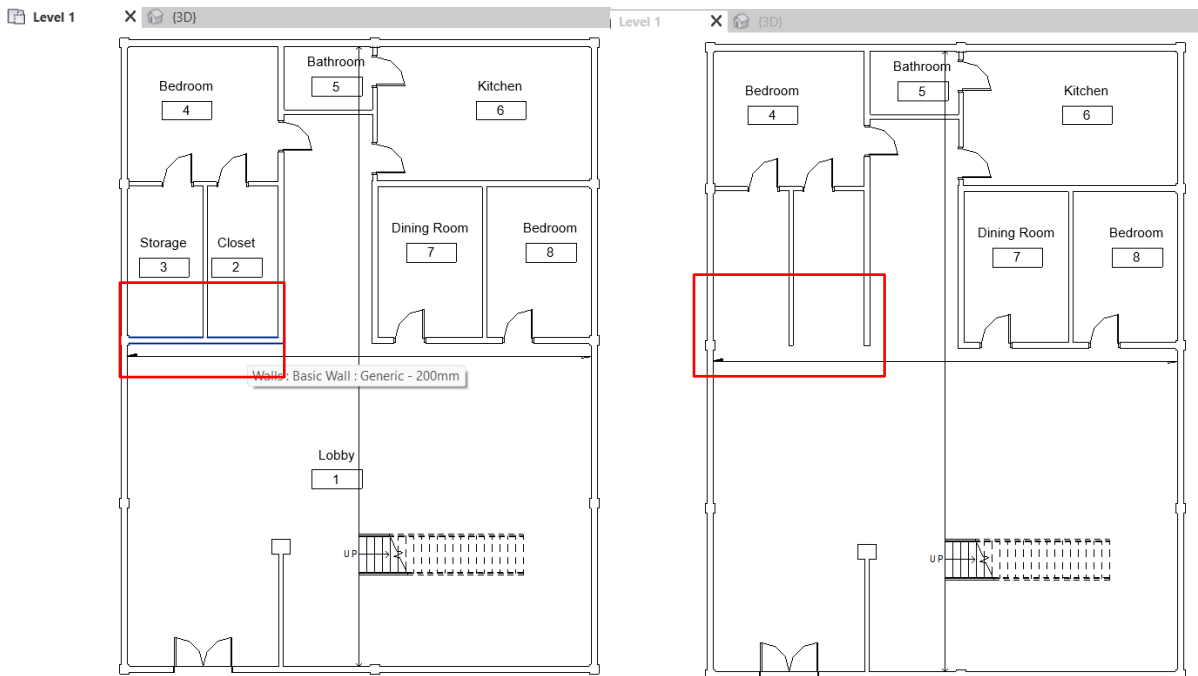


Figure 63: First operation – Structural Wall Deletion (the deleted wall resides in Level 1, the figure on the left is the original status, while the one on the right is the status after the DELETE operation)

The first operation was to delete a structural wall based on the default critical conditions (Figure 26 & Figure 63). The critical condition for the deletion of a structural wall is the number of vertically connecting structural walls, which is 2 by default. Namely, if there are at least two connecting structural walls below or above from other levels, the deletion operation is not possible, and the operation will be therefore terminated. Since there are only two floors in the test model, meaning that there is maximum only one vertically connecting structural wall, the deletion of the selected structural wall is permitted by default. Before the command is executed, the DRM can give us an overview of the current constraints of the target wall (pre-constraints), which can help to decide if to continue with the operation or not (Figure 65). After performing the operation, a new collection named Deleted Revisions is created in the database (Figure 64), aiming at recording the latest operation and the effects of change propagation, e.g. three rooms were deleted in this test case. The results of change propagation are also to be displayed on the DRM by simply querying data that's stored in the Revisions collection and also related to the first revision, as illustrated in the section of Change Propagation Results in Figure 65. Additionally, the reasons for the revision were recorded in the database by using the SetReasons command as well. For the deleted structural wall here, its version number stops updating ever since. Moreover, the wall was marked as deleted and the version number, in which the wall was deleted, was documented in the RevisionCycle property as well, representing that the wall was created at version 0 and then deleted at version 1 (Figure 64).

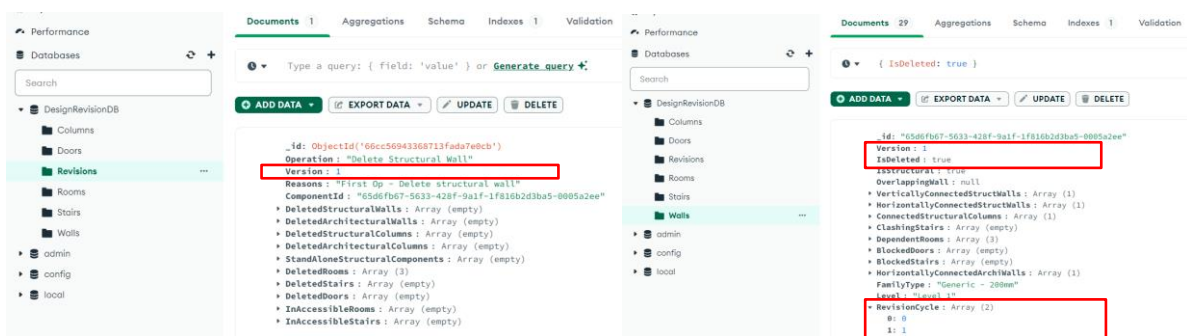


Figure 64: 1st version of test model (the left figure presents that the operation is documented in the Revisions collection, while the right one shows the updates of the deleted wall's properties)

Design Revision Operations

Delete structural wall	Delete stair	Delete structural column
Add structural wall	Add stair	Add structural column

Element Properties:
 Unique ID: **65d6fb67-5633-428f-9a1f-1f816b2d3ba5-0005a2ee**
 Name: **Generic - 200mm** Revision Cycle:
 Base Level: **Level 1** Top Level:

Reasons for selected operations:
 First Op - Delete structural wall

Exit
 Continue
 Set Reasons

Change Propagation Results based on queries:

Operation: **Delete Structural Wall**
 Version: **1**
 Component ID: **65d6fb67-5633-428f-9a1f-1f816b2d3ba5-0005a2ee**

Deleted Structural Walls:	0	Deleted Structural Columns:	0
Deleted Architectural Walls:	0	Deleted Architectural Columns:	0
Stand-alone Structural Components:	0	Deleted Rooms:	3
Deleted Stairs:	0	Deleted Doors:	0

The number of stairs connecting the current level to the upper level.

Pre-Constraints

Current Version:	0
IsDeleted:	False
IsStructural:	True
IsAccessible:	
Connected Struct Columns:	1
Horizontally Connected Struct Walls:	1
Vertically Connected Struct Walls:	1
Horizontally Connected Archi Walls:	1
Surrounding Struct Columns:	
Surrounding Archi Columns:	
Dependent Rooms:	3
Overlapping Wall:	
Overlapping Column:	
Clashing Struct Walls and Columns:	
Clashing Archi Walls and Columns:	
Clashing Doors:	0
Clashing Stairs:	0
Blocked Stairs:	0
Blocking Struct Walls and Columns:	
Blocking Archi Walls and Columns:	
FromToRooms:	

Post-Constraints

Current Version:	1
IsDeleted:	True
IsStructural:	
IsAccessible:	
Connected Struct Columns:	
Horizontally Connected Struct Walls:	
Vertically Connected Struct Walls:	
Horizontally Connected Archi Walls:	
Surrounding Struct Columns:	
Surrounding Archi Columns:	
Dependent Rooms:	
Overlapping Wall:	
Overlapping Column:	
Clashing Struct Walls and Columns:	
Clashing Archi Walls and Columns:	
Clashing Doors:	
Clashing Stairs:	
Blocked Stairs:	
Blocking Struct Walls and Columns:	
Blocking Archi Walls and Columns:	
FromToRooms:	

Quick Commands based on queries of topological constraints from DB:

Structurally stand-alone components

Inaccessible stairs (clearance area of 1 square meter in front of the stair's entrance/exit is not guaranteed)

Inaccessible rooms (A room without any exit/entrance)

Figure 65: Design Revision Manager – Structural Wall Deletion (pre-constraints are the dependencies of building components before the operation is performed)

5.1.3 Structural Column Deletion

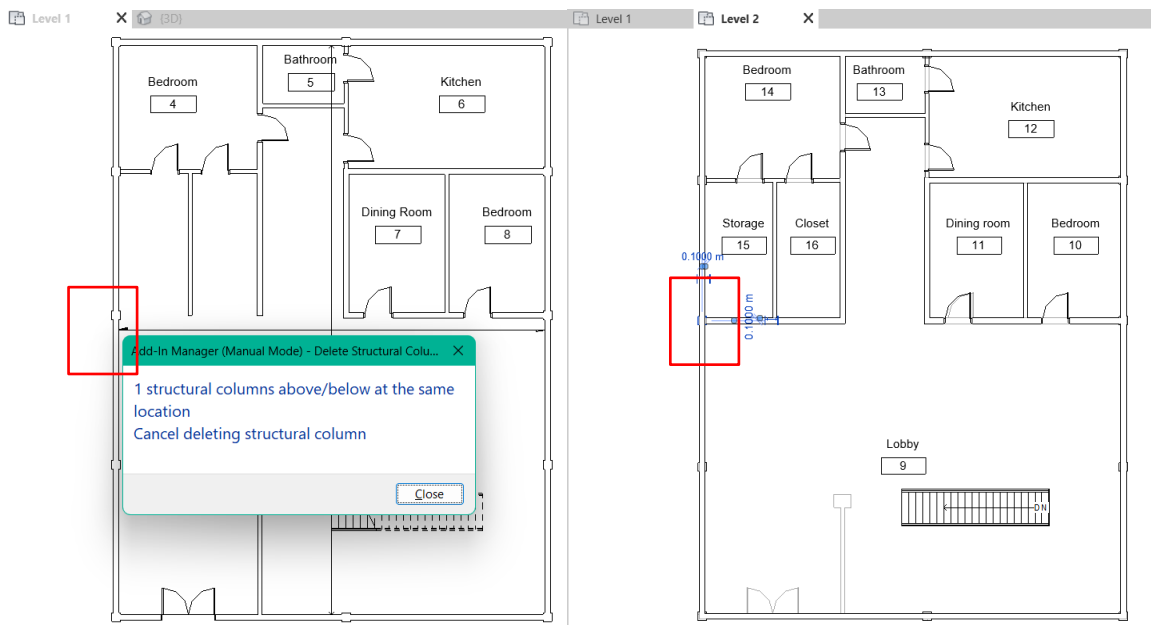


Figure 66: Second operation – Structural Column Deletion (the to-be-deleted structural column from different views, namely Level 1 in the left figure and Level 2 in the right one)

The second operation aimed to delete a structural column, whose execution was subjected to one of the critical conditions as well. However, different from the condition for deleting structural walls, the number was set to 1 by default. Namely, if there is at least

one connecting structural column below or above, the intended operation will be automatically terminated, and no change propagation is possible. Since the to-be-deleted structural column at level 1 had a connection to one structural column from level 2 (Figure 66), it's not possible to perform the deletion operation. Therefore, the test model itself as well as the data stored in the database stayed unchanged, meaning that no updates of the test model were possible regardless of all other pre-constraints.

5.1.4 Stair Deletion & Addition

In this test, we intended to first relocate an existing stair, which comprises a DELETE and an ADD operation. Afterwards we inserted a new stair, resulting in 2 stairs in total in the project.

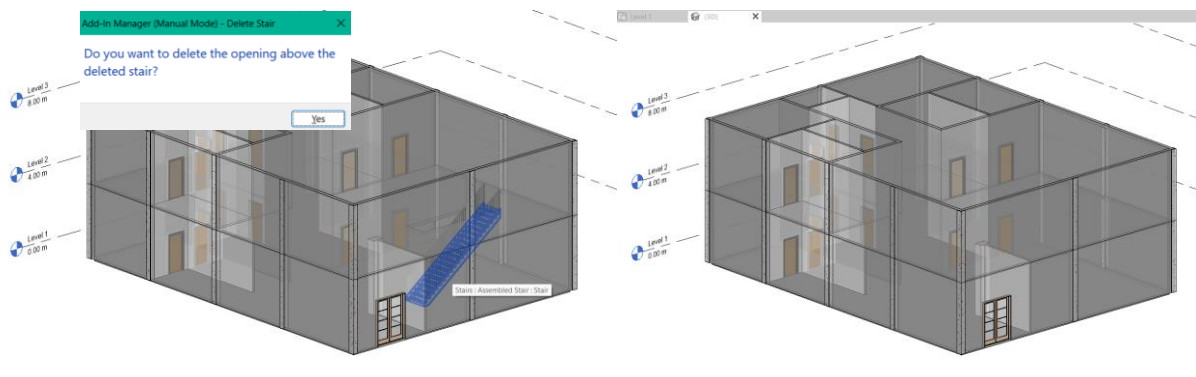


Figure 67: Third operation – Stair Deletion (the figure on the left is the original status, containing the to-be-deleted stair with an opening on top, the right one is the updated status after the DELETE operation)

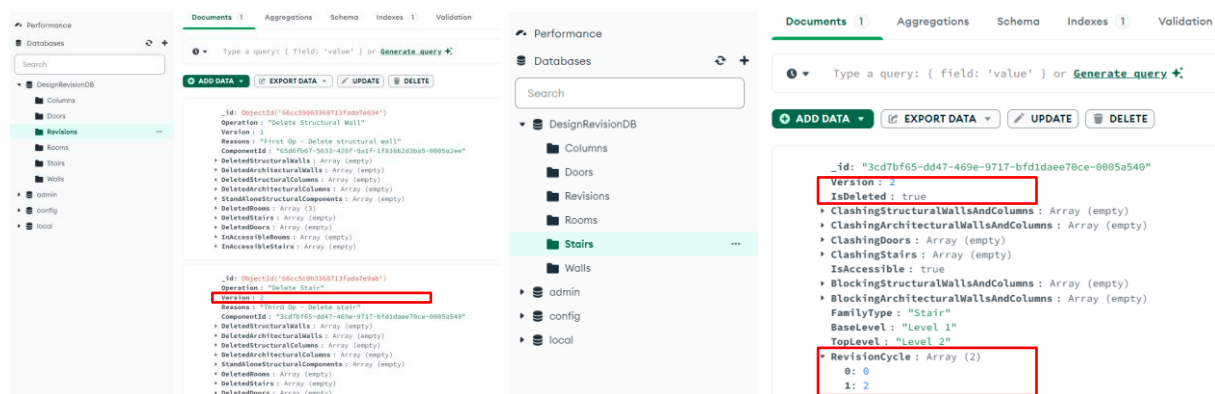


Figure 68: 2nd version of test model (the left figure presents that the operation is documented in the Revisions collection, while the right one shows the updates of the deleted stair's properties)

For relocating a stair, we first deleted the target stair in the project. Since there's no significant violation of pre-constraints, the deletion operation can be performed. As a result of change propagation, the opening of the floor above the stair was deleted by choice as well (Figure 67). Consequently, an update of the version number to 2 was committed for every existing building component in the database (Figure 68). After the

deletion, the `IsDeleted` and `RevisionCycle` properties of the deleted stair were updated accordingly, which is similar to the deletion of structural walls. With the updated properties, the stair is prevented from any further updates. The 2nd revision was also added to the `Revisions` collection (Figure 68). In addition, there's no stair that can establish a connection between the ground floor and the 1st floor, resulting in inaccessible areas.

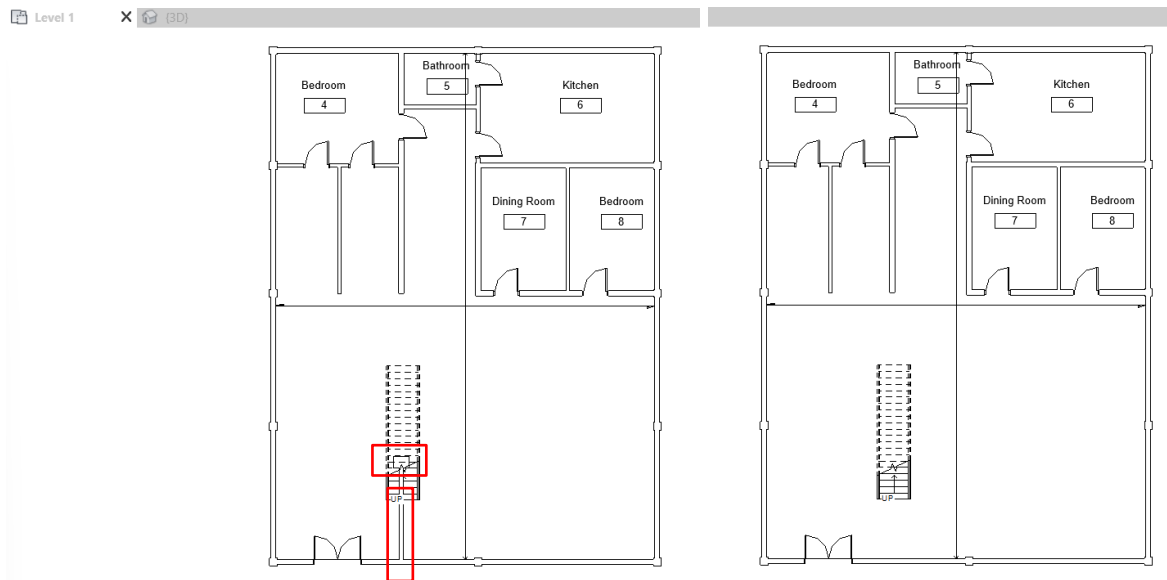


Figure 69: Fourth operation – Stair Addition (the left figure shows an intended insertion of a stair at a desired location by picking two points, the right one shows the updated status after adding the stair)

Following is the fourth operation, which is to add a stair by directly picking two points on the ground floor. Since the stair was added at the 3rd version, the first entry of the `RevisionCycle` property is 3 (Figure 70). In addition, the direction of the new stair can be determined ourselves, namely the first picked point is to be located at the base level while the second picked point is at the top level. As illustrated in Figure 69, the to-be-inserted stair has conflicts with an architectural wall and an architectural column respectively. Since it's of greater importance to guarantee the accessibility of stairs, the obstacles were deleted automatically, meaning that the clashing components are of less significance compared to stairs and therefore, can be deleted without confirmation. This information can be utilized by the DRM to help with decision-making.

Because of the existence of clashing components, which block the exit or entrance of the stair, the `IsAccessible` property was marked as `False` (Figure 70). With the pre-constraints provided by the DRM, which indicates that there's no conflict or connection between the new stair and structural components, a conclusion can be drawn. Namely, the clashes result from conflicts with non-structural components, which normally require no further confirmation to resolve these conflicts. As a result, the clashes were

addressed automatically, and the accessibility of the new stair can be maintained after the ADD operation. In addition, the effects resulting from the operation were also provided by the DRM, namely the operation resulted in the deletion of one architectural wall and one architectural column (Figure 71). What’s more, the accessibility between two floors was evaluated as well by counting the existing stairs which are supposed to connect the two stories.

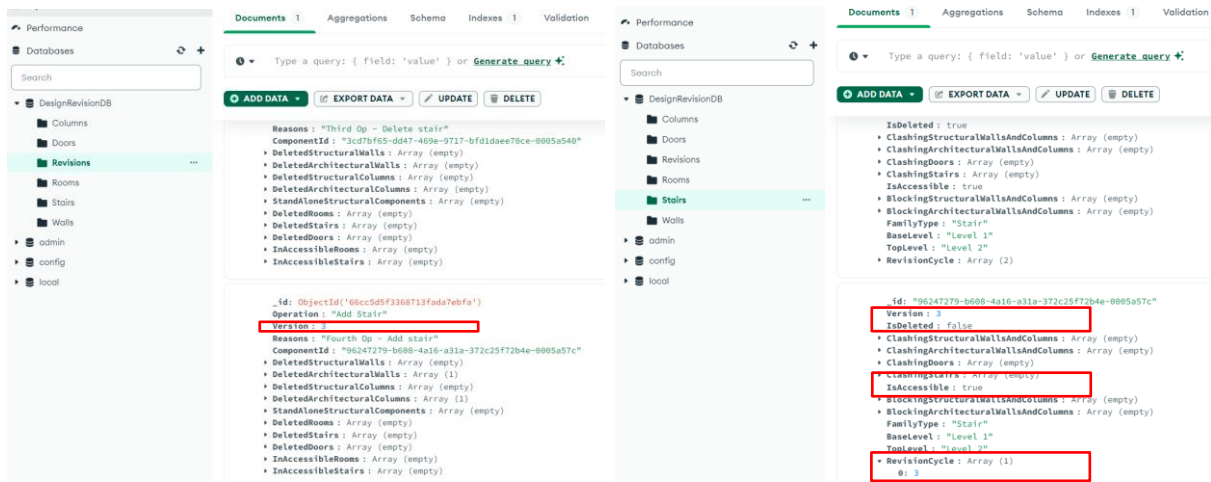


Figure 70: 3rd version of test model (the left figure presents that the operation is documented in the Revisions collection, while the right one shows the updates of the inserted stair’s properties)

Change Propagation Results based on queries:

Operation: **Add Stair**

Version: **3**

Component ID: **96247279-b608-4a16-a31a-372c25f72b4e-0005a57**

Deleted Structural Walls:	0	Deleted Structural Columns:	0
Deleted Architectural Walls:	1	Deleted Architectural Columns:	1
Stand-alone Structural Components:	0	Deleted Rooms:	0
Deleted Stairs:	0	Deleted Doors:	0
The number of stairs connecting the current level to the upper level:			1

Figure 71: Design Revision Manager – Stair Addition (change propagation results)

The next operation was to insert a new stair at a desired location. However, this ADD operation had conflicts with two structural walls, which had the same location line but reside in different stories, as illustrated in Figure 72. Since the maximum number of intersecting structural components allowed for an ADD operation was set to 2 (Figure 26), the operation was automatically denied and no changes were committed as a result, neither in the test model nor in the design revision database. Therefore, the intended insertion of the marked stair in Figure 72 was discarded by the DRM.

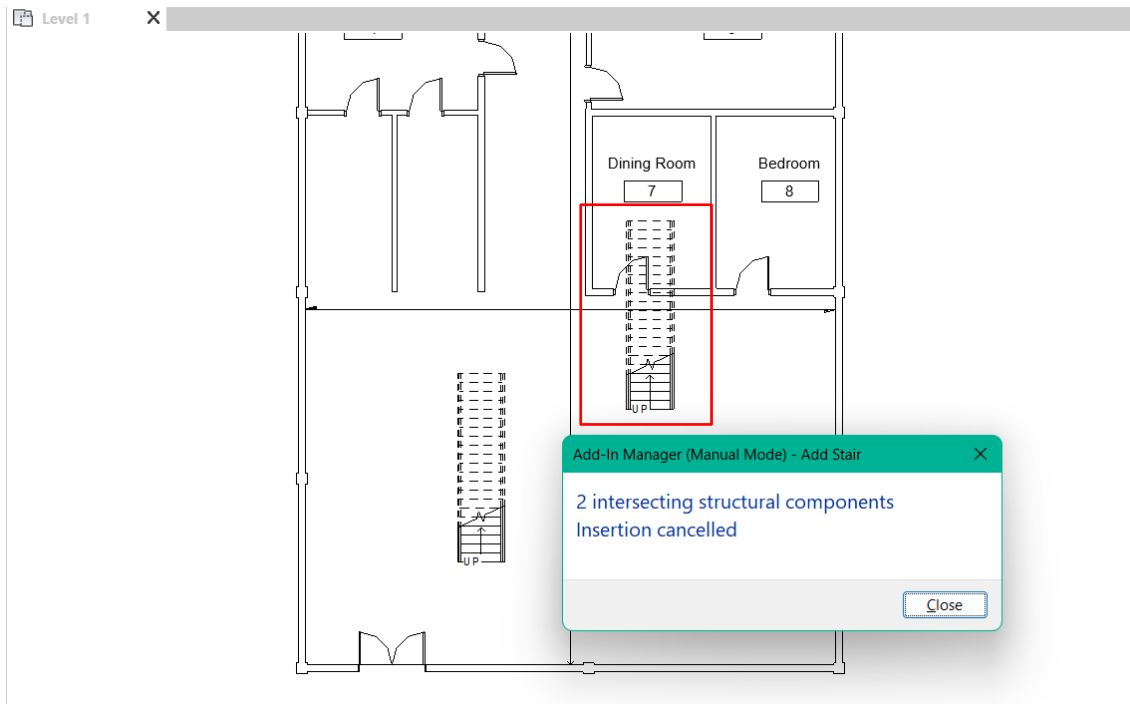


Figure 72: Fifth operation – Stair Addition

Table 5: The number of building components (3rd version)

	Structural wall	Architectural wall	Structural column	Architectural column	Door	Room	Stair
Version 0	19	11	24	1	15	16	1
Version 1	18	11	24	1	15	13	1
Version 2	18	11	24	1	15	13	0
Version 3	18	10	24	0	15	13	1

5.2 Test Results of Topological Constraint-based Commands

We performed two more operations so that the test model is more suitable for testing the three topological constraint-based commands (Figure 61), which are based on querying data stored in the design revision database.

Different from the change propagation results, which present themselves as various numbers of different types of building component, the constraint-based commands rather consider multiple topological constraints simultaneously. With the DRM, we can easily find the structurally stand-alone structural components, inaccessible rooms and inaccessible stairs. The results are based on queries on the database and therefore totally independent from the test model itself. Namely, the dependencies are separated from the test model to a certain extent. In addition, the active constraints of a specific

component can be queried by using the Look-up-constraints command. As illustrated in Figure 74, one inaccessible stair was found through the quick commands and then its constraints were queried from the database with its GUID. The post-constraints, namely the currently active ones, indicate that a structural wall or a structural column blocks the stair. Since connections between components are established by referencing their GUIDs, we can easily locate this problematic dependency in the design revision database (Figure 75). Similarly, we can find this conflict in the BIM model easily with GUIDs and then find out how to resolve the conflict.

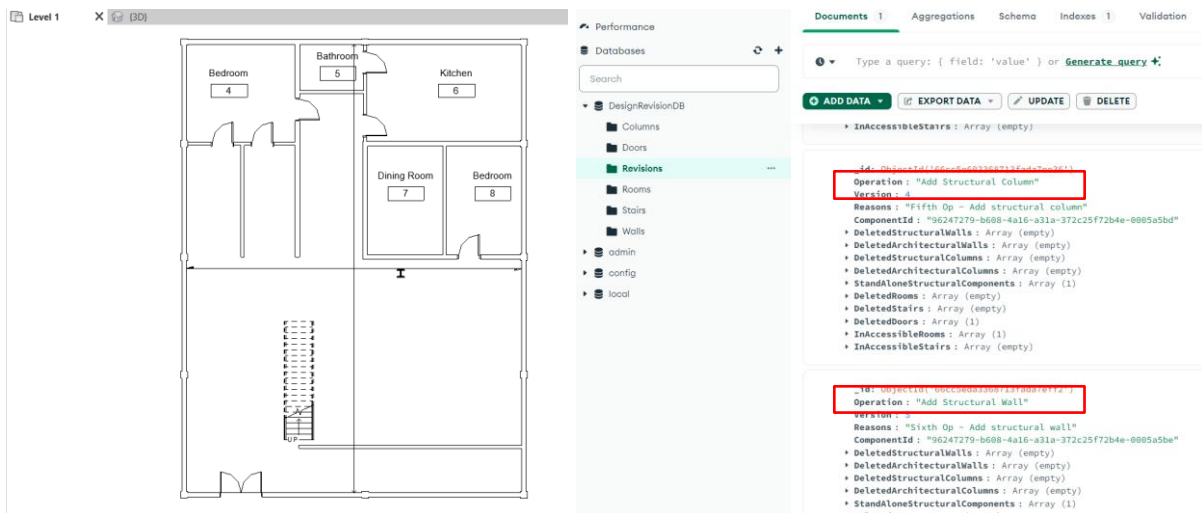


Figure 73: The latest state of test model (5th version of test model on the left and the corresponding database on the right)

Table 6: The number of building components (5th version)

	Structural wall	Architectural wall	Structural column	Architectural column	Door	Room	Stair
Version 0	19	11	24	1	15	16	1
Version 1	18	11	24	1	15	13	1
Version 2	18	11	24	1	15	13	0
Version 3	18	10	24	0	15	13	1
Version 4	18	10	25	0	14	13	1
Version 5	19	10	25	0	14	13	1

Design Revision Operations

Delete structural wall	Delete stair	Delete structural column
Add structural wall	Add stair	Add structural column

Element Properties:
Unique ID:
Name: **Stair** Revision Cycle:
Base Level: **Level 1** Top Level: **Level 2**

Reasons for selected operations:

Exit Continue Set Reasons

Change Propagation Results based on queries:

Operation:
Version:
Component ID:
Deleted Structural Walls: Deleted Structural Columns:
Deleted Architectural Walls: Deleted Architectural Columns:
Stand-alone Structural Components: Deleted Rooms:
Deleted Stairs: Deleted Doors:
The number of stairs connecting the current level to the upper level:

Quick Commands based on queries of topological constraints from DB:

Structurally stand-alone components
96247279-b608-4a16-a31a-372c25f72b4e-0005a5bd;
Inaccessible stairs (clearance area of 1 square meter in front of the stair's entrance/exit is not guaranteed)
96247279-b608-4a16-a31a-372c25f72b4e-0005a57c;
Inaccessible rooms (A room without any exit/entrance)
fa0694c5-d6db-4e05-a205-beb7092add32-0005a0b6;

Look up constraints: 279-b608-4a16-a31a-372c25f72b4e-0005a57c

Pre-Constraints	Post-Constraints
Current Version:	Current Version: 5
IsDeleted:	IsDeleted: False
IsStructural:	IsStructural:
IsAccessible:	IsAccessible: False
Connected Struct Columns	Connected Struct Columns:
Horizontally Connected Struct Walls:	Horizontally Connected Struct Walls:
Vertically Connected Struct Walls:	Vertically Connected Struct Walls:
Horizontally Connected Archi Walls:	Horizontally Connected Archi Walls:
Surrounding Struct Columns:	Surrounding Struct Columns:
Surrounding Archi Columns:	Surrounding Archi Columns:
Dependent Rooms:	Dependent Rooms:
Overlapping Wall:	Overlapping Wall:
Overlapping Column:	Overlapping Column:
Clashing Struct Walls and Columns:	Clashing Struct Walls and Columns: 0
Clashing Archi Walls and Columns:	Clashing Archi Walls and Columns:
Clashing Doors:	Clashing Doors: 0
Clashing Stairs:	Clashing Stairs: 0
Blocked Stairs:	Blocked Stairs:
Blocking Struct Walls and Columns:	Blocking Struct Walls and Columns: 1
Blocking Archi Walls and Columns:	Blocking Archi Walls and Columns: 0
FromToRooms:	FromToRooms:

Figure 74: Test results of topological constraint-based commands

Search

- DesignRevisionDB
 - Columns
 - Doors
 - Revisions
 - Rooms
 - Stairs**
 - Walls
- admin
- config
- local

Type a query: { field: 'value' } or [Generate query](#)

ADD DATA EXPORT DATA UPDATE DELETE

```
IsDeleted : true
  ▶ ClashingStructuralWallsAndColumns : Array (empty)
  ▶ ClashingArchitecturalWallsAndColumns : Array (empty)
  ▶ ClashingDoors : Array (empty)
  ▶ ClashingStairs : Array (empty)
IsAccessible : true
  ▶ BlockingStructuralWallsAndColumns : Array (empty)
  ▶ BlockingArchitecturalWallsAndColumns : Array (empty)
FamilyType : "Stair"
BaseLevel : "Level 1"
TopLevel : "Level 2"
  ▶ RevisionCycle : Array (2)

_id : "96247279-b608-4a16-a31a-372c25f72b4e-0005a57c"
Version : 5
IsDeleted : false
  ▶ ClashingStructuralWallsAndColumns : Array (empty)
  ▶ ClashingArchitecturalWallsAndColumns : Array (empty)
  ▶ ClashingDoors : Array (empty)
  ▶ ClashingStairs : Array (empty)
IsAccessible : false
  ▶ BlockingStructuralWallsAndColumns : Array (1)
    0 : "96247279-b608-4a16-a31a-372c25f72b4e-0005a5be"
  ▶ BlockingArchitecturalWallsAndColumns : Array (empty)
FamilyType : "Stair"
BaseLevel : "Level 1"
TopLevel : "Level 2"
  ▶ RevisionCycle : Array (1)
```

Figure 75: Find the problematic dependency with GUID (the red marked GUID refers to the inaccessible stair found in Fig. 73, the blue marked GUID is the structural wall or column that blocks the exit or entrance of the stair)

5.3 Key Findings

The proposed BIM-based change management framework has been tested and demonstrated its effectiveness in supporting building design revisions. A core component of this system is the change propagation mechanism, which outlines the operational principles of the Design Revision Manager (DRM). This mechanism can be tailored to different projects, accommodating varying requirements for change propagation based on specific design contexts.

Moreover, the system exhibits a high degree of adaptability, as the change propagation mechanism can be applied to most BIM-based building design processes. The definitions of these mechanisms rely on common engineering knowledge in building design, ensuring that many relationships between building components are universally applicable. The change propagation mechanism primarily aids decision-making regarding design revisions by evaluating the potential impacts on components affected by an initial change. Consequently, designers can determine whether to proceed with a proposed revision based on the severity of its effects on related building components.

An additional significant feature of the framework is the versioning system, which functions as an engineering database designed to track the current state of a BIM project. This system provides crucial information about the existence and topological constraints of building components within a BIM model, ensuring that designers always have access to relevant dependencies among components. Furthermore, design revisions throughout the entire design process can be documented in the database, allowing past revisions and their propagation outcomes to inform decision-making for similar design revisions in the future.

In summary, the proposed framework serves as a comprehensive change management system that enhances decision-making in building design revisions by supplying pertinent information regarding change propagation results, topological constraints of building components, and historical design revisions. Additionally, the prototype illustrates how to leverage engineering knowledge—specifically, the dependencies between building components—to achieve design automation.

6 Conclusion

Building design is inherently complex and uncertain, making design revisions an unavoidable aspect of the process. Specifically, revisions in Building Information Modeling (BIM)-based design are significantly more intricate due to the rich information embedded in BIM models. This study proposes a novel change management framework to support revisions in BIM-based building design, providing a comprehensive solution for managing change propagation.

The findings of this research validate several key aspects of the proposed change management system. Firstly, it has been confirmed that expert knowledge in building design can be effectively translated into topological constraints, which govern the direction of change propagation. Furthermore, change propagation processes, facilitated by these topological constraints, can be integrated into BIM-based design operations through a dedicated change propagation mechanism. This mechanism consists of a set of pre-defined topological constraints and a significance matrix that dictates the extent of change propagation, enabling prior evaluation of potential propagation processes while conducting design operations.

Secondly, the change propagation mechanisms exhibit flexibility, allowing them to be defined and adjusted according to various design contexts. The complexity of these mechanisms correlates with the number of building components and their respective dependencies involved in the propagation process. Nevertheless, these mechanisms are broadly applicable across most building design projects, as they typically share common disciplinary knowledge. Lastly, having access to the dependency network of building components and evaluating potential change propagation processes prior to executing design operations significantly enhances decision-making capabilities.

This research also contributes to the field of building design automation by addressing change propagation dynamically during design operations. The outcomes of propagation processes can be implemented automatically with a single click by the designer. However, a notable limitation of this study is that the proposed approach is restricted to Autodesk Revit and the Revit API, with a limited scope of considered topological constraints.

6.1 Contribution

This research contributes significantly to the field of change management in BIM-based building design. The proposed BIM-based change management system introduces a change propagation mechanism that aids decision-making throughout the building design process. The mechanism primarily facilitates the integration of change propagation processes into design operations within BIM authoring tools. Additionally, it serves as a tool for tracking design revisions throughout the entire design lifecycle. Its inherent flexibility and adaptability stem from its reliance on topological constraints and a corresponding significance matrix, both derived from expert knowledge in building design. This means that the behavior of change propagation can be tailored to suit different design contexts.

Furthermore, the Design Revision Manager serves as the practical implementation of the proposed change management framework. Overall, the prototype enhances decision-making in BIM-based building design revisions by providing insights into the dependencies between building components, the effects of change propagation resulting from intended design operations, and records of past revisions. By presenting the potential consequences of each decision in advance, the system enables more informed and rational decision-making.

6.2 Limitation

This research has not yet fully investigated the dependencies among building components, as it focused on a limited set of components. An increased number of components could yield a more complex network of dependencies, resulting in more sophisticated change propagation processes and potentially additional levels of change propagation.

Additionally, the range of design operations implemented in the prototype is limited. Currently, the system only supports the addition and deletion of individual building components. More complex design operations, such as rotation or modification of semantic properties, are not yet available. These functionalities could be realized by considering a broader spectrum of dependencies among building components.

Furthermore, the design process experiences interruptions due to user interactions required during the execution of design operations. To enhance the efficiency of the design workflow, the change propagation mechanism could be refined to allow

operations to run in the background, thereby facilitating a more seamless design experience.

Finally, the versioning system is restricted to maintaining records of the latest design version, with no access to previous versions. The ability to compare different versions could significantly contribute to change management. Moreover, the current versioning system operates only on a local host, limiting collaborative efforts among team members via a server connection.

6.3 Future Work

Based on the findings of this research, several recommendations can be made for future enhancements in this field. First, it is essential to conduct a more comprehensive investigation into the dependency network of building components. Specifically, a broader range of building components and their interrelationships should be considered in the context of change propagation. This expanded scope would facilitate a more thorough understanding of the factors influencing building design revisions, ultimately leading to more accurate and desired outcomes in change propagation processes. However, it is important to note that as the number of components and dependencies increases, the complexity of the corresponding change propagation mechanism will also rise, necessitating greater effort in its development.

Second, there is potential for further refinement of the proposed change propagation mechanism. This research currently focuses on two fundamental design revision operations: ADD and DELETE. Furthermore, the effects of change propagation primarily pertain to the deletion of components. Future research should explore the implementation of additional operations, such as modifying component attributes rather than solely deleting or adding components. Additionally, various methods of propagating changes should be examined, including the relocation of components and the modification of geometric or semantic properties.

Lastly, the database infrastructure should be enhanced in alignment with the increased number of building components and dependencies. Furthermore, the versioning system could be improved to maintain records of every version of a BIM project. Given that BIM projects typically involve numerous participants from diverse disciplines, establishing a central server for the database becomes critical. This can be effectively accomplished using Node.js, a cross-platform and open-source JavaScript runtime environment suitable for developing servers and web applications.

Bibliography

- Borrmann, A., König, M., Koch, C., & Beetz, J. (2018). *Building information modeling: Why? what? how?* (pp. 1-24). Springer International Publishing. https://doi.org/10.1007/978-3-319-92862-3_1
- Borrmann, A., & Rank, E. (2009). Specification and implementation of directional operators in a 3D spatial query language for building information models. *Advanced Engineering Informatics*, 23(1), 32-44. <https://doi.org/10.1016/j.aei.2008.06.005>
- National Institute of Building Sciences. (2015). *National BIM Standard United States version 3*. Washington, DC: National Institute of Building Sciences. <http://www.nationalbimstandard.org/> (Accessed December 9, 2017).
- Volk, R., Stengel, J., & Schultmann, F. (2014). Building Information Modeling (BIM) for existing buildings—Literature review and future needs. *Automation in construction*, 38, 109-127. <https://doi.org/10.1016/j.autcon.2013.10.023>
- Eastman, C. M. (2011). *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors*. John Wiley & Sons. https://www.academia.edu/download/31053339/BIM_Handbook_1st.pdf
- Wong, J., & Yang, J. (2010). Research and application of building information modeling (BIM) in the architecture, engineering and construction (AEC) industry: a review and direction for future research. In *Proceedings of the 6th International Conference on Innovation in Architecture, Engineering and Construction (AEC)* (pp. 356-365). Department of Civil and Building Engineering, Loughborough University. <https://eprints.qut.edu.au/38333/1/38333.pdf>
- Eastman, C. M. (2018). *Building product models: computer environments, supporting design and construction*. CRC press. <https://doi.org/10.1201/9781315138671>
- Penttilä, H., Rajala, M., & Freese, S. (2007). Building information modelling of modern historic buildings. *Predicting the Future, 25th eCAADe Konferansı, Frankfurt am Main, Germany*, 607-613. https://papers.cumin-cad.org/data/works/att/ecaade2007_124.content.pdf

- Cross, N. (2021). *Engineering design methods: strategies for product design*. John Wiley & Sons. https://www.academia.edu/download/37650372/engineering_design_methods.pdf
- Machairas, V., Tsangrassoulis, A., & Axarli, K. (2013). Algorithms for optimization of building design. *Renewable & Sustainable Energy Reviews*, (IKEEART-2014-804). <https://doi.org/10.1016/j.rser.2013.11.036>
- Jarratt, T. A. W., Eckert, C. M., Caldwell, N. H., & Clarkson, P. J. (2011). Engineering change: an overview and perspective on the literature. *Research in engineering design*, 22, 103-124. <https://doi.org/10.1007/s00163-010-0097-y>
- De Wit, S., & Augenbroe, G. (2002). Analysis of uncertainty in building design evaluations and its implications. *Energy and buildings*, 34(9), 951-958. [https://doi.org/10.1016/S0378-7788\(02\)00070-1](https://doi.org/10.1016/S0378-7788(02)00070-1)
- Jarratt, T., Clarkson, J., & Eckert, C. (2005). Engineering change. In *Design process improvement: a review of current practice* (pp. 262-285). London: Springer London. https://doi.org/10.1007/978-1-84628-061-0_11
- Koh, E. C., Caldwell, N. H., & Clarkson, P. J. (2012). A method to assess the effects of engineering change propagation. *Research in Engineering Design*, 23, 329-351. <https://doi.org/10.1007/s00163-012-0131-3>
- Ahmad, N., Wynn, D. C., & Clarkson, P. J. (2013). Change impact on a product and its redesign process: a tool for knowledge capture and reuse. *Research in Engineering Design*, 24, 219-244. <https://doi.org/10.1007/s00163-012-0139-8>
- Eckert, C., Clarkson, P. J., & Zanker, W. (2004). Change and customisation in complex engineering domains. *Research in engineering design*, 15, 1-21. <https://doi.org/10.1007/s00163-003-0031-7>
- Clarkson, P. J., Simons, C., & Eckert, C. (2004). Predicting change propagation in complex design. *J. Mech. Des.*, 126(5), 788-797. <https://doi.org/10.1115/1.1765117>
- Giffin, M., De Weck, O., Bounova, G., Keller, R., Eckert, C., & Clarkson, P. J. (2009). Change propagation analysis in complex technical systems. <https://doi.org/10.1115/1.3149847>

- Keller, R., Eckert, C. M., & Clarkson, P. J. (2005, July). Multiple views to support engineering change management for complex products. In *Coordinated and Multiple Views in Exploratory Visualization (CMV'05)* (pp. 33-41). IEEE. 10.1109/CMV.2005.11
- Eastman, C., Parker, D. S., & Jeng, T. S. (1997). Managing the integrity of design data generated by multiple applications: The principle of patching. *Research in engineering design*, 9, 125-145. <https://doi.org/10.1007/BF01596599>
- Reddi, K. R., & Moon, Y. B. (2009). A framework for managing engineering change propagation. *International Journal of Innovation and Learning*, 6(5), 461-476. <https://doi.org/10.1504/IJIL.2009.02506>
- Katz, R. H. (1990). Toward a unified framework for version modeling in engineering databases. *ACM Computing Surveys (CSUR)*, 22(4), 375-409. <https://dl.acm.org/doi/pdf/10.1145/98163.98172>
- Brahma, A., & Wynn, D. C. (2023). Concepts of change propagation analysis in engineering design. *Research in Engineering Design*, 34(1), 117-151. <https://doi.org/10.1007/s00163-022-00395-y>
- Saoud, L. A., Omran, J., Hassan, B., Vilutienė, T., & Kiaulakis, A. (2017). A method to predict change propagation within building information model. *Journal of Civil Engineering and Management*, 23(6), 836-846. <https://doi.org/10.3846/13923730.2017.1323006>
- Beitz, G. P. W., Wallace, K., Blessing, L., & Bauert, F. (1996). Engineering Design: A Systematic Approach. *MRS BULLETIN*, 71. <https://doi.org/10.1557/S0883769400035776>
- Ma, S., Song, B., Lu, W. F., & Zhu, C. F. (2003, January). A knowledge-supported system for engineering change impact analysis. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (Vol. 37009, pp. 439-447). <https://doi.org/10.1115/DETC2003/DAC-48749>
- Rutka, A., Guenov, M. D., Lemmens, Y., Schmidt-Schäffer, T., Coleman, P., & Riviere, A. (2006). Methods for engineering change propagation analysis. <http://dspace.lib.cranfield.ac.uk/handle/1826/2621>

- Jacobsen, K., Eastman, C., & Jeng, T. S. (1997). Information management in creative engineering design and capabilities of database transactions. *Automation in Construction*, 7(1), 55-69. [https://doi.org/10.1016/S0926-5805\(97\)00052-6](https://doi.org/10.1016/S0926-5805(97)00052-6)
- Pilehchian, B., Staub-French, S., & Nepal, M. P. (2015). A conceptual approach to track design changes within a multi-disciplinary building information modeling environment. *Canadian Journal of Civil Engineering*, 42(2), 139-152. <https://doi.org/10.1139/cjce-2014-0078>
- Wynn, D. C., Caldwell, N. H., & John Clarkson, P. (2014). Predicting change propagation in complex design workflows. *Journal of Mechanical Design*, 136(8), 081009. <https://doi.org/10.1115/1.4027495>
- Li, Y., Zhao, W., & Shao, X. (2012). A process simulation based method for scheduling product design change propagation. *Advanced Engineering Informatics*, 26(3), 529-538. <https://doi.org/10.1016/j.aei.2012.04.006>
- Moayeri, V. (2017). Design change management in construction projects using Building Information Modeling (BIM). *Canada: Concordia University*. https://spectrum.library.concordia.ca/id/eprint/983221/1/Moayeri_PhD_S2018.pdf
- Jeng, T. S., & Eastman, C. M. (1998). A database architecture for design collaboration. *Automation in Construction*, 7(6), 475-483. [https://doi.org/10.1016/S0926-5805\(98\)00056-9](https://doi.org/10.1016/S0926-5805(98)00056-9)
- Whyte, J., Soman, R., Sacks, R., Mohammadi, N., Naderpajouh, N., Hong, W. T., & Lee, G. (2024). How digital twins provide new opportunities for managing change in complex projects. *arXiv preprint arXiv:2402.00325*. <https://doi.org/10.48550/arXiv.2402.00325>
- Huang, G. Q., & Mak, K. L. (1999). Current practices of engineering change management in UK manufacturing industries. *International Journal of Operations & Production Management*, 19(1), 21-37. <https://doi.org/10.1108/01443579910244205>
- Yin, L., Tang, D., Kang, Y., & Leng, S. (2016). Topology face-based change propagation analysis in aircraft-assembly tooling design. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 230(1), 120-135. <https://doi.org/10.1177/095440541455869>

- Masmoudi, M., Leclaire, P., Zolghadri, M., & Haddar, M. (2017). Change propagation prediction: A formal model for two-dimensional geometrical models of products. *Concurrent Engineering*, 25(2), 174-189. <https://doi.org/10.1177/1063293X17698192>
- Chen, J., Zhang, S., Wang, M., & Xu, C. (2017). A novel change feature-based approach to predict the impact of current proposed engineering change. *Advanced Engineering Informatics*, 33, 132-143. <https://doi.org/10.1016/j.aei.2017.06.002>
- Regli, W. C., & Cicirello, V. A. (2000). Managing digital libraries for computer-aided design. *Computer-Aided Design*, 32(2), 119-132. [https://doi.org/10.1016/S0010-4485\(99\)00095-0](https://doi.org/10.1016/S0010-4485(99)00095-0)
- Szykman, S., Sriram, R. D., Bochenek, C., Racz, J. W., & Senfaute, J. (2000). Design repositories: next-generation engineering design databases. *IEEE Intelligent Systems*, 15(3), 48-55. https://www.researchgate.net/profile/Rdhanapal-R/publication/2807188_Design_Repositories_Next-Generation_Engineering_Design_Databases/links/09e415099c6e321096000000/Design-Repositories-Next-Generation-Engineering-Design-Databases.pdf
- Zahedi, A., Abualdenien, J., Petzold, F., & Borrmann, A. (2022). BIM-based design decisions documentation using design episodes, explanation tags, and constraints. *J. Inf. Technol. Constr.*, 27, 756-780. <https://www.semanticscholar.org/reader/48a5d9bede2a0ef910a4b8daaf6592f81eb79bb4>
- Zahedi, A., & Petzold, F. (2022). REVIT ADD-IN FOR DOCUMENTING DESIGN DECISIONS AND RATIONALE. https://papers.cumincad.org/data/works/att/caadria2022_76.pdf
- Martin, W. M., Heylighen, A., & Cavallin, H. (2003, April). Building stories. A hermeneutic approach to studying design practice. In *Proceedings of the 5th european academy of design conference, Barcelona, Spain* (pp. 28-30). <https://www.academia.edu/download/31905565/MartinCH.pdf>
- Xue, D., Yang, H., & Tu, Y. L. (2005, January). Modeling of evolutionary design database. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (Vol. 4739, pp. 109-122). <https://doi.org/10.1115/DETC2005-84956>

- Voropajev, V. (1998). Change management—A key integrative function of PM in transition economies. *International Journal of Project Management*, 16(1), 15-19. [https://doi.org/10.1016/S0263-7863\(97\)00010-0](https://doi.org/10.1016/S0263-7863(97)00010-0)
- Antill, J. M., & Woodhead, R. W. (1991). *Critical path methods in construction practice*. John Wiley & Sons.
- Cox, I. D., Morris, J. P., Rogerson, J. H., & Jared, G. E. (1999). A quantitative study of post contract award design changes in construction. *Construction Management & Economics*, 17(4), 427-439. <https://doi.org/10.1080/014461999371358>
- Rahmani Mirshekarlou, B. (2012). *A taxonomy for causes of changes in construction* (Master's thesis, Middle East Technical University). <http://etd.lib.metu.edu.tr/upload/12614692/index.pdf>
- MacLeamy, P. (2004). Collaboration, integrated information and the project lifecycle in building design, construction and operation. *WP-1202, The construction users roundtable*. <https://kcuc.org/wp-content/uploads/2013/11/Collaboration-Integrated-Information-and-the-Project-Lifecycle.pdf>
- Murdock, J. W., Szykman, S., & Sriram, R. D. (1997, September). An information modeling framework to support design databases and repositories. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (Vol. 80463, p. V004T31A042). American Society of Mechanical Engineers. <https://doi.org/10.1115/DETC97/DFM-4373>
- Tammik, J., & Contributors. (2023). *Revit Lookup [Software]*. Version 2023.0.1. GitHub. <https://github.com/jeremytammik/RevitLookup>
- Ullman, J.D. (1988). *Principles Of Database And Knowledge-Base Systems*.
- Kim, I., Lee, M., Choi, J., & Kim, G. (2016). Development of an application to generate 2D drawings in automation using open BIM technologies. *Korean Journal of Computational Design and Engineering*, 21(4), 417-425. <https://doi.org/10.7315/cde.2016.417>
- Beach, T., Petri, I., Rezgui, Y., & Rana, O. (2017). Management of collaborative BIM data by federating distributed BIM models. *Journal of Computing in Civil Engineering*, 31(4), 04017009. [https://doi.org/10.1061/\(ASCE\)CP.1943-5487.0000657](https://doi.org/10.1061/(ASCE)CP.1943-5487.0000657)

-
- Oh, M., Lee, J., Hong, S. W., & Jeong, Y. (2015). Integrated system for BIM-based collaborative design. *Automation in construction*, 58, 196-206.
<https://doi.org/10.1016/j.autcon.2015.07.015>

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Thesis selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ich versichere außerdem, dass die vorliegende Arbeit noch nicht einem anderen Prüfungsverfahren zugrunde gelegen hat.

München, 11. Oktober 2024

Lingyun Yan

Lingyun Yan

██████████

██████████████████

████████████████████