Isabela de Carvalho Figueiró

Automation of Sub-Processes in the Design Stage

BIM A+

UMinho | 2023

Universidade do Minho
Escola de Engenharia

Isabela de Carvalho Figueiró

Automation of Sub-Processes in the Design Stage

BIM A+ European Master in Building Information Modelling

Co-funded by the Erasmus+ Programme of the European Union

October 2023

Universidade do Minho
Escola de Engenharia

Isabela de Carvalho Figueiró

Automation of Sub-Processes in the Design Stage

European Master in
Building Information Modelling

Master Dissertation
European Master in Building Information Modelling

Work conducted under supervision of:
José Granja
Rui Dias (Tutor in company)

Co-funded by the
Erasmus+ Programme
of the European Union

October, 2023

ACKNOWLEDGMENTS

I would like to begin by extending my gratitude to all professors who have imparted their invaluable knowledge throughout this master's program. Their lectures and guidance have played an instrumental role in nurturing our skills and preparing us to excel as BIM practitioners.

I am thankful to my supervisor, José Granja, for his support and expert guidance throughout this journey.

I would also like to express my appreciation to NOZ Arquitectura, with a special mention to Rui Dias, for their generous support and assistance in the development of my dissertation.

My warmest thanks go out to my fellow classmates who have been by my side throughout this challenging and rewarding master's course. To Andrijana, Andréa, Nguyen Son, and Tijana, I extend my gratitude for the countless moments we have shared over the past year.

I am profoundly grateful to my friends from Brazil for being there to listen and support me during this significant period of my life.

In conclusion, my deepest and most heartfelt gratitude goes to my family, particularly my parents Nelso and Rosane, who have always believed in me and supported me in the pursuit of my dreams. I love you all.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

RESUMO

Essa dissertação aborda o papel da automação em BIM na etapa de definição de projeto. O grande uso do BIM e o desejo de responder de forma mais eficaz aos requisitos de construção difundiram o desenvolvimento de software, plug-ins, add-ons e scripts de linguagens de programação. Este estudo avalia os desenvolvimentos atuais nesse campo, suas aplicações e sua usabilidade com base em subprocessos que foram identificados com potencial para serem automatizados.

A tese está organizada em 5 capítulos. O primeiro fornece uma introdução ao tópico e o segundo uma revisão da literatura sobre as etapas de projeto, os elementos constituintes dessas etapas, a automação atualmente disponível nessas etapas e como é possível desenvolver novas automatizações. O terceiro capítulo é dedicado à pesquisa dos subprocessos que serão automatizados neste trabalho e como eles foram resolvidos. O quarto capítulo ilustra um estudo de caso com uma lista de tarefas a serem realizadas, manualmente e com automação, com base nos subprocessos e apresentados os resultados desse estudo de caso. O capítulo final é a conclusão deste trabalho e as orientações para desenvolvimentos futuros.

A integração de automação em ferramentas BIM estabelece novas possibilidades, capacitando os arquitetos a ampliar a área de trabalho, simplificar as tarefas manuais e elevar a qualidade de seus resultados.

Palavras chave: Automação, BIM (Building Information Modelling), Dynamo, Design, Processos.

ABSTRACT

The dissertation addresses the role of automation in the Design Stage of BIM. The large use of BIM and the desire to respond more effectively to the construction requirements have diffused the development of software, plug-ins, add-ons, and Programming Languages scripts. This study evaluates the current developments in this field, its applications, and its usability based on sub-processes that were identified as potential for being automated.

The thesis is organized into 5 chapters. The first provides an introduction to the topic, and the second a literature review about the design stage, the deliverables in this stage, the currently available automation in this stage, and how is possible to develop new ones. The third chapter is the survey of the sub-processes that are going to be automated in this work and how they were solved. The fourth chapter illustrates a case study with a list of tasks to do, manually and with automation, based on the sub-processes and the results of this case study. The final chapter is the conclusion of this work and the directions to future developments.

The integration of automation into BIM tools ushers in a new era of possibilities, empowering architects to augment their work, streamline manual tasks, and elevate the quality of their deliverables.

Keywords: Automation, BIM (Building Informaiton Modelling), Design Stage, Dynamo, Processes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Building Information Modelling (BIM) is one of the foundation of digital transformation in the Architecture, Engineering, and Construction (AEC) industry. According to Succar (2009), BIM is "a set of interacting policies, processes, and technologies generating a methodology to manage the essential building design and project data in digital format throughout a building's life cycle." While the utilization of solely BIM data to check legal codes and fulfill regulatory prerequisites is currently under examination within the contexts of Portugal, Brazil, Finland, United Kingdom, Denmark, Germany and Spain, (Building Smart, 2023) also the projects such as *Check, Accord and Digichecks* that are studying this subject in Europe, it is noteworthy that conventional 2D drawings and documentation persist as the established norm in the domains of construction contracts, legal frameworks, and the operational realm of construction activities.

While BIM has garnered substantial adoption within the design phase, its integration within the construction phase remains notably limited, with prevalent reliance on conventional paper-based drawings among on-site personnel. Contemporary construction operations primarily hinge on the utilization of traditional two-dimensional (2D) drawings (Bråthen et al., 2015). Several factors contribute to this prevailing practice, including historical limitations in the availability of suitable solutions for implementing BIM within work environments such as construction sites. Challenges stemming from transformative cultural shifts and the requisite time investment for training construction workers and artisans further compound this scenario. Consequently, the attainment of benefits surpassing associated costs through BIM deployment remains a complex subject, and not universally applicable to all construction projects in the present context (Disney et al., 2022).

Within this context, the extraction of two-dimensional (2D) perspectives and schematics from a three-dimensional (3D) model constitutes a pivotal aspect of the practical deployment of BIM in real-world scenarios (An et al., 2020). The primary goal of Building Information Modelling (BIM) is directed towards enhancing the efficacy and value of information, in contrast to the principal intent underlying 2D drawings, which is the communication of essential construction-related information. Nonetheless, the transformation of BIM-generated data into 2D deliverables necessitates a substantial investment of both time and laborious effort (Kim et al., 2022). According to Kim, in practical implementation, the process of producing two-dimensional (2D) outputs from Building Information Modelling (BIM) leads to an augmented workload, exceeding a notable threshold of 41% until process is implemented.

To reduce the amount of time wasted, as well as the inaccuracies caused by handwork, and to enhance the quality related to the two-dimensional (2D) drawings, automation is the main potential solution. Automation in the AEC industry is not a new concept, it has been employed from the early days of computer-aided modeling and simulation (Dixon, 1995). The goal of automation in an information system is to shift work from humans toward computers and machines (Kroenke, 2011). Automation may be used to accomplish design processes more consistently and quickly than conventional methods. It has also been stated that automation not only increases productivity but also serves as an enabler in identifying ideas that would otherwise go unexplored throughout the design phase and different solutions for the project (Mukkavaara, 2018). It eliminates the time for the designer to ensure that sufficient information has been incorporated into the construction drawings or fabrication drawings (Deng, 2018). Automation has been applied in the design stage in a lot of processes, such as tools to streamline feasibility studies, the use of Visual Programming Language to develop initial optimization facade solutions, automation tools to automate model checking for building permits, tools for Quantity Take-off (QTO) extraction and documentation generation. Despite these incredible accomplishments, it's important to recognize that there is still room for new automation technologies to improve and innovate within the construction sector.

1.1. Objectives

Considering all the above mentioned, this study aims to evaluate the advantages, potentialities, and challenges inherent in the implementation of automation within the Building Information Modeling (BIM) framework, with particular emphasis on its applicability to architectural context during the Conceptual Design phase. The main goal of this research is to create, and test automation related to project modeling within BIM software, as well as the generation of 2D drawings from the BIM model.

This dissertation is driven by the intention to demonstrate that the integration of automation in certain processes yields time-saving benefits for collaborative teams and engenders 2D shop drawings characterized by heightened precision, clarity, and legibility. The objectives are defined as follows: 1) investigation into existing automated solutions available within the market scenario; 2) creation of Dynamo routines to solve sub-processes in the Design Phase; 3) developing a study case in two ways: one in the usual way (manually) and the other using the researched and developed automation; 4) comparison analysis of outcomes, time used in both ways, with the objective to ascertaining whether the adoption of automation contributes to a discernible reduction in working hours

In conclusion, this research endeavor seeks to advance an enhanced understanding of the role of automation in the architectural context within the purview of BIM implementation. Through a methodical exploration of automated functionalities, market offerings, and customized scripting, this study aims to underscore the potential benefits of automation, particularly in terms of temporal efficiency and the quality of resulting project documentation.

1.2. Dissertation Structure

Chapter 2 provides an in-depth literature review on the current state of architectural project development using Building Information Modeling (BIM) methodology. This review encompasses prevalent workflows, expected deliverables, identified improvements, particularly through automation, and associated challenges. The study investigates the benefits and challenges of integrating automation within architectural projects, focusing on design phase automation initiatives. It concludes by surveying ongoing efforts to implement automation in architectural design processes and the way to develop more automation.

Chapter 3 delineates the approach of the dissertation. This involves a collaborative involvement with NOZ Arquitectura, aimed at discerning sub-processes within their work that are possible to be automated and following the methodology of how these sub-processes are going to be solved.

Chapter 4 delves into the process of sub-processes resolution. It unfolds a dual approach: sub-processes addressed via existing add-ons or plug-ins are expounded, encompassing their application, benefits, and challenges. Simultaneously, sub-processes necessitating an alternative route are approached through the development of customized Dynamo scripts, wherein the logic and developmental challenges are shown.

Chapter 5 is about the application of sub-processes within a designated study case in two distinct ways. The first way involves manual execution, utilizing only Autodesk Revit's intrinsic capabilities, while the second way leverages the automation developed in Chapter 4. This dichotomous approach was made for a comprehensive evaluation of time efficiencies and drawing quality improvements, culminating in an assessment of automation's impact in the design phase of a project.

Chapter 6, the conclusion, presents the achieved outcomes of the developed work. Additionally, the conclusion points towards future directions by highlighting potential areas for further automation advancements in the design phase. This chapter serves as a dual-purpose juncture, summarizing the study's significance while also indicating future developments.

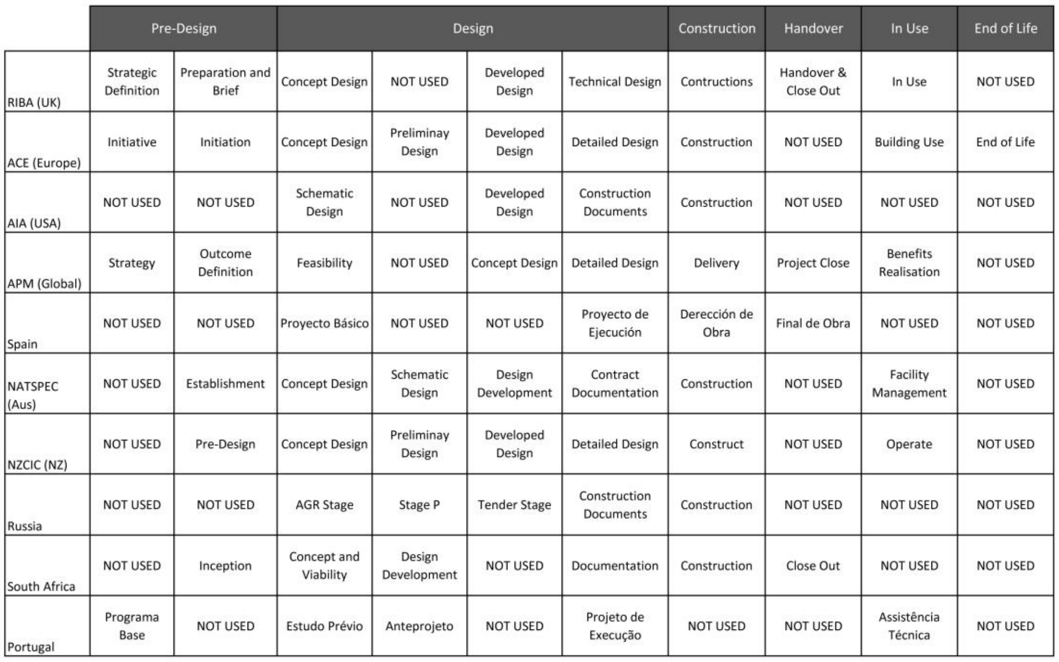
This page is intentionally left blank

2. LITERATURE REVIEW

Building Information Modeling (BIM) has emerged as a transformative method in the architecture, engineering, and construction (AEC) industry, revolutionizing the way projects are conceived, designed, and executed. Central to the success of BIM is the integration of automation throughout the project lifecycle, particularly in the Design Stage. This literature review investigates into automation in BIM during the Design Stage, exploring recent developments, and the profound impact of these technologies on the AEC industry. As construction projects grow increasingly complex and demand efficiency and accuracy, understanding the role of automation in BIM design processes becomes essential. This study seeks to offer a thorough overview of the current state of the art, exposing the difficulties and opportunities brought on by automation in BIM design, and exposing the future development of this subject.

2.1. Design Stage

A project, regardless of its purpose and size, has a systematic division of its plan of work into distinct phases. These phases are contingent upon variables encompassing country, corporate entities, project typology, size of the project, and standards. In alignment with the delineations established by the Royal Institute of British Architects (RIBA,2020), which has collected and compared plans of work from diverse countries, it is evident that a prevailing consensus emerges. This consensus asserts the division of a project's lifecycle into six stages: Pre-design, Design, Construction, Handover, In Use, and End of Life. The design stage as shown in Figure 1, is usually divided into more stages. Take into account the RIBA plan of work to define the Design Stage, it is divided into 3 stages: Concept Design, Developed Design, and Technical Design.



	Pre-Design		Design				Construction	Handover	In Use	End of Life
RIBA (UK)	Strategic Definition	Preparation and Brief	Concept Design	NOT USED	Developed Design	Technical Design	Contructions	Handover & Close Out	In Use	NOT USED
ACE (Europe)	Initiative	Initiation	Concept Design	Preliminay Design	Developed Design	Detailed Design	Construction	NOT USED	Building Use	End of Life
AIA (USA)	NOT USED	NOT USED	Schematic Design	NOT USED	Developed Design	Construction Documents	Construction	NOT USED	NOT USED	NOT USED
APM (Global)	Strategy	Outcome Definition	Feasibility	NOT USED	Concept Design	Detailed Design	Delivery	Project Close	Benefits Realisation	NOT USED
Spain	NOT USED	NOT USED	Proyecto Básico	NOT USED	NOT USED	Proyecto de Ejecución	Derección de Obra	Final de Obra	NOT USED	NOT USED
NATSPEC (Aus)	NOT USED	Establishment	Concept Design	Schematic Design	Design Development	Contract Documentation	Construction	NOT USED	Facility Management	NOT USED
NZCIC (NZ)	NOT USED	Pre-Design	Concept Design	Preliminay Design	Developed Design	Detailed Design	Construct	NOT USED	Operate	NOT USED
Russia	NOT USED	NOT USED	AGR Stage	Stage P	Tender Stage	Construction Documents	Construction	NOT USED	NOT USED	NOT USED
South Africa	NOT USED	Inception	Concept and Viability	Design Development	NOT USED	Documentation	Construction	Close Out	NOT USED	NOT USED
Portugal	Programa Base	NOT USED	Estudo Prévio	Anteprojeto	NOT USED	Projeto de Execução	NOT USED	NOT USED	Assistência Técnica	NOT USED

Figure 1 – Plan of work by the author adapted from RIBA

The fundamental objective inherent to a plan of work is to provide structured guidance to clients spanning the continuum of project evolution, from the beginning of the project, through design and construction, culminating in project handover and subsequent stages.

In the Concept Design stage, the preparation of proposals considers the Site Information, the Project Brief, and the Spatial Requirements. The client and other project stakeholders' opinions are regularly sought through design reviews, and the design is then revised in response. Any derogations from the project brief are agreed upon, or the project brief is changed to better suit the architectural concept. To include feedback from the design team, specialized consultants, and strategic engineering needs (building services, civil and structural engineering), versions of the architectural concept proposals must also be made. The ideas and outline specification should be in conformity with the project budget, as shown by the cost plan. The proposals should show that the spatial criteria and any adjacency requirements have been met. The exterior façade must both fulfill the client's vision and the requirements of the surrounding context and environment (RIBA, 2020).

The main goal of Spatial Coordination is to test and validate that the Architectural Concept, to ensure that the engineering and architectural information produced in the Concept Design is Spatially Coordinated. Detailed Design Studies and Engineering Analyses are made to ensure the assumptions made during Concept Design and to layer more detail onto the design. Spatial Coordination is not about adjusting the Architectural Concept, which should remain significantly unaltered, although detailed design or engineering processes may require some adjustments to ensure that the project is Spatially Coordinated. Versions of the design may be necessary to verify that the Cost Plan is in accordance with the Project Budget. Design studies should be aligned with Cost Exercises and the preparation of the Outline Specification. Product manufacturers and specialized subcontractors may be consulted to test or validate particular design elements. Without further significant design iterations, a spatially coordinated design enables all designers, including specialized subcontractors, to conclude their knowledge at the Technical Design stage (RIBA, 2020).

The Technical Design is a critical stage in the project where the design is refined into a detailed and coordinated set of technical drawings, specifications, and other construction documentation. It involves choosing the specific details of how the building will be built, such as materials, systems, and processes. Architects and engineers work closely together throughout the Technical Design phase to ensure that the design is not only aesthetically beautiful but also possible and practical to build. The goal is to create a comprehensive set of information that contractors may use for pricing and construction.

Following are some of the main activities that may occur during the Technical Design stage: the development of detailed architectural drawings and plans, the coordination of structural and engineering systems, the specification of materials, finishes, and components, the incorporation of relevant regulations and standards, the consideration of sustainability and energy efficiency strategies, the refinement of the project budget and schedule, and the preparation of any required permit applications.

Considering all the above, the design stage can be considered the most important stage of a project in relation to its further contribution to the project life cycle. In this stage critical aspects, such as resources estimations and costs, modeling, and planning are set (Maylor, 2003). And it is in this stage that the details and information that are going to be used in a construction project are defined. In recent years, BIM technology has been focused on the design stage because it is considered the stage of the project where most of the elements are defined and most of the information is provided.

2.1.1. Workflow of an architectural project in the design stage

According to BIMe (BIMe, 2023) "a workflow identifies major successive activities to perform, decision gates to pass-through, and delivery milestone to reach. A BIM workflow is typically part of larger BIM Processes - aimed at fulfilling strategic/operational objectives - and may include several documented Procedures".

The creation of a workflow is a crucial part in order to get the greatest outcomes out of a process and to handle information properly. An architectural project workflow is a structured and systematic process that outlines the various stages and steps involved in conceiving, designing, and executing an architectural project. It provides a roadmap for architects and design teams to follow, ensuring that the project progresses efficiently from inception to completion while meeting the client's objectives and adhering to industry standards and regulations.

In order to understand the processes of the Design Stage, and to identify the outcomes, an architectural project workflow was establish based on RIBA stages and based on experience within architectural field (Figure 4). This workflow aims to provide valuable insights into the potential processes where automation can be applied for improvements and to enhance efficiency, reduce errors, and improve project outcomes. By carefully analyzing each stage of the workflow, is possible to identify opportunities for automation that align with their specific project needs and objectives.

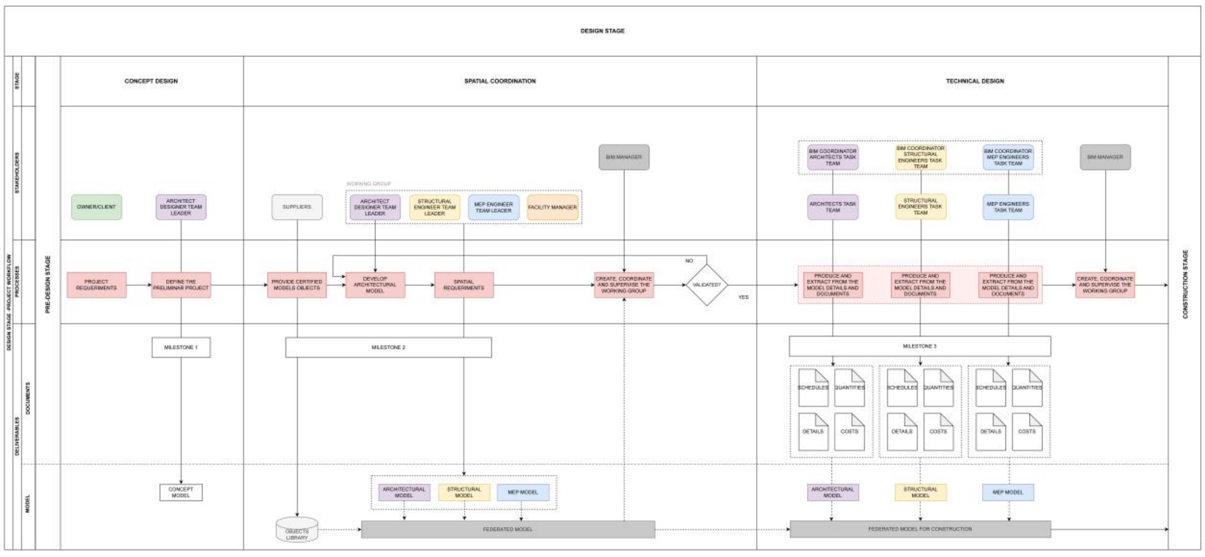

DESIGN STAGE
STAGE
STAKEHOLDERS
DESIGN STAGE -PROJECT WORKFLOW
PROCESSES
DELIVERABLES
DOCUMENTS
MODEL
PRE-DESIGN STAGE
CONCEPT DESIGN
SPATIAL COORDINATION
TECHNICAL DESIGN
CONSTRUCTION STAGE
OWNER/CLIENT
ARCHITECT DESIGNER TEAM LEADER
SUPPLIERS
WORKING GROUP
ARCHITECT DESIGNER TEAM LEADER
STRUCTURAL ENGINEER TEAM LEADER
MEP ENGINEER TEAM LEADER
FACILITY MANAGER
BIM MANAGER
BIM COORDINATOR ARCHITECTS TASK TEAM
BIM COORDINATOR STRUCTURAL ENGINEERS TASK TEAM
BIM COORDINATOR MEP ENGINEERS TASK TEAM
BIM MANAGER
ARCHITECTS TASK TEAM
STRUCTURAL ENGINEERS TASK TEAM
MEP ENGINEERS TASK TEAM
PROJECT REQUIREMENTS
DEFINE THE PRELIMINAR PROJECT
PROVIDE CERTIFIED MODELS OBJECTS
DEVELOP ARCHITECTURAL MODEL
SPATIAL REQUIREMENTS
CREATE, COORDINATE AND SUPERVISE THE WORKING GROUP
NO
VALIDATED?
YES
PRODUCE AND EXTRACT FROM THE MODEL DETAILS AND DOCUMENTS
PRODUCE AND EXTRACT FROM THE MODEL DETAILS AND DOCUMENTS
PRODUCE AND EXTRACT FROM THE MODEL DETAILS AND DOCUMENTS
CREATE, COORDINATE AND SUPERVISE THE WORKING GROUP
MILESTONE 1
MILESTONE 2
MILESTONE 3
SCHEDULES
QUANTITIES
DETAILS
COSTS
CONCEPT MODEL
OBJECTS LIBRARY
ARCHITECTURAL MODEL
STRUCTURAL MODEL
MEP MODEL
FEDERATED MODEL
FEDERATED MODEL FOR CONSTRUCTION

Figure 2 – Workflow of an architectural project – small version - large version on Appendix 1.

2.1.2. EIR-Deliverables

In the Design Stage is important that all the deliverables contain all the information defined in the EIR. The EIR document (Exchange Information Requirements, sometimes also referred to as Employer Information Requirements) is defined in ISO 19650-1 and ISO 19650-2, and it is composed by the appointing party (based on the OIR, AIR, and PIR) and the lead appointed party (depending on the project plan of work) requirements. The appointing party's EIR specifies which information and data must be delivered at each exchange through the lifecycle of a project. Also, must contain the specification of the Level of Information Need, and the dates for reviewing and accept information, and all of that must ensure consistency, accuracy, and efficiency in information exchange, reducing misunderstandings and errors during the project's lifecycle. The Exchange Information Requirements (EIR) is not universally standardized; instead, it is customized for each individual project. This customization takes into consideration a range of project-specific circumstances, encompassing aspects such as size, complexity, geographical location, type, and specific requirements.

As shown in the EIR (Figure 3) of a project, since the Design Stage is divided into 3 stages there are many moments that are necessary to deliver information. The way of delivering information can be in several ways, it depends on who is going to receive the information and it is specified in the EIR. In Concept Desing most of the deliverables are 3D images and 2D drawings from the 3D model made in a BIM platform since is a handover for the client. In Spatial Coordination, the delivery usually is the BIM model, generally in IFC format. In this stage, also happens the approval of the project by legal authorities, and for that, is necessary to deliver the project information according to the rules of the municipality. In the current context, there are few countries where the use of BIM is mandatory, according to Figure 4, and there are some countries, depending on the municipality, where the possibility of using 3D modeling files, such as IFC, is possible (Choi et al., 2018). To the approval of the project, in most countries, the use of 2D drawing remains consistent, in digital or physical files, to this process (Building Smart, 2023).

PLAN OF WORK - DESIGN STAGE			
DESIGN STAGE	Stage	Milestone	Deliverables
	Concept Design	Milestone 1	. Architectural Concept Design Model . Drawings
	Spatial Coordination	Milestone 2	. Architectural Spatial Coordination Model . Structural Model . MEP Model . Objects for Model . Federated Model . Drawings
	Technical Design	Milestone 3	. Architectural Technical Design Model . Drawings . Schedules . Details . Costs

Figure 3 - EIR Plan of Work in the Design Stage

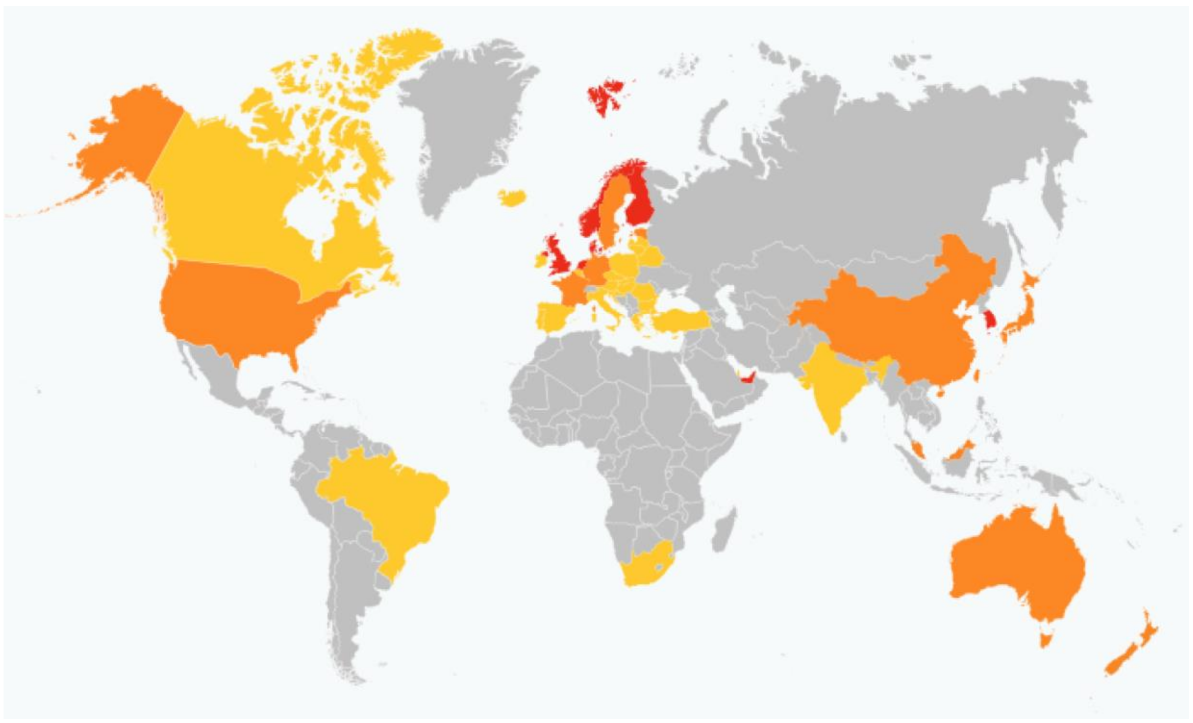


Figure 4 – Map of International Implementation of BIM (BuildingSmart, 2023)

In the Technical Design Stage, the main outcome is the technical drawings and documentation for construction. Nowadays, with the use of BIM, the construction site has become more digital, and the use of digital equipment increased, but most of the is used by the managers of the construction and less by the carpenters and bricklayers (Murvold et al., 2016). The main obstacles to using digital equipment, such as BIM stations, are the quality of the model, the on-site education, the technical support, the user-friendliness, and the cost (Sundqvist. et al., 2020). Because of those reasons, the site work in construction projects is still dominated by paper in the form of 2D drawings and other design information (Davies and Harty, 2013)

2.1.3. Opportunities for Process Improvement

The Building Information Modeling (BIM) technology has yielded substantial enhancements to architectural and engineering processes, especially within the Design Stage of construction projects. However, as elucidated in the previous chapter, conventional 2D drawings remain indispensable across many applications. While BIM has revolutionized the generation of drawings compared to conventional Computer-Aided Design (CAD) tools, it is noteworthy that the production of 2D drawings remains a requisite.

The BIM technology brought for construction a lot of improvements in the Design Stage, but as explained in the previous chapter, the 2D drawings are still necessary for many uses. With BIM, the creation of them already improved from what is the usual, doing with CAD tools. To generate the drawings using BIM methodology is more automatic and every change in the model, changes all the drawings related to it, making them accurate and always updated.

Even though BIM makes the creation of drawings more quickly and easier placing the information on the drawing still is necessary to do a lot of processes manually and repeatedly. In practice, generating 2D deliverables from BIM results in additional work (>41%) and reduced consistency between drawings and BIM model data (Kim et al., 2022).

To decrease the additional work that is necessary for the creation of 2D drawings, one of the ways is to use automation to perform the repetitive and time-wasting processes. The inclusion of automation in the project workflow, allows design professionals to dedicate their knowledge to more intricate aspects of the project. Automation not only accelerates the generation of 2D drawings but also produces consistency and accuracy throughout the documentation process. Additionally, using automation reduces the possibility of mistakes resulting from manual interventions by producing standardized and consistent drawings.

2.2. Automation

Automation according to Groover (2023), is the “application of machines to tasks once performed by human beings or, increasingly, to tasks that would otherwise be impossible.”

Although the term "mechanization" is frequently used to describe the simple substitution of machines for human labor, automation typically signifies the incorporation of machines into a self-governing system. There is hardly a part of modern life that has not been impacted by automation, which has revolutionized the fields in which technology has been used (Groover, 2023).

Around 1946, the word "automation" was used in the car industry to characterize the growing use of automated machinery and controls in mechanized production lines. The word is frequently used in the context of manufacturing, but it is also used outside of manufacturing in relation to a number of systems where human effort and intellect are significantly replaced by mechanical, electrical, or computerized action (Groover, 2023).

Several notable innovations happened in numerous disciplines throughout the twentieth century, including the digital computer, advancements in data-storage technology and software to build computer programs advances in sensor technology, and the creation of a mathematical control theory. All these advancements have led to the advancement of automation technology (Groover, 2023).

2.2.1. Automation in the AEC industry

The AEC industry also was impacted by automation. First was impacted by the emergence of CAD, and after with the emergence of BIM. Since around 1970, there was an increasing tendency in manufacturing organizations toward using computers to undertake numerous design-related jobs. Computer-aided design (CAD) is the technology linked with this movement. CAD is built on a computer system's capacity to process, store, and display enormous volumes of data representing components and products. Computer-aided design (CAD) use computer technologies to help in the development, modification, analysis, and optimization of a design. Working with a CAD system rather of a traditional drawing board, the designer produces the lines and surfaces that make up the item (product, part, structure, etc.).

CAD impacted the AEC industry by revolutionizing the way professionals plan, design, and execute construction projects. It helped the industry to be automated and grow by changing from manual drawing to digital design.

But with the technological advances, the increase of complex projects, the need for interdisciplinary collaboration, and the necessity of a lifecycle project approach, the BIM concept started gaining traction, focusing on creating intelligent digital models that contain both geometric and non-geometric data. BIM is seen as a technical advance compared to traditional CAD, providing a higher level of capability regarding intelligence and interoperability (Miettinen & Paavola, 2014)

2.2.2. Automation in BIM

One type of automation is process automation, which can be defined as the use of technology and software to simplify, optimize, and automate different operations and workflows inside an organization. Process automation aims to increase productivity, decrease mistakes, promote cooperation, and free up human resources from repetitive and manual duties, allowing them to focus on more strategic and creative parts of their work (Mukkavaara, 2018). Process automation may be used in a variety of departments and sectors, ranging from finance and human resources to manufacturing and customer service.

Automation within BIM processes aims to streamline and enhance various processes associated with the design, construction, and maintenance of buildings and infrastructure. Automation finds application in several key areas within the Building Information Modeling (BIM). These areas include: 1) the automated generation of models derived from point cloud data, commonly referred to as Scan-to-BIM; 2) the identification and resolution of clashes within the model; 3) the automated of quantities take-off and estimation; 4) the automated assessment of compliance with pertinent building codes; 5) the automated exploration and analysis of diverse design options; 6) the automated facilitation of facility management and maintenance processes; 7) the automated generation of documentation; 8) and lastly, the automated execution of energy analyses for the assessment of a structure's energy performance; 9) automation in construction (Mukkavaara, 2018).

The main approaches to automating BIM-based workflows in the design process are the application of parametric modelling, design optimization, clash detection, extraction of quantity take-off, analysis and simulations and documentation (Mukkavaara, 2018). In the architectural projects automation can be applied in all the project stages that architecture is involved. In the Strategic Definition, automation can be applied in the feasibility studies helping to generate design alternatives based on parameters and constrains, helping the architects to explore different feasibility scenarios. In the Preparation and Brief and in the Concept Design, automation can assist in generating design and shape options, allowing architects to quicky visualize and show different design approaches to the client. Following, in the Spatial Coordination, it can be used in clash detection tools, identifying conflicts between the architectural components and building systems, improving coordination, and reducing errors. In the Technical Design automation can help to generate automatically detailed architectural drawings, annotations, and schedules from the BIM model, ensuring accuracy and consistency in documentation.

In conclusion, the integration of automation within Building Information Modeling (BIM) stands as a transformation, revolutionizing various phases across the construction lifecycle. By making the process with enhanced efficiency, accuracy, and versatility, automation not only streamlines routine processes but also increases the overall quality and coherence of project outcomes.

2.2.3. Current developed automations for BIM in the Design Stage

There has been a long historical interest in automation applied to Building Information Modeling (BIM) in the design stage, driven by the ever-increasing complexity of construction projects and the need for more efficient and effective ways to plan, design, and execute these projects.

In the Concept Design is set the Architectural Concept for a project. During this stage, it is often essential to develop a 3D visualization to facilitate the comprehension of the project's fundamental concept by employers, consultants, and other stakeholders (Rossi, 2020). In this field, many automations have been developed to help architects to deliver proposals of a concept design aligned with the Site Information and the Project Brief, including the Spatial Requirements (RIBA, 2020). The software Testfit generates massing of the building on the real site’s GIS data (Testfit, 2023). Based on the data from the client the software can generate multiple results of a building and it can be adjusted when modifications happen. This tool significantly streamlines the process of conducting feasibility studies by making them faster and more efficient. In this stage, the first solar analysis can be performed to get the best solutions for the façade and the use of Dynamo, a Visual Programming Language, could be an efficient tool to make it (Kensek et al. 2015).

The Spatial Coordination is about testing and validating the Architectural Concept, to make sure that the architectural and engineering information prepared in the Concept Design is Spatially Coordinated before the detailed information required to manufacture and construct the building is produced. Several automations have been developed to enhance efficiency and accuracy in this phase (RIBA, 2020). The research of M. Oliveira (2022) describes the tools developed to automate model checking for building permits, such as Corenet in Singapore, KBIM in South Korea, EUnet4DBP in Europe, and Metropolis in Brazil. A. Aghabayli (2021) developed a work about the use of machine learning to automatic prediction and labeling spaces in the BIM model. These developments reflect a growing trend in the industry towards leveraging automation, data analysis, and machine learning to enhance the Spatial Coordination phase. These tools contribute to a smoother transition from concept to construction and ultimately result in more efficient and accurate building processes.

The Technical Design involves the preparation of all design information required to manufacture and construct the project (RIBA, 2020). The use of automation in this stage can be related to extraction of Quantity Take-off (QTO) as developed by L. Vieira (2020) that developed a tool BIM-based that automatically generates a coordinated report for specification and quantity take-off, setting up also a preliminary cost estimation. Also another development in this stage is the automation of the documentation that was researched by E. Ermolenko (2020), and resulted in the development of a Dynamo Script for the views generation and automatic layout on sheets.

The integration of automation in various stages of the architectural and construction process, particularly in the Design Stage, represents a significant evolution in the industry. These advancements have been driven by the need to tackle the growing complexity of construction projects while striving for greater efficiency and accuracy. However, it's important to recognize that the journey towards automation in the construction industry is far from complete. There are still many opportunities for further advancements and developments in automation technologies.

2.2.4. Challenges in implementing automation within architectural projects in BIM

The integration of automation in architectural projects offers a lot of benefits, from increased efficiency to enhanced accuracy in design and construction processes. However, it's important to know the challenges of applying automation in architectural projects. These challenges came from the complexities of architectural creativity, where automation might struggle to encapsulate the nuanced decisions and artistic expressions that architects bring to their work (Disney et al., 2022).

Data standards and quality are a considerable additional hurdle. Automation primarily depends on precise and consistent data, however, the variety of information sources from many stakeholders might create mistakes and inconsistencies. Additionally, it can be difficult to integrate automation with traditional systems used by architecture companies, necessitating thorough transition planning.

The initial setup of automation tools necessitates time and resource dedication, as well as a learning curve as architects become used to new technology. For smaller businesses with tighter resources, the expense of purchasing and maintaining automation software could be also a barrier. The successful adoption of automation might be complicated by resistance to change among architectural teams accustomed to conventional techniques. Furthermore, the unpredictable and dynamic nature of architecture projects may make it difficult for automated systems to adjust, calling for human intervention (Disney et al., 2022).

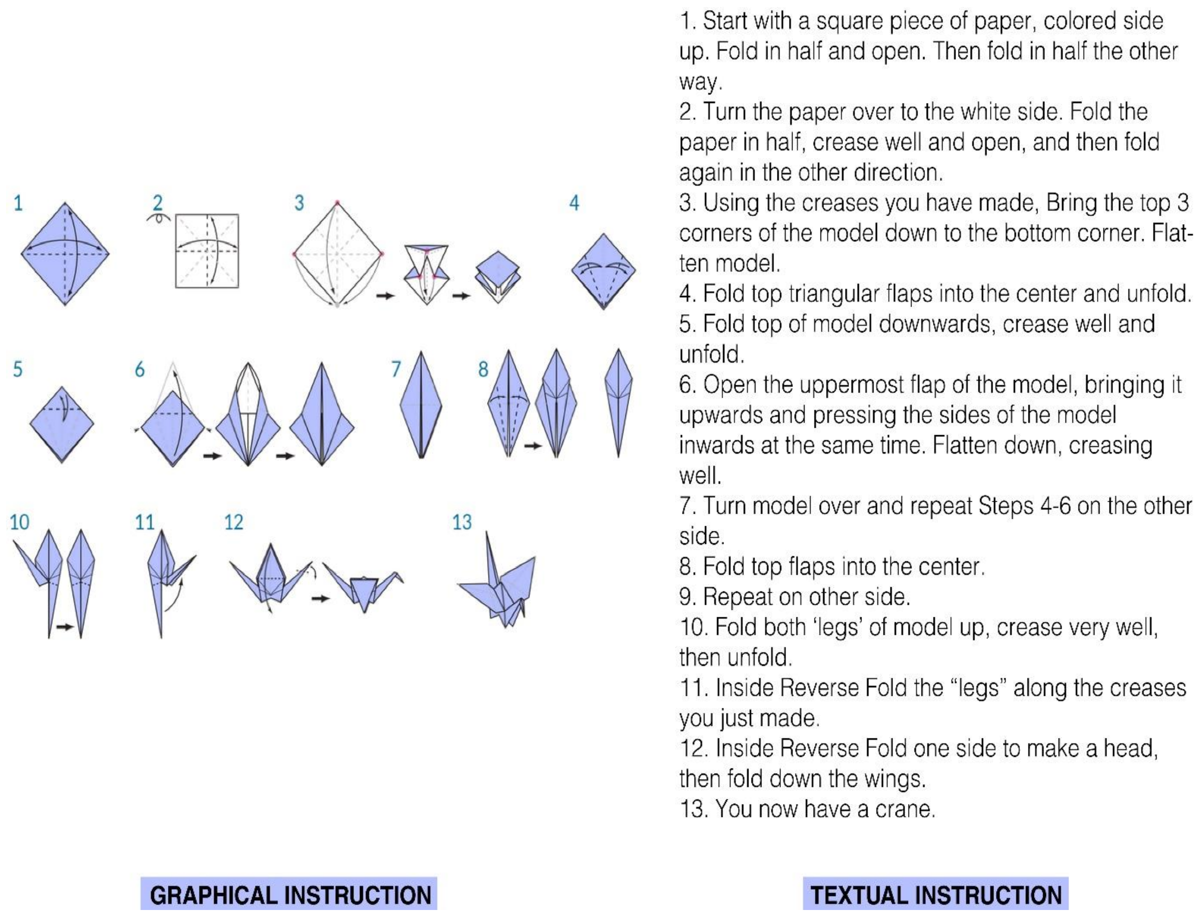
Automated systems must undergo upgrades and troubleshooting to maintain their performance. Over time, ignoring these factors may lead to system malfunctions and diminished effectiveness. Architectural businesses must approach automation strategically to overcome these obstacles, taking into consideration the particular requirements of their company, offering thorough training, and guaranteeing a seamless changeover from manual procedures to automated workflows (Czmoch and Pekala, 2014).

2.3. Development of automation in BIM

There are numerous methodologies for effectuating automation within the context of Building Information Modeling (BIM). The main methods are the utilization of Textual Programming Language (TPL), Visual Programming Language (VPL) Tools, the incorporation of Add-ons and Plug-ins, and the use of BIM softwares. The next sub-chapter delves into the explication of these methodologies, as they are going to be used in this work to develop automation in the design stage.

2.3.1. VPL

Design creation many times involves setting up geometric, visual, or systemic relationships between parts of a project (Brell-Cokcan et al., 2014). Workflows that get from concept to result by a set of rules are used to develop these relationships. By defining a step-by-step set of actions that follow a basic logic of input, processing, and output we are working algorithmically (Terzidis, 2006). Programming allows us to continue working this way by formalizing our algorithms. The algorithms can be defined as the abstract set of steps, that can represent in a couple of ways: textually or graphically (Figure 5) (Terzidis, 2006).

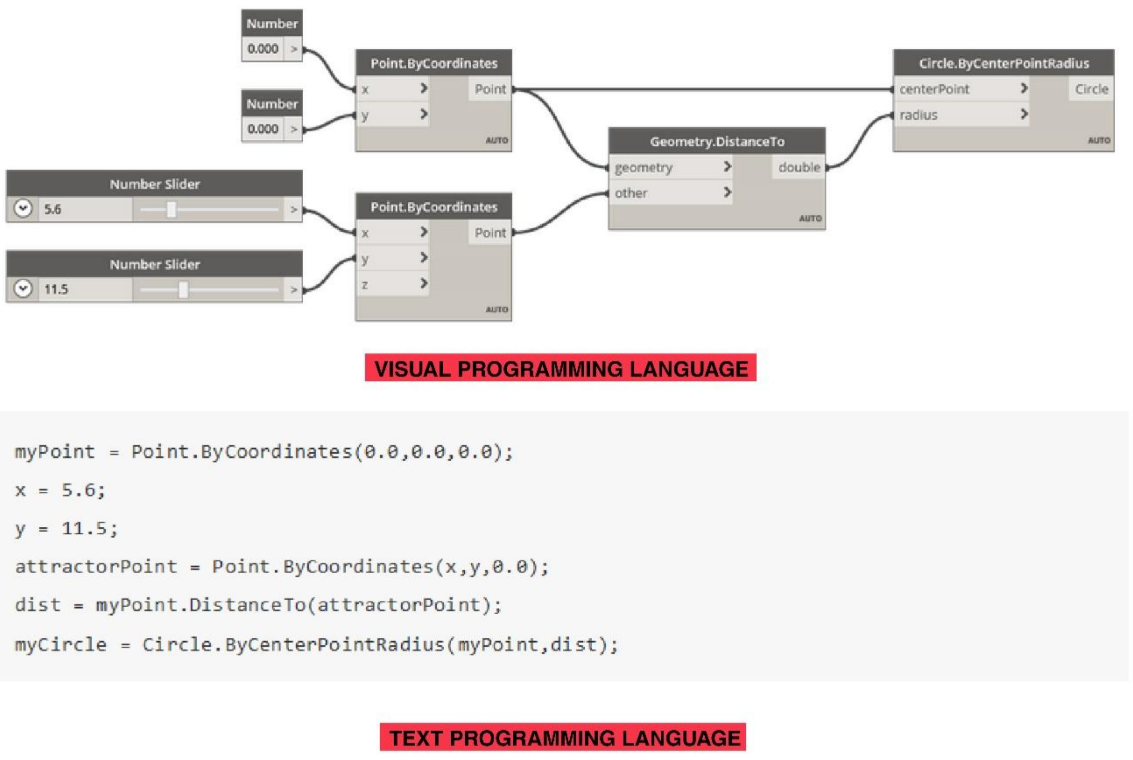


1. Start with a square piece of paper, colored side up. Fold in half and open. Then fold in half the other way.
2. Turn the paper over to the white side. Fold the paper in half, crease well and open, and then fold again in the other direction.
3. Using the creases you have made, Bring the top 3 corners of the model down to the bottom corner. Flatten model.
4. Fold top triangular flaps into the center and unfold.
5. Fold top of model downwards, crease well and unfold.
6. Open the uppermost flap of the model, bringing it upwards and pressing the sides of the model inwards at the same time. Flatten down, creasing well.
7. Turn model over and repeat Steps 4-6 on the other side.
8. Fold top flaps into the center.
9. Repeat on other side.
10. Fold both 'legs' of model up, crease very well, then unfold.
11. Inside Reverse Fold the "legs" along the creases you just made.
12. Inside Reverse Fold one side to make a head, then fold down the wings.
13. You now have a crane.

Figure 5 - Graphical and Textual Instruction (https://primer.dynamobim.org/, 2023)

The process of VPL and TPL is the same, both uses the same framework, the difference is that one uses a visual interface and the other writing-texts.(Monteiro, 2016) VPL is a low-code tool, and it doesn't need assembling code or experience with TPL, this makes it easier to beginners to understand the logic behind it and it is more intuitive to learn how to use this type of language (Aish, 2011). Many VPLs are built on the concept of "boxes and arrows", in which boxes are considered entities with arrows, lines or arcs representing relations connecting them (Dynamo Primer, 2023).

In Figure 6, shows the example of a VPL and a TPL, both examples do the same thing and have the same input and output. VPL main potential is to shorten the learning curve when used to educate inexperienced designers to parametric design and programming (Aish, 2011). However, when several functions are used, graph complexity rises exponentially, and nodes require mode code lines than their text-based equivalents, making the usage of scripting desirable to maintain programming efficiency (Leitão et al., 2011). The multi-paradigm nature of VPL and scripting capabilities not only expand the pool of prospective users but also provide enhanced usability through the fusion of both methods.



```
myPoint = Point.ByCoordinates(0.0,0.0,0.0);
x = 5.6;
y = 11.5;
attractorPoint = Point.ByCoordinates(x,y,0.0);
dist = myPoint.DistanceTo(attractorPoint);
myCircle = Circle.ByCenterPointRadius(myPoint,dist);
```

Figure 6 – Visual and Programming Language

2.3.1.1. Dynamo

Dynamo is an open-source visual programming tool based in the programming language DesignScript, and therefore its technical features and capabilities are inherited from it (Aish, 2013). It's a software developed by Autodesk and already comes integrated inside Revit. Is used in the architecture, engineering, and construction (AEC) industry for computational design, automation, and parametric modelling. One of the potential applications of using Dynamo is the creation of complex objects, which is widely diffused and most used. Furthermore, it has the inherent capacity to automate repetitive modeling activities, documentation, verification, and modeling correction. which overload the work of design professionals, subjecting the process to the occurrence human errors and consequential delays within the workflow (Sena, 2019).

Dynamo’s way of programming consists of nodes connected with each other by lines that create a script that can manipulate various objects in Revit (Kensek, 2014). Each node contains its own programming code, as well as its own function. The nodes can be strings, numbers, inputs, outputs, coordinates, or other types, and if the user modifies one of the nodes, the related nodes will also change. According to the node, every output can be connected to a number of inputs and vice versa; every input can be connected to a number of outputs. Some nodes only have one connectable input or output. By connecting nodes in a visual interface representing various actions, activities, and data items, Dynamo's users can create custom scripts or workflows (Figure 7).

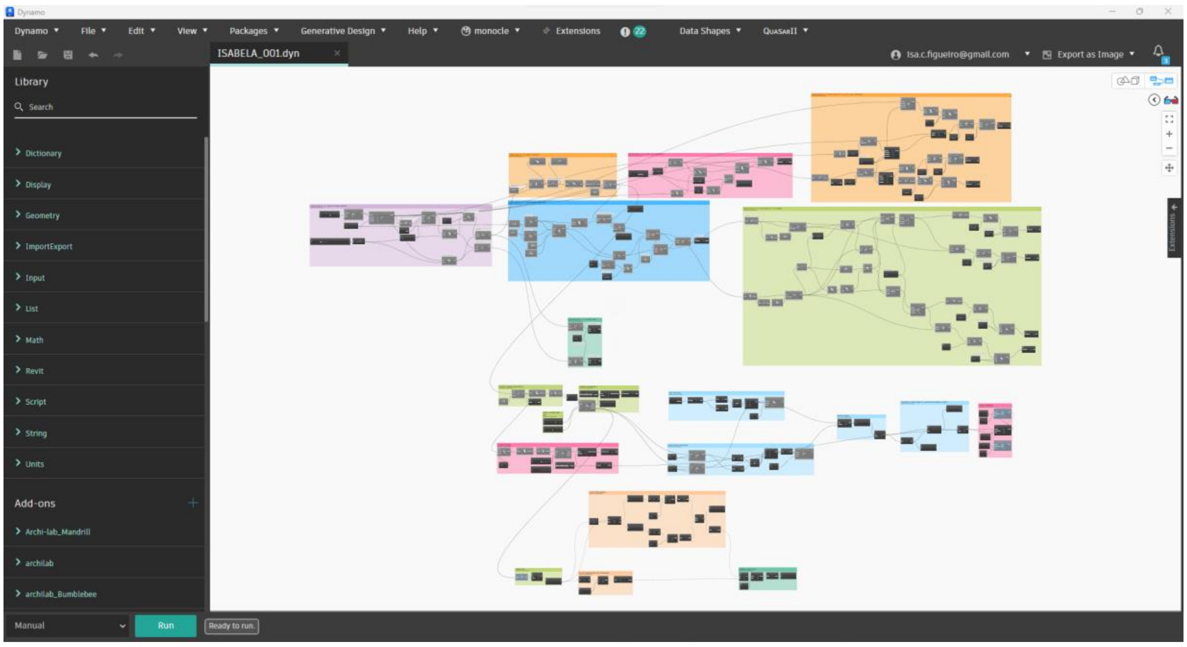


Figure 7 – Exemple of a Dynamo Script

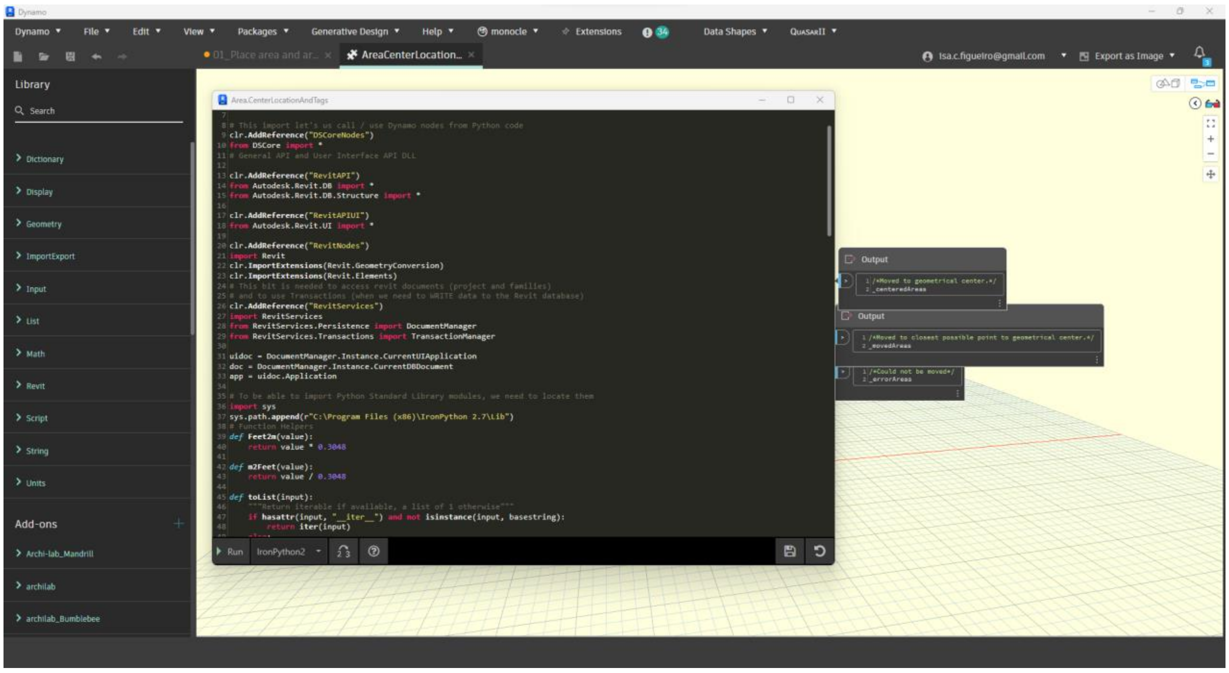
Additionally, Dynamo offers add-ons for download from the software itself. The add-ons include nodes that are created by other users or nodes that are not in the standard version of Dynamo. The software also extracts and reads data from Excel, which allows the exchange of information between them (Sena, 2019).

2.3.1.2. Dynamo and Revit API

Application Programming Interfaces (API) define the foundation for interfacing with the data from the core programs in a way that is integral and not provided by the Graphical User Interface (GUI). Through the use of API, Dynamo is able to access the external Dynamic Link Libraries (DLL), perform and automate connected actions from several programs at once, access the BIM model database, and produce geometry parametrically in the core application. The Revit API enables users to extend core functionalities of Revit (Autodesk, 2023) allowing developers and other skilled users to access the internal guts of the software to create new sub-programs (Kensek, 2014).

There is a public release policy for the Revit API, and it "allows you to program with any.NET compliant language, including VB.NET, C#, and C++/CLI" (Autodesk, 2023) so that existing libraries may be utilized. An ever-more-popular object-oriented Open-Source language, Python is also.NET Common Language Runtime (CLR) compatible (Python.NET, 2023).

Among its other built-in features, Dynamo by default integrates IronPython, a Python implementation. Because of this, this language is used for the majority of the scripting in Dynamo's packages (Figure8).



```
# This import let's us call / use Dynamo nodes from Python code
clr.AddReference("DSCoreNodes")
from DSCore import *
# General API and User Interface API DLL

clr.AddReference("RevitAPI")
from Autodesk.Revit.DB import *
from Autodesk.Revit.DB.Structure import *

clr.AddReference("RevitAPIUI")
from Autodesk.Revit.UI import *

clr.AddReference("RevitNodes")
import Revit
clr.ImportExtensions(Revit.GeometryConversion)
clr.ImportExtensions(Revit.Elements)
# This bit is needed to access revit documents (project and families)
# and to use Transactions (when we need to WRITE data to the Revit database)
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager

uidoc = DocumentManager.Instance.CurrentUIApplication
doc = DocumentManager.Instance.CurrentDBDocument
app = uidoc.Application

# To be able to import Python Standard Library modules, we need to locate them
import sys
sys.path.append(r"C:\Program Files (x86)\IronPython 2.7\Lib")
# Function Helpers
def Feet2m(value):
    return value * 0.3048

def m2Feet(value):
    return value / 0.3048

def toList(input):
    """Return iterable if available, a list of 1 otherwise"""
    if hasattr(input, "__iter__") and not isinstance(input, basestring):
        return iter(input)
```

```
/*Moved to geometrical center.*/
_centeredAreas
```

```
/*Moved to closest possible point to geometrical center.*/
_movedAreas
```

```
/*Could not be moved*/
_errorAreas
```

Figure 8 – Exemple of a python code in Dynamo

2.3.2. Add-ons and Plug-ins

The terms “Add-ons” and “Plug-ins” refer to additional software components that can be integrated into the Revit software to enhance the use of the software and extend its functionality, also provide specific features that might not be included in the core program. They can save time, increase efficiency, and enable users to accomplish processes that might be more complex or time-consuming using only the core Revit software.

2.3.2.1. Nonica

NonicaTab is a Revit Add-on that creates a toolbar with buttons to put a Dynamo script on it. Once the script is uploaded and run once, it gets archived in the Add-In and in the future times that the script is run, it's faster (Nonica, 2023). It runs Dynamo in the background pops up input windows and notifies if the script was run successfully or with errors or warnings. When the script has an error is possible to click in the notification cloud and find out which specific nodes and messages brought the error. It has two types of versions, the Free version, and the PRO version (paid version). In the free version is possible to upload 12 scripts and in the PRO version 36 scripts, besides that, other features are available in the PRO version, such as export/import a .nonica file, a playlist to create a list of Dynamo scripts that would be run one after the other (Nonica, 2023). The Add-on creates a tab in Revit for the user (Figure 9).

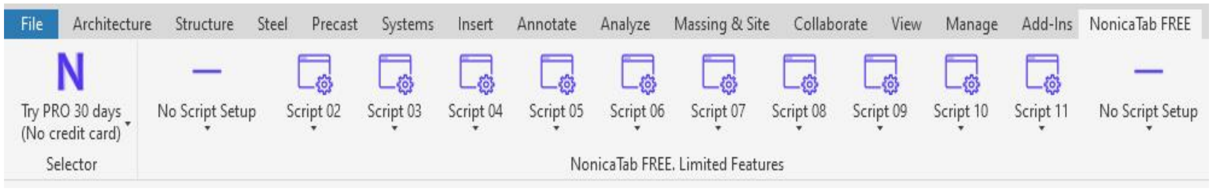


Figure 9 -Nonica Tab Toolbar in Revit

2.3.2.2. Glyph

Glyph is a Revit plug-in that automates and standardizes multiple documentation processes, they are: Automate dimensioning, automate tagging, automate view creation, and automate sheet creation (EvolveLab, 2023). Glyph also provides customizable bundles that run multiple processes together at the same time and it's possible to share the bundle settings between projects. and synchronizes custom processes among projects (Figure 10). Using the automated dimension is possible to dimension multiple views at the same time and also dimension the views in the sheets (EvolveLab, 2023).

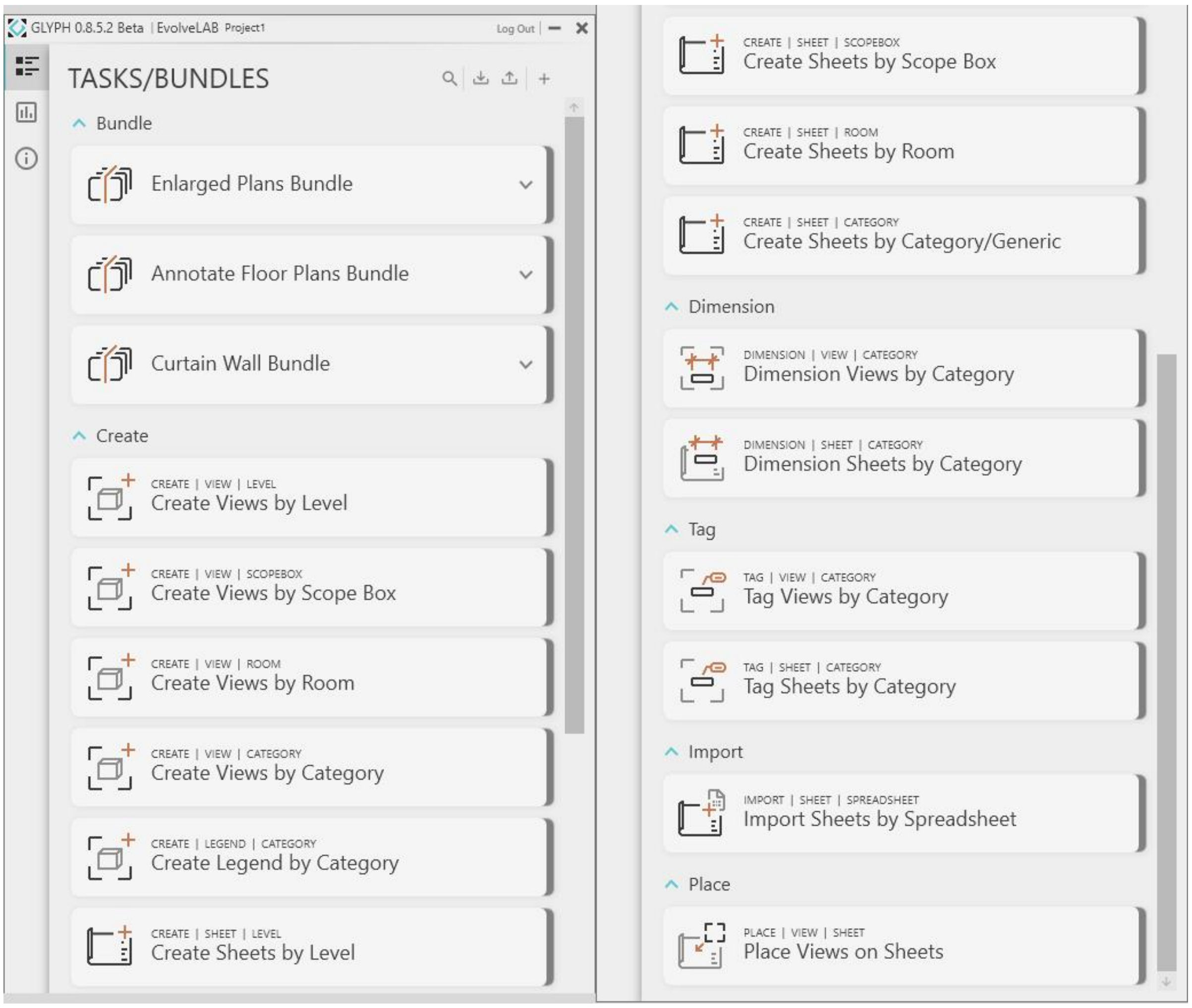


Figure 10 – Glyph plug-in tasks options in Revit

2.3.2.3. DiRoots One

DiRoots One is a single application bundle that includes 8 free Autodesk Revit Plugins to boost your productivity. It allows seamless interoperability between our plugins, opening doors to new workflows and ways to get your job done faster and better (DiRoots, 2023). The 8 plug-ins are SheeLink, FamilyRevisor, OneFilter, TableGen, SheetGen, ReOrdering, ParaManager, and PointKit (Figure 11).

The plug-ins of interest in this thesis are ReOrdering and SheetGen, with ReOrdering, is possible to renumber Revit elements/instance parameters by using a prefix, a suffix, and a multiplier. It is possible to do it randomly, one by one, or using a detail line. And with SheetGen it allows the user to batch-create Revit sheets, place views on the sheets based on a pre-defined template, and easily manage the sheet revisions (DiRoots, 2023).

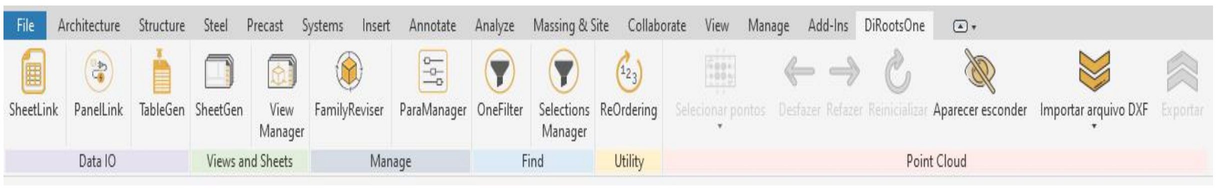


Figure 11 – DiRootsOne Toolbar in Revit

2.3.2.4. Archilizer – Tiny Tools

Tiny Tools is a free add-on with a collection of a few simple, but very powerful, tools for Autodesk Revit. “The tools are useful to draftsmen who want to shrink production time significantly” (Autodesk, 2023). The set includes 5 tools (Figure 12) that can be used in combination to be more productive in the architectural office. With these tool is possible to automatically duplicate views and sheets, rename them, create sheets from views, and align views on sheets (Autodesk, 2023).

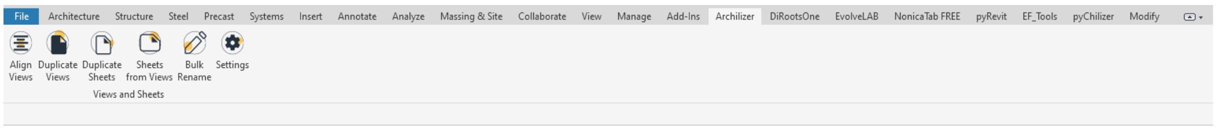


Figure 12 – Tine Tools Toolbar in Revit

2.3.2.5. PyRevit

PyRevit is an open-source, free-to-use plug-in for Autodesk Revit, which means it is actively developed and maintained by a community of contributors. It contains a collection of tools, scripts, and functionalities that enhance the capabilities of Revit, improve productivity, and streamline various workflows for users (pyRevit, 2023). It allows users to create their own Python scripts and Add-ons ideas to automate and optimizes repetitive tasks and checks the model for errors and collisions, in whichever programming language inside the Revit environment and using its APIs (Application Programming Interface). Inside of pyRevit is possible to download extensions (Figure 13) from other developers that used pyRevit to create their scripts. Overall, pyRevit is a strong Autodesk Revit plugin that enhances functionality, speeds up workflows, and gives users the ability to personalize and enhance their BIM experience (pyRevit, 2023).

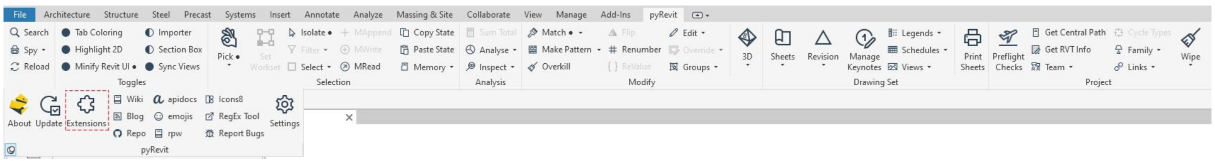


Figure 13 – PyRevit Toolbar in Revit

2.3.2.6. Ef-Tools

EF-Tolls is a free pyRevit extension for Autodesk Revit, developed by Erik Frits, and it has tools that automate tasks to solve issues and improve workflows. The extension website says, “Revit is missing a lot of features we need, but hopefully you will find a few tools in EF-Tools that will make you more efficient in your daily work” (Frits, 2023). With the extension (Figure 14)is possible to do the main following tasks: align viewports, create graphic overviews, copy view filters, room to floor/region, replace materials, isolate warnings, many selection tools, create legend from view filters, many renaming tools, open select DWG, hide revision clouds, transfer view templates, match graphic overrides, wall match constraints, attached groups tools, add level elevation, renumber parking, sheet revision history, text transform, duplicate sheets, and more (Frits, 2023).

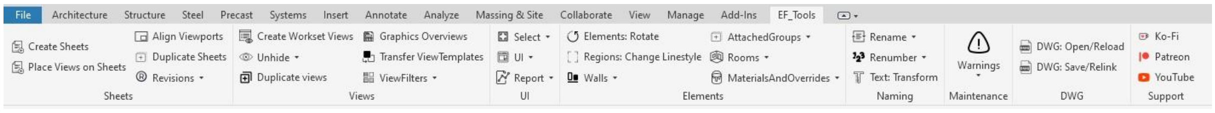


Figure 14 – Ef-Tools Toolbar in Revit

3. IMPLEMENTATION OF SUB-PROCESSES AUTOMATION

3.1. Survey of most relevant sub-processes to automate

The implementation of automation within Building Information Modeling (BIM), particularly during the Design Stage, as elucidated in the preceding chapters, constitutes a method by which architects expedite their work processes while still maintaining high standards of quality in their deliverables. The objectives developed within this study have the mean focus on the enhancement of the operational flow at NOZ Arquitectura, particularly during the phases of modeling and the subsequent generation of 2D drawing-based deliverables. To define the specific sub-processes, a series of collaborative sessions were held with the Head of the company and its associates. The main focus of this dissertation was on the sub-processes that are related to the 2D deliverables, but during the sessions with the company, as is possible to see in the table, some sub-processes related to 3D modeling were also requested to check about automating them, and they were also developed in the work. A spreadsheet was formulated to systematically catalog the ideas, facilitating an analysis of their pertinence category, to either the 3D modeling phase or the production of 2D drawings and their relevancy. Moreover, this mechanism was deployed to ascertain the significance of these ideas and to validate if they are time-consuming for the personnel involved.

Table 1 - List of Sub-processes ideias with the relevance and the category

Sub-processes Ideas	**Relevance**	**Category**
Alignment of the floor plan views in the same place in different sheets	High	2D Documentation
Insertion of revisions on sheets automatically	Medium	2D Documentation
Creation of dimension annotations automatically	High	2D Documentation
Element renumbering	High	2D Documentation
Make all the automation scripts available in an easy to use	High	-
Alignment of the view name of the viewport in a chosen position	Medium	2D Documentation
Alignment of the view name with the viewport	Medium	2D Documentation
Detection of overlapping information	High	2D Documentation
Centralization of rooms, spaces, areas, and their respective tags	Medium	2D Documentation

Removal of not placed rooms, spaces, areas, and their respective tags	Medium	2D Documentation
Removal of value 0 from dimension	Medium	2D Documentation
Insertion of light switch referenced to a door	High	3D Modeling
Insertion of spot elevation in a view aligned to all room tags	Medium	2D Documentation
Insertion of the stair path	Medium	2D Documentation
Insertion of north symbol in all floor plans	Medium	2D Documentation

As previously underscored, given the enduring significance of 2D deliverables still maintain their significance for regulatory approvals and construction processes, and for that necessitating the inclusion of information within the 2D views. The relevance of the sub-processes, high or medium was determined by having high on those that, in the company opinion, require the most time to do manually and medium on the others.

3.2. Methodology for sub-processes automation

To solve the identified sub-processes, a comprehensive investigation was undertaken to ascertain the availability of existing solutions of add-ons or plug-ins for Revit. In instances where a suitable solution was not found, recourse was taken to the creation of scripts in Dynamo (Figure 15). It is noteworthy to emphasize that the pursuit of add-ons and plug-ins was specifically for those designed to work with Revit, while the scripts were developed within the Dynamo due to its connection with Revit, the BIM software used by NOZ Arquitectura.

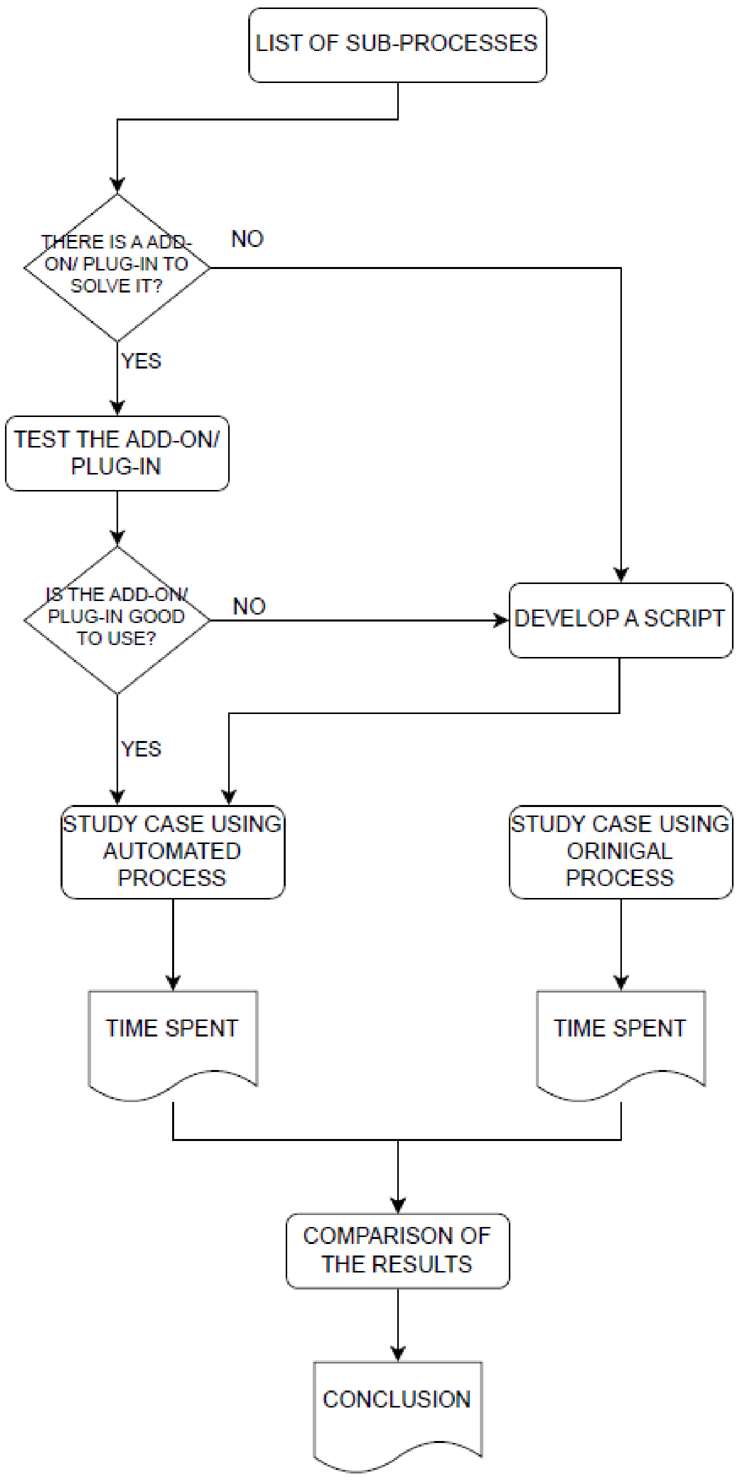

LIST OF SUB-PROCESSES
THERE IS A ADD-ON/ PLUG-IN TO SOLVE IT?
NO
YES
TEST THE ADD-ON/ PLUG-IN
IS THE ADD-ON/ PLUG-IN GOOD TO USE?
NO
DEVELOP A SCRIPT
YES
STUDY CASE USING AUTOMATED PROCESS
STUDY CASE USING ORINIGAL PROCESS
TIME SPENT
TIME SPENT
COMPARISON OF THE RESULTS
CONCLUSION

Figure 15 – Process Map of the methodology

After a search of potential solutions within plug-ins and add-ons, it was discerned that certain sub-processes could effectively be done through plug-ins and add-ons. On the other hand, for the remaining sub-processes, the attainment of resolution necessitated the creation of bespoke scripts utilizing Dynamo. In the Table 2 is the list of sub-processes and what is the way that they were resolved.

Table 2 - Sub-processes list with and the method that were solved

Sub-processes	**Add-on/Plug-in or script?**
Alignment of the floor plan views in the same place in different sheets	Add-on: Tiny Tools
Insertion of revisions on sheets automatically	Add-on: SheetGen
Creation of dimension annotations automatically	Plug-in: Glyph
Element renumbering	Plug-in: ReOrdering
Make all the automation scripts available in an easy to use	Plug-in: Nonica
Alignment of the view name of the viewport in a chosen position	Script needed
Alignment of the view name with the viewport	Script needed
Detection of overlapping information	Script needed
Centralization of rooms, spaces, areas, and their respective tags	Script needed
Removal of not placed rooms, spaces, areas, and their respective tags	Script needed
Removal of value 0 from dimension	Script needed
Insertion of light switch referenced to a door	Script needed
Insertion of spot elevation in a view aligned to all room tags	Script needed
Insertion of stair path	Script needed
Insertion of north symbol in all floor plans	Script needed

3.3. Sub-processes solved with add-ons or plug-ins

This chapter elucidates the sub-processes that have been successfully resolved through the utilization of add-ons or plug-ins, which were subject to comprehensive investigation and analysis in Chapter 2, encompassing the Literature Review.

3.3.1. Alignment of the floor plan views in the same place on different sheets

In the context of an architectural project, the spatial arrangement of floor plans on respective sheets bears significance due to its potential to facilitate a coherent and uninterrupted visual progression when perusing through the sequential sheets.

In instances where automated mechanisms are not employed, an approach to solving it is the creation of guidelines as points of reference. This involves the replication of these guidelines across all pertinent sheets, subsequently moving the viewport in accordance with these reference markers to get all of them aligned. In projects characterized by a limited number of levels, this method of creating guidelines might not have so much time wasted. However, the complexity enlarges significantly when it is an architectural project with many levels. The use of guidelines to move the viewport to get everything aligned gives rise to a noteworthy expenditure of time and effort.

To address this challenge, the Tiny Tools (Autodesk, 2023) Add-on presents a pragmatic solution by furnishing an automated functionality. This feature within the Tiny Tools Add-on obviates the need for manual interventions by doing the alignment of floor plans on respective sheets. By using this automated feature, the time used compared to the manual alignment of numerous floor plans is effectively decreased, thereby enhancing the efficiency and expediency of the architectural project workflow.

When clicking on tab of Archilizer in Revit, it shows all the functions that the Add-on has (Figure 16), and when using the Align Views function (Figure 17), it is necessary to select which viewport will serve as a reference (1) and after the sheets that the viewports will be aligned with reference (2) and after running the plug-in (3). With this add-on, the viewport and the viewcrop will stay the same, and even if they have different sizes they will be aligned in the same place.

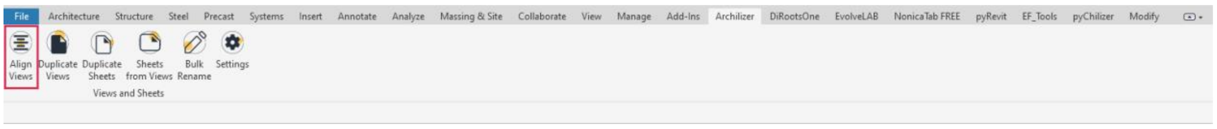
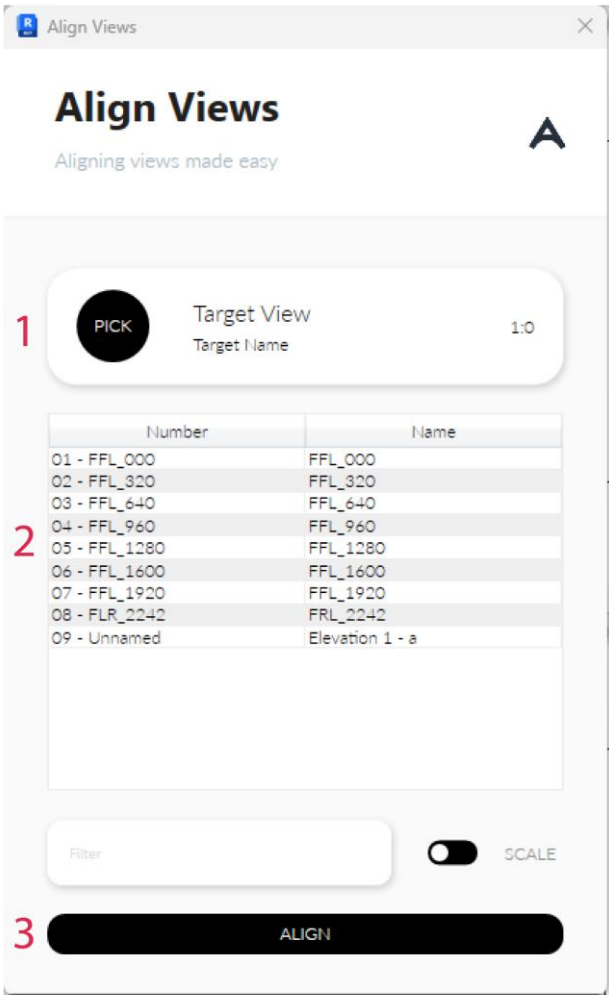


Figure 16 – Align Views function on Tiny Tools Toolbar in Revit


Align Views

Align Views

Aligning views made easy

1

PICK

Target View

Target Name

1:0

2

Number	Name
01 - FFL_000	FFL_000
02 - FFL_320	FFL_320
03 - FFL_640	FFL_640
04 - FFL_960	FFL_960
05 - FFL_1280	FFL_1280
06 - FFL_1600	FFL_1600
07 - FFL_1920	FFL_1920
08 - FLR_2242	FRL_2242
09 - Unnamed	Elevation 1 - a

Filter

SCALE

3

ALIGN

Figure 17 – The pop-up window to Align Views of the plug-in Tiny Tools

In the subsequent images, Figure 18 and Figure 19, is possible to see that in the sheets marked in blue and green, the view crops are different from the reference marked in red. In these instances, a noteworthy occurrence happens, when the plug-in runs even with different view crops, and consequently, different viewports. The alignment of the building is executed perfectly, in all the sheets that the plug-in ran, and the building/project is in the same place. From my point of view, th probably means that the plug-in uses a point of reference in the project to align the views, and not the viewport itself. That is what I would do to develop an automation to do this sub-process. Because the viewport and the viewcrop is not always the same in every view of the project.

Notably, alternative plugins with the same functionality underwent evaluation. These plugins were proven to produce successful alignment only when the viewcrop dimensions lined up with those of the reference viewport. These plugins provide the option of applying the same viewcrop dimensions, however, this uniformity may not always coincide with the project's diverse requirements, necessitating different viewcrop settings.

Additional efforts were made to make use of scripts on the Dynamo platform. However, the alignment, which was based on viewport properties, became the key problem. Therefore, even slight differences in the view crop or elements outside the view crop were sufficient to prevent accurate alignment within the project.

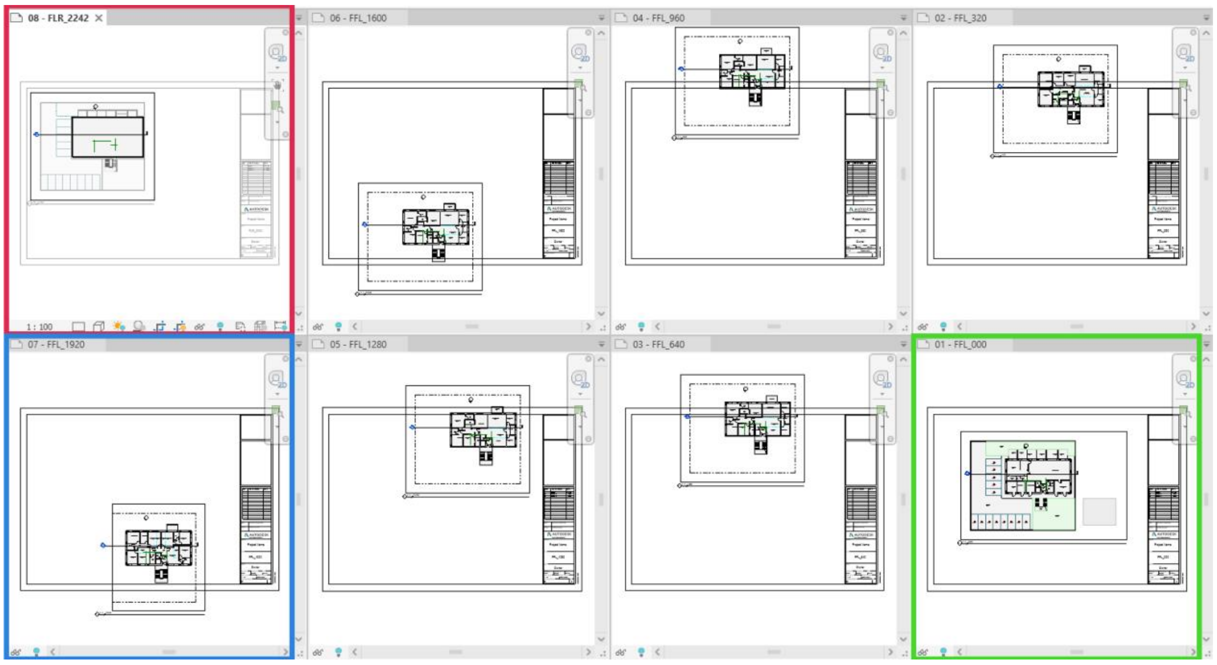


Figure 18 – Workspace of Revit with the sheets before running the plug-in

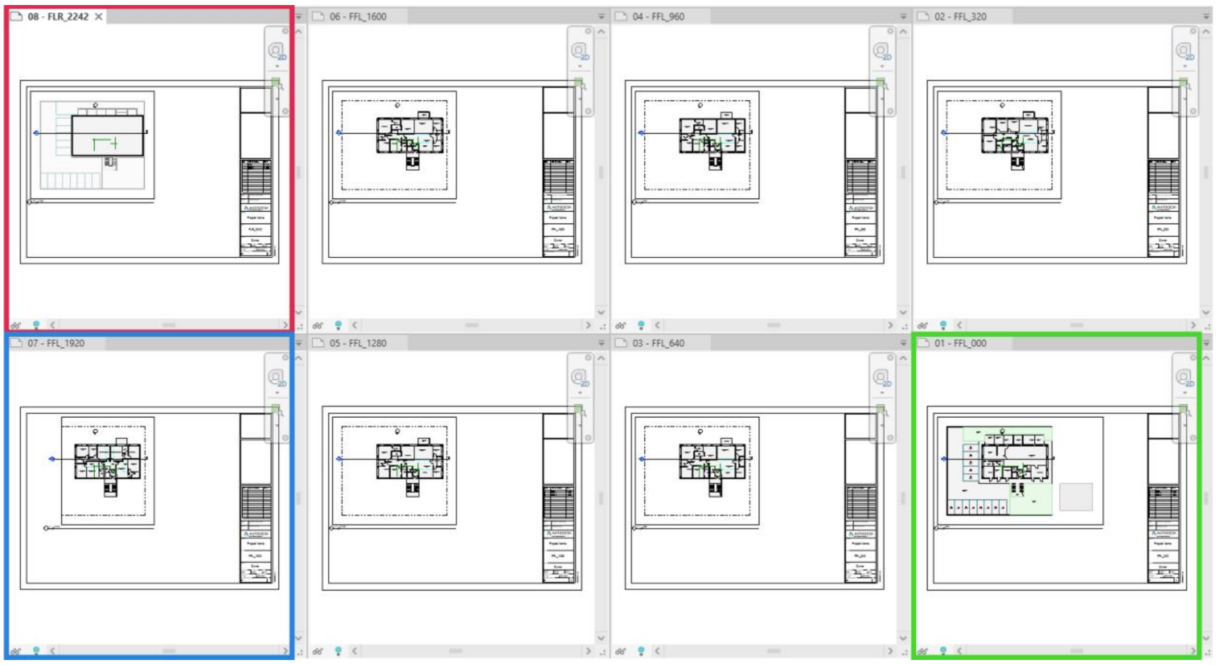


Figure 19 - Workspace of Revit with the sheets after running the plug-in

3.3.2. Insertion of revisions on sheets automatically

In the realm of architectural documentation delivery, the inclusion of pertinent information regarding revisions is a requisite role for maintaining an accurate and up-to-date record of design modifications. In the evolution of Autodesk's Revit software, specifically within versions 2023 and 2024, there is already a capability of placing the revision data across a selection of sheets. This innovation effectively streamlines the revision annotation process, necessitating a unique execution for the selection of sheets. However, it is noteworthy that this enhancement is not retroactively applicable to preceding iterations of Revit, from the 2022 version and its precursors. In these earlier versions, the process of placing revision entails an obligatory per-sheet approach.

The sub-process of individually designating revisions to numerous sheets, characteristic of the antecedent Revit versions, is a time waste for the collaborators, especially when is necessary to place or take off the revision in a large number of sheets.

It is in this context that the intervention of the SheetGen (DiRoots, 2023) plugin, developed by DiRoots, can be used to help this sub-process be done quickly. SheetGen proffers an efficacious solution to this sub-process by letting the user be capable of selectively designating or removing revisions on the sheets by choosing the revision and the sheet.

The DiRootsOne (DiRoots, 2023) tab on Revit, show all the functions of the plug-in, as shown on Figure 20. After open SheetGen (DiRoots, 2023) is possible to see that the function is user-friendly, boasting an intuitive interface that makes it a breeze to navigate. Figure 21 shows that the plug-in enables the management of revisions (Step 1), facilitates importing and exporting through Excel files (Step 2), allows placement of revisions onto one or multiple sheets (Step 3), and finally, applies to the project (Step 4).

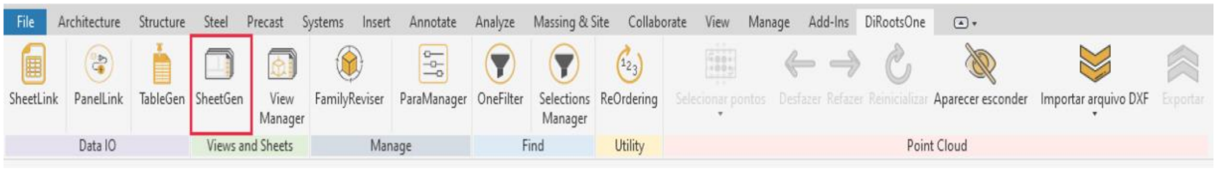
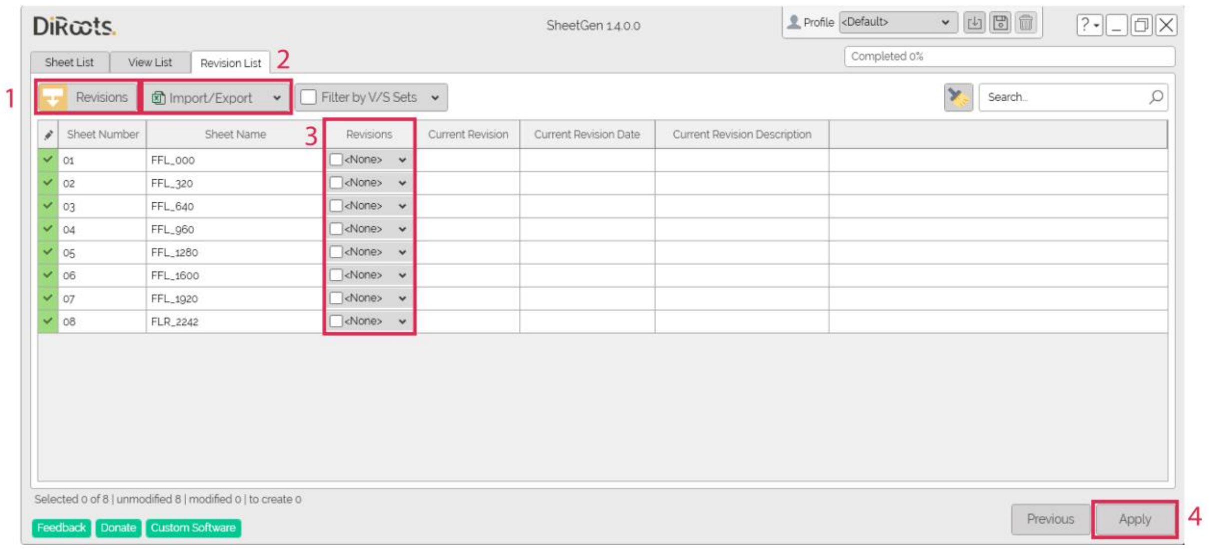


Figure 20 – DiRoots One Toolbar in Revit with the plug-in SheetGen in highlight

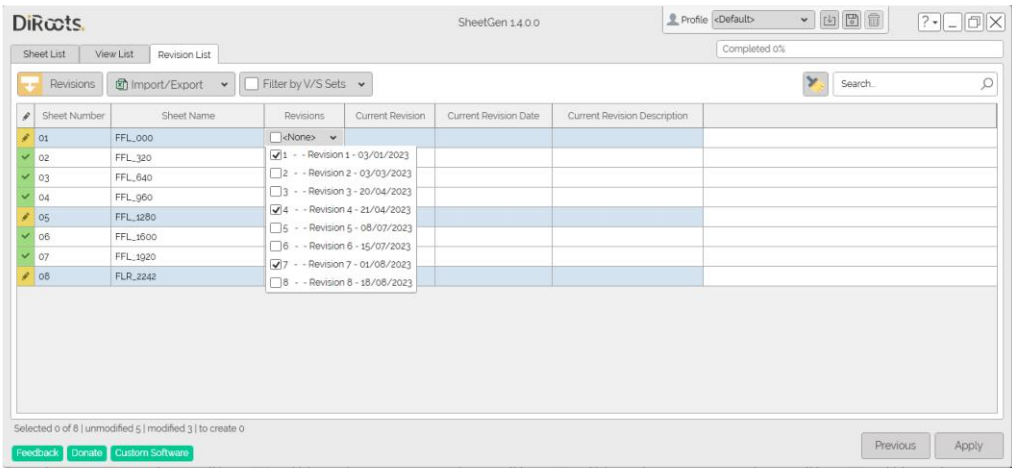


	Sheet Number	Sheet Name	Revisions	Current Revision	Current Revision Date	Current Revision Description
✓	01	FFL_000	☐ <None>			
✓	02	FFL_320	☐ <None>			
✓	03	FFL_640	☐ <None>			
✓	04	FFL_960	☐ <None>			
✓	05	FFL_1280	☐ <None>			
✓	06	FFL_1600	☐ <None>			
✓	07	FFL_1920	☐ <None>			
✓	08	FLR_2242	☐ <None>			

Selected 0 of 8 | unmodified 8 | modified 0 | to create 0

Figure 21 – Revision option in DiRoots SheetGen plug-in

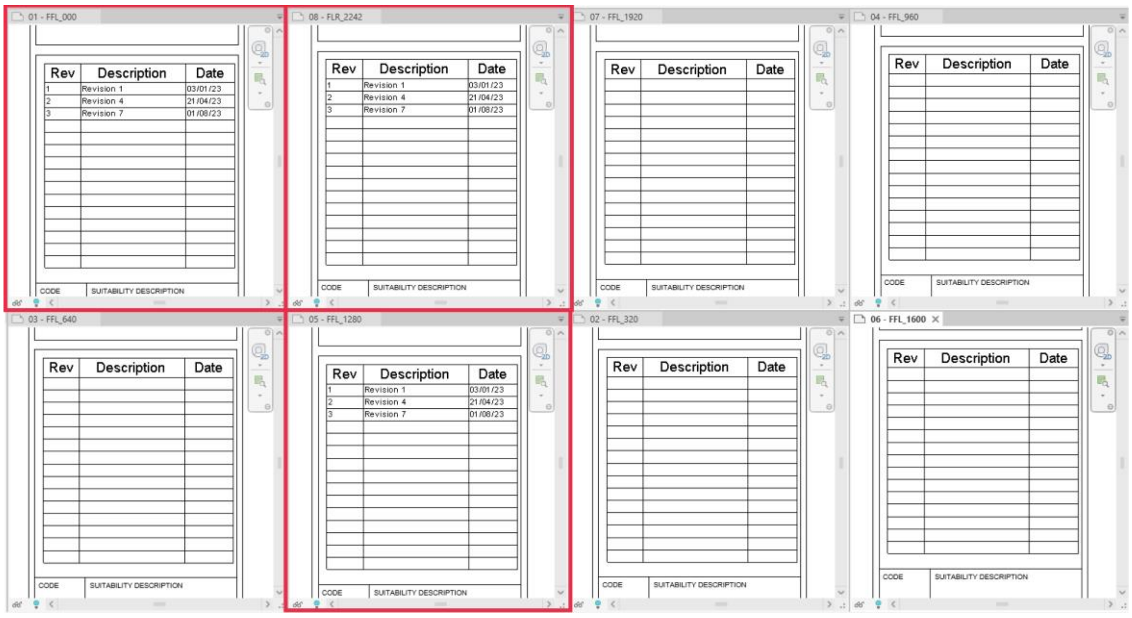
Figure 22 shows the use of the plug-ing, placing the revision 1, 4 and 7 on the sheets 01-05-08 and Figure 23 shows the result in the sheets after running the plugin.



	Sheet Number	Sheet Name	Revisions	Current Revision	Current Revision Date	Current Revision Description
✎	01	FFL_000	<None>			
✓	02	FFL_320	[✓] 1 - - Revision 1 - 03/01/2023			
✓	03	FFL_640	2 - - Revision 2 - 03/03/2023			
✓	04	FFL_960	3 - - Revision 3 - 20/04/2023			
✎	05	FFL_1280	[✓] 4 - - Revision 4 - 21/04/2023			
✓	06	FFL_1600	5 - - Revision 5 - 08/07/2023			
✓	07	FFL_1920	6 - - Revision 6 - 15/07/2023			
✎	08	FLR_2242	[✓] 7 - - Revision 7 - 01/08/2023			
			8 - - Revision 8 - 18/08/2023			

Selected 0 of 8 | unmodified 5 | modified 3 | to create 0

Figure 22 – Revisions in DiRoots SheetGen



Rev	Description	Date
1	Revision 1	03/01/23
2	Revision 4	21/04/23
3	Revision 7	01/08/23

Figure 23 – The Revisions placed in the selected sheets

3.3.3. Creation of dimension annotations automatically

The dimensions annotations play a significant role in the context of architectural design and is a fundamental element for the accurate and communicative representation of building dimensions. Through dimensions, it is possible to clearly define the dimensions, proportions, and spatial relationships of each element present in a drawing. They provide a common language for describing sizes, distances, and heights, making it easier for all project stakeholders to understand the intended dimensions. Lack of precision in dimensions can lead to errors during construction. Dimensions ensure that builders adhere accurately to the project specifications, reducing the likelihood of rework, unplanned changes, and delays. When architectural drawings are used for producing construction documents, dimensions ensure that contractors and subcontractors understand the necessary dimensions for carrying out the work. This results in more comprehensive construction documents and fewer misinterpretations.

To create dimensions in Revit, there are built-in tools within the software that help create them more quickly, though not always very accurately. Users often need to review and adjust them to accurately dimension everything they desire. Adding dimension annotations is a sub-process that usually takes a lot of time, and even when copied from one level to another, as in a typical floor plan, errors in reference can occur, causing some dimensions to be displayed as 0.

A more practical way to create these dimensions would be automatically, by identifying wall faces and dimensioning all of them in a view. With the Glyph (EvolveLab, 2023) plug-in, it's possible to create dimensions using two methods: automatically or by drawing a line along which you want the dimension to appear. This can be done in either the view itself or on sheets. In the plugin, you can choose which elements (one or more) the dimension should measure, shown in Figure 24, and which type of reference the dimension should use. For example, for walls, you can choose which face or element of the wall the dimension should measure, as shown in Figure 25. You can also choose the type of tie condition, the offset the dimension should have from the element, and the dimension family. Furthermore, to be even more precise, the plug-in gives the option to run it avoiding collision.

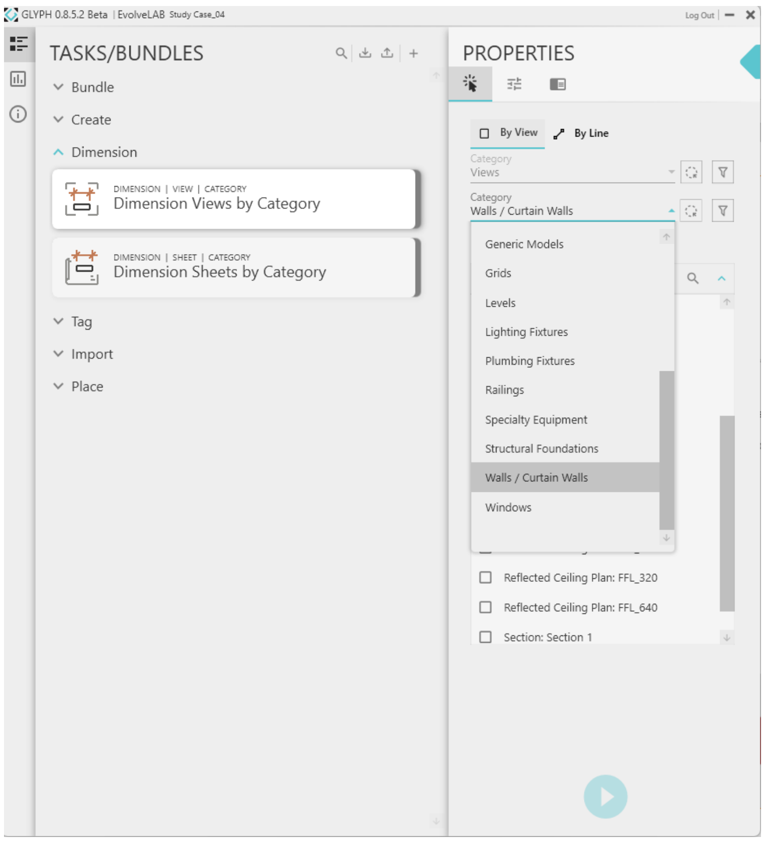
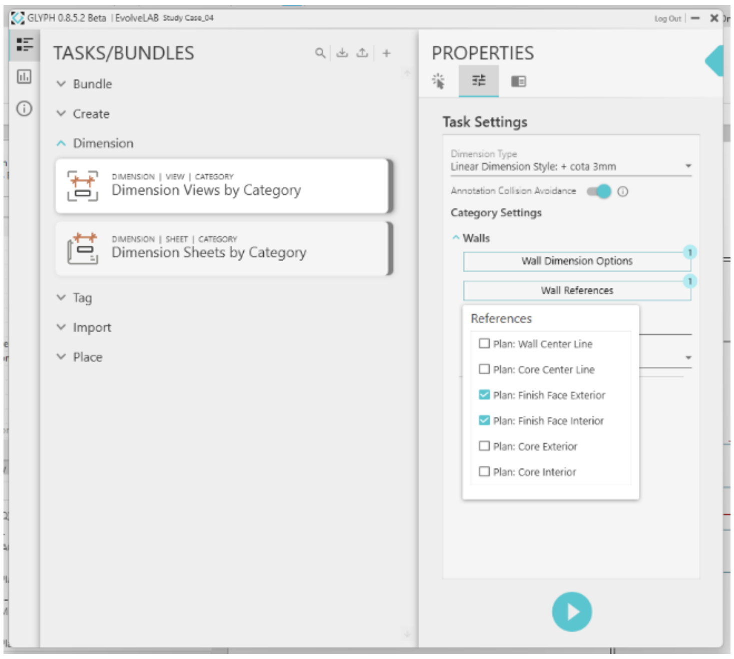


Figure 24 – The options of what to place the dimension annotation with Glyph plug-in


GLYPH 0.8.5.2 Beta | EvolveLAB Study Case_04
Log Out
TASKS/BUNDLES
Bundle
Create
Dimension
DIMENSION | VIEW | CATEGORY
Dimension Views by Category
DIMENSION | SHEET | CATEGORY
Dimension Sheets by Category
Tag
Import
Place
PROPERTIES
Task Settings
Category Settings
Walls
Wall Dimension Options
Wall References
1
1

Dimension Type
Linear Dimension Style: + cota 3mm

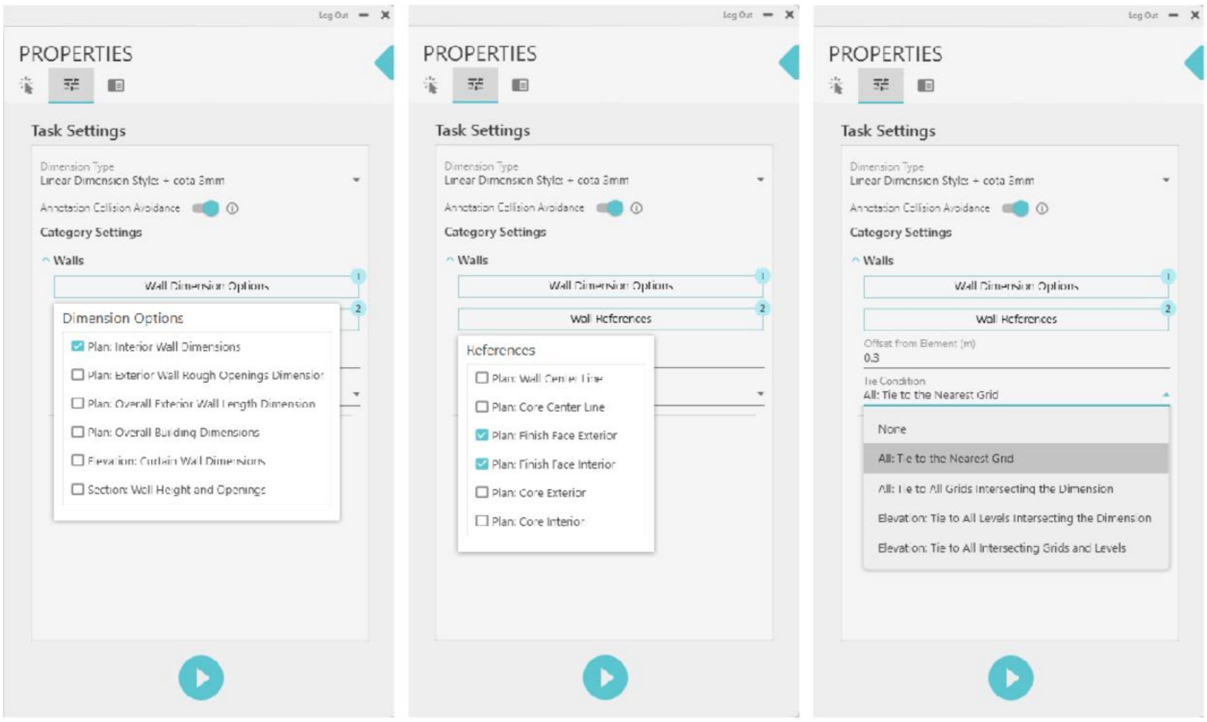
Annotation Collision Avoidance ●

References

- ☐ Plan: Wall Center Line
- ☐ Plan: Core Center Line
- ☑ Plan: Finish Face Exterior
- ☑ Plan: Finish Face Interior
- ☐ Plan: Core Exterior
- ☐ Plan: Core Interior

Figure 25 – The Wall References options in the plug-in Glyph

To show how the plug-in works, it was chosen to test in a floor plan view and in a section, using both ways that the plug-in has as an option, automatically and with the line. In the floor plan it was selected to dimensioning the walls, and in the section, to dimensioning the floors. To configurate how the plug-in should work, in the Figure 26 is possible to see that the Wall Dimension Option was chosen the one that only dimensioning the walls and not any openings, the refence is the finishing face interior and exterior and all tied to the nearest grid. For using the grid, was necessary to place them in the projects. The Annotation Collision Avoidance is also on. Figure 27 shows the floor plan before running the plug-in, while Figure 28 shows the same floor plan after running the plug-in in automatically way, and Figure 29 after running with the line option.



Dimension Type
Linear Dimension Style: + cota 3mm
Annotation Collision Avoidance
Category Settings
Walls
Wall Dimension Options
Dimension Options
- [x] Plan: Interior Wall Dimensions
- [ ] Plan: Exterior Wall Rough Openings Dimension
- [ ] Plan: Overall Exterior Wall Length Dimension
- [ ] Plan: Overall Building Dimensions
- [ ] Elevation: Curtain Wall Dimensions
- [ ] Section: Wall Height and Openings

Wall References
References
- [ ] Plan: Wall Center Line
- [ ] Plan: Core Center Line
- [x] Plan: Finish Face Exterior
- [x] Plan: Finish Face Interior
- [ ] Plan: Core Exterior
- [ ] Plan: Core Interior

Offset From Element (m)
0.3
Tie Condition
All: Tie to the Nearest Grid
- None
- All: Tie to the Nearest Grid
- All: Tie to All Grids Intersecting the Dimension
- Elevation: Tie to All Levels Intersecting the Dimension
- Elevation: Tie to All Intersecting Grids and Levels

Figure 26 – The procedure to set up Glyph plug-in to use it in walls

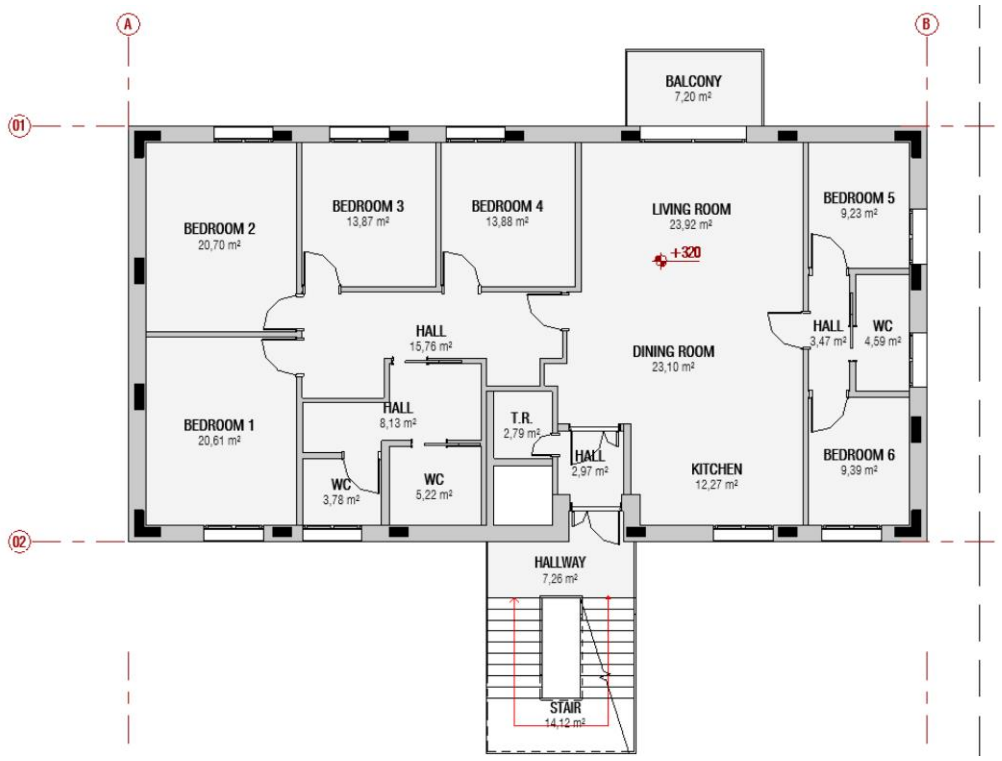


Figure 27 – Floor plan before running the plug-in

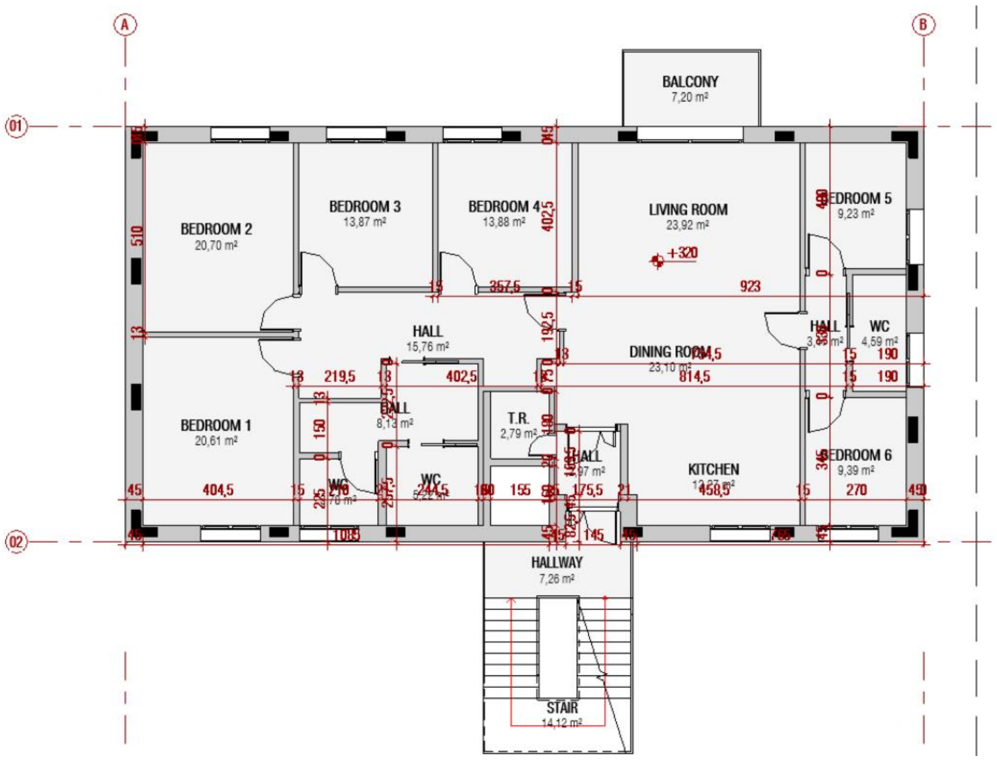


Figure 28 – Floor plan after running the plug-in to place dimensions annotation in walls using By View mode

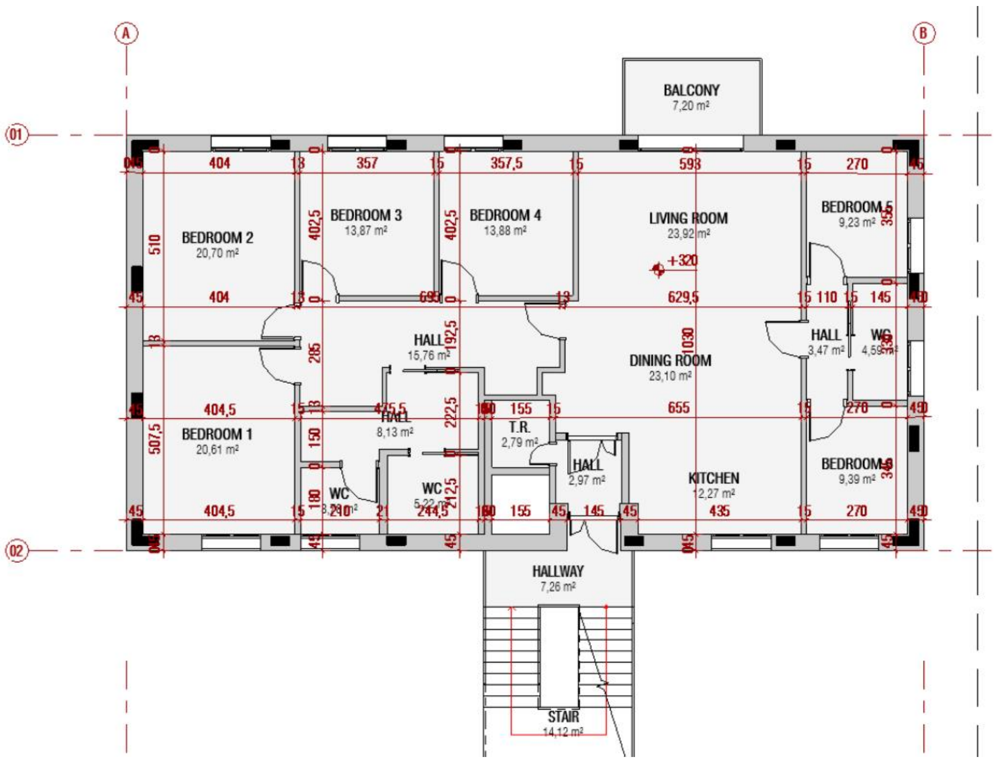
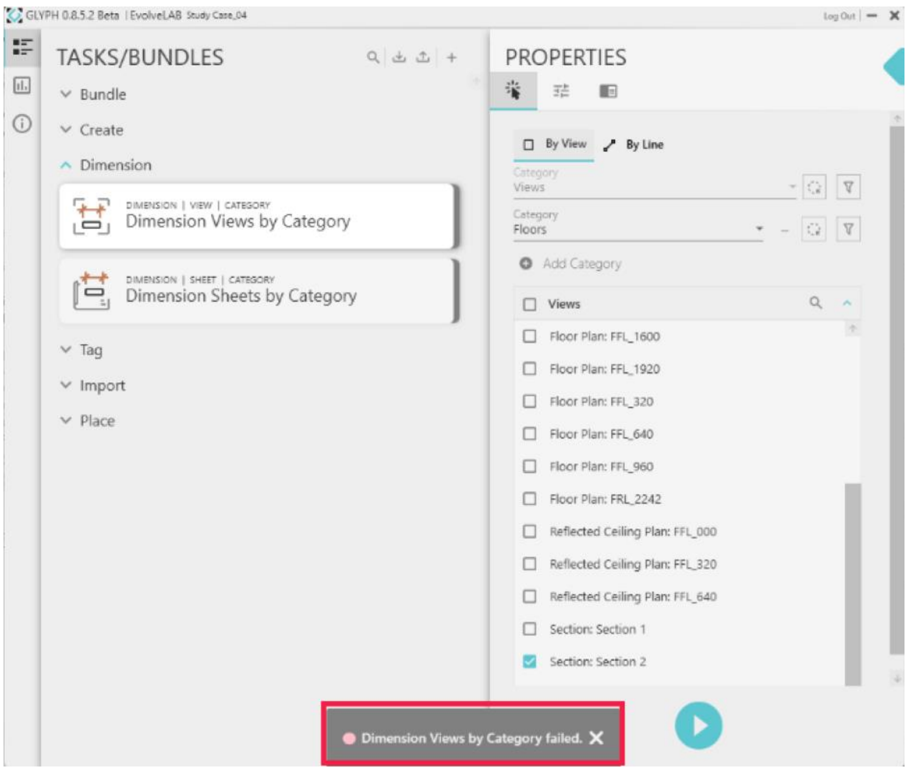


Figure 29 – Floor plan after running the plug-in to place dimensions annotation in walls using By Line mode

By utilizing an automated approach, it becomes evident that the arrangement of dimensions in certain areas lacks harmony. Some dimensions possess a value of 0, neglecting to account for certain walls, while others redundantly measure the same elements. Despite employing the Annotation Collision Avoidance feature, some dimensions still intersect. When utilizing a manual approach like lines, users can selectively position dimensions, resulting in a drawing that is more coherent and legible. It's worth noting that even when using lines as a guide, instances of 0 dimensions still arise. One positive aspect of utilizing a manual approach with lines for dimension placement in this scenario is the enhanced level of control and precision it offers.

When attempting to employ the plug-in for section dimensioning in automatic mode, using the floor as the dimension reference, an issue arises. Upon executing the plug-in, an error message "Dimension Views by Category failed" is displayed (Figure 30). Consequently, it becomes necessary to modify the category to "levels." This adjustment enables the successful execution of the plug-in. The section's state prior to running the plug-in is depicted in Figure 31, while Figure 32 illustrates the section after its execution, with "levels" used as the reference an in the By View mode. Despite encountering an error message initially, users have the ability to modify the category to "levels" as a workaround. This flexibility allows users to find alternative solutions and still achieve their desired results, ultimately saving time.


GLYPH 0.8.5.2 Beta | EvolveLAB Study Case_04
Log Out
TASKS/BUNDLES
Bundle
Create
Dimension
DIMENSION | VIEW | CATEGORY
Dimension Views by Category
DIMENSION | SHEET | CATEGORY
Dimension Sheets by Category
Tag
Import
Place
PROPERTIES
By View
By Line
Add Category
Views

Category
Views

Category
Floors

- [ ] Floor Plan: FFL_1600
- [ ] Floor Plan: FFL_1920
- [ ] Floor Plan: FFL_320
- [ ] Floor Plan: FFL_640
- [ ] Floor Plan: FFL_960
- [ ] Floor Plan: FRL_2242
- [ ] Reflected Ceiling Plan: FFL_000
- [ ] Reflected Ceiling Plan: FFL_320
- [ ] Reflected Ceiling Plan: FFL_640
- [ ] Section: Section 1
- [x] Section: Section 2

Dimension Views by Category failed.

Figure 30 – Error message in the plug-in after trying to use in a section view with the By View mode

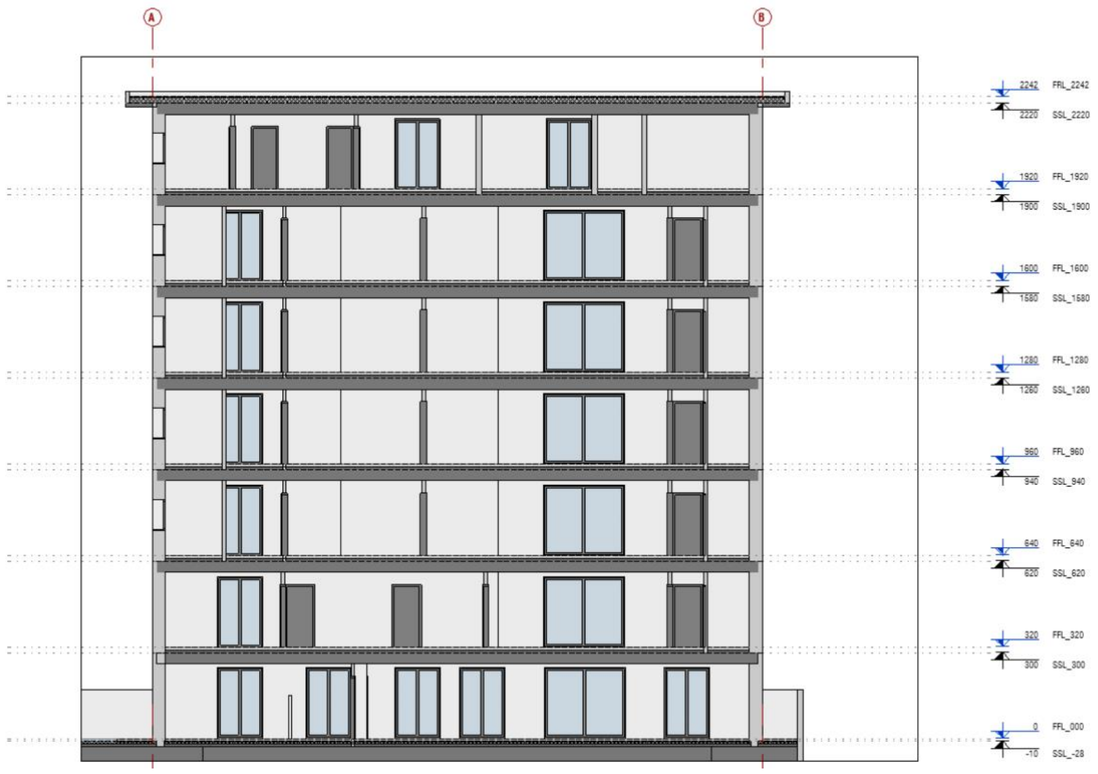


Figure 31 – Section view before running the plug-in

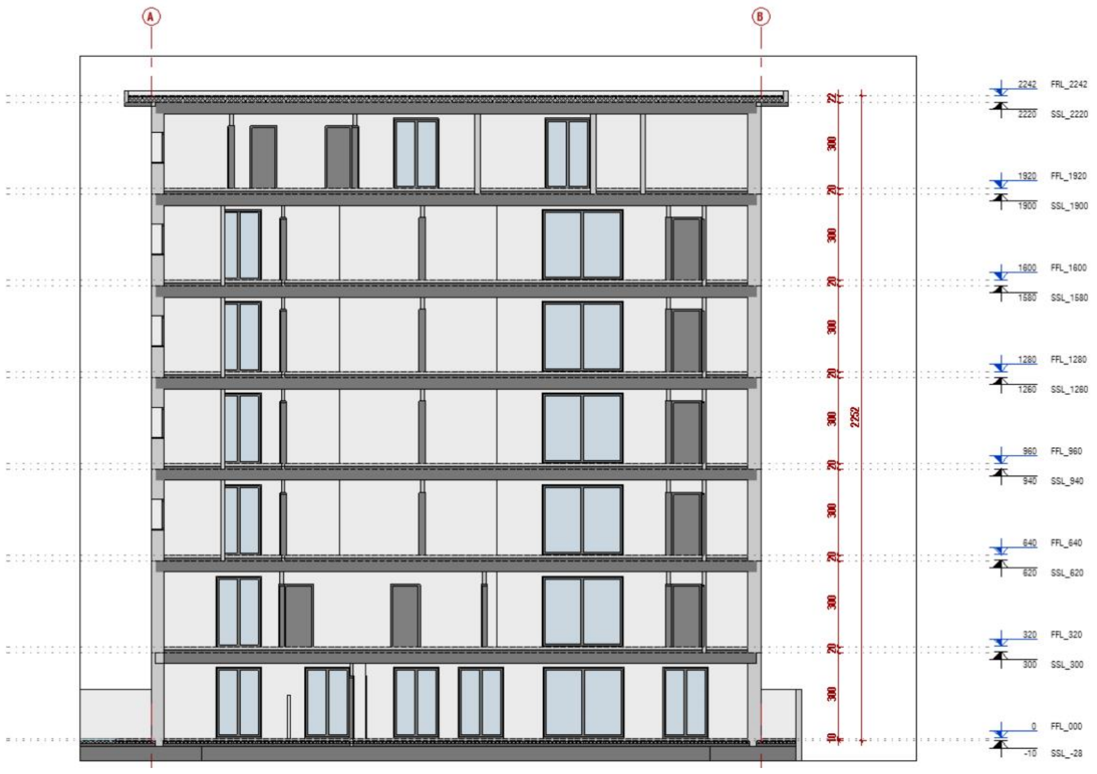
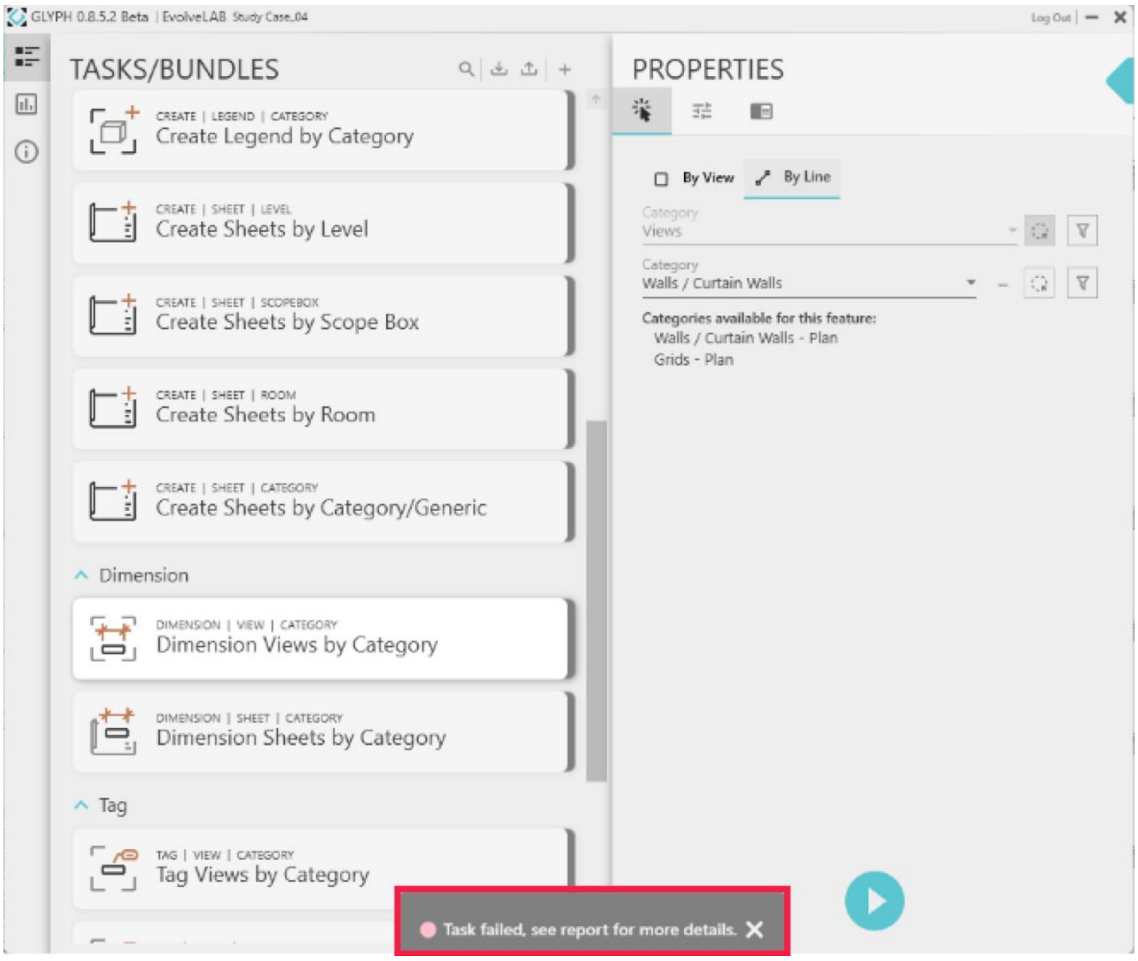


Figure 32 – Section viwe after running the plug-in to place annotations dimension in levels using By View mode

Employing the By Line method for dimensioning was not possible, even using the category Wall / Curtain Wall as a point of reference, the outcome remains unsatisfactory, as indicated by the plug-in, it shows the error message "Task failed, see report for more details" (Figure 33). Consequently, the use of lines for dimensioning within the section view becomes unviable. The fact that the error message "Task failed, see report for more details" is displayed is valuable for users because it provides insight into what went wrong. This level of feedback can be instrumental in troubleshooting and diagnosing issues, helping users understand where improvements or adjustments may be needed in their approach.


GLYPH 0.8.5.2 Beta | EvolveLAB Study Case_04

TASKS/BUNDLES

CREATE | LEGEND | CATEGORY
Create Legend by Category

CREATE | SHEET | LEVEL
Create Sheets by Level

CREATE | SHEET | SCOPEBOX
Create Sheets by Scope Box

CREATE | SHEET | ROOM
Create Sheets by Room

CREATE | SHEET | CATEGORY
Create Sheets by Category/Generic

Dimension

DIMENSION | VIEW | CATEGORY
Dimension Views by Category

DIMENSION | SHEET | CATEGORY
Dimension Sheets by Category

Tag

TAG | VIEW | CATEGORY
Tag Views by Category

PROPERTIES

By View / By Line

Category: Views

Category: Walls / Curtain Walls

Categories available for this feature:
Walls / Curtain Walls - Plan
Grids - Plan

Task failed, see report for more details.

Figure 33 - Error message in the plug-in after trying to use in a secction view with By Line mode

3.3.4. Element renumbering

Proper numbering and naming are essential for Revit project coordination, yet keeping up with design changes can be time-consuming. Mark is a field that often is used for delivering information—for example, door number mark. Revit automatically increments the mark numbering. However, sometimes we need to change the number, add prefixes/suffixes, change the numbering sequence, and others. It is a tedious sub-process, and unfortunately, Revit doesn’t have a feature for automatic numbering.

To renumber elements, use the ReOrdering (DiRoots, 2023) tool. The reordering tool makes it possible to select which parameter to put the number, not limited to the mark parameter. It's possible to renumber all elements in a view, project, or just a selection of elements as required.

Categories and individual elements can also be specified either from a line or from a crossing or vertex selection via the use of a detail line. The elements can then be assigned a starting number and prefix or suffix as appropriate.

If the automated numbering sequence doesn't suit, it is possible to export the data out to an Excel file, edit it, and then read it back into the project and populate the chosen parameters with the modified values. Figure 34 shows the function in the Revit tab of the plug-in DiRootsOne (DiRoots, 2023).

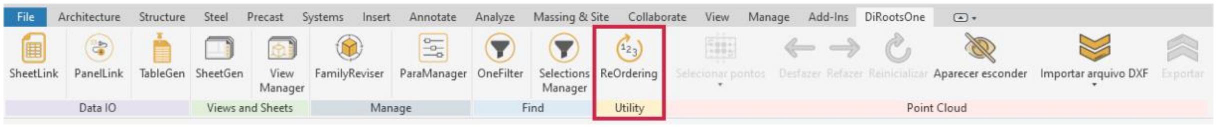
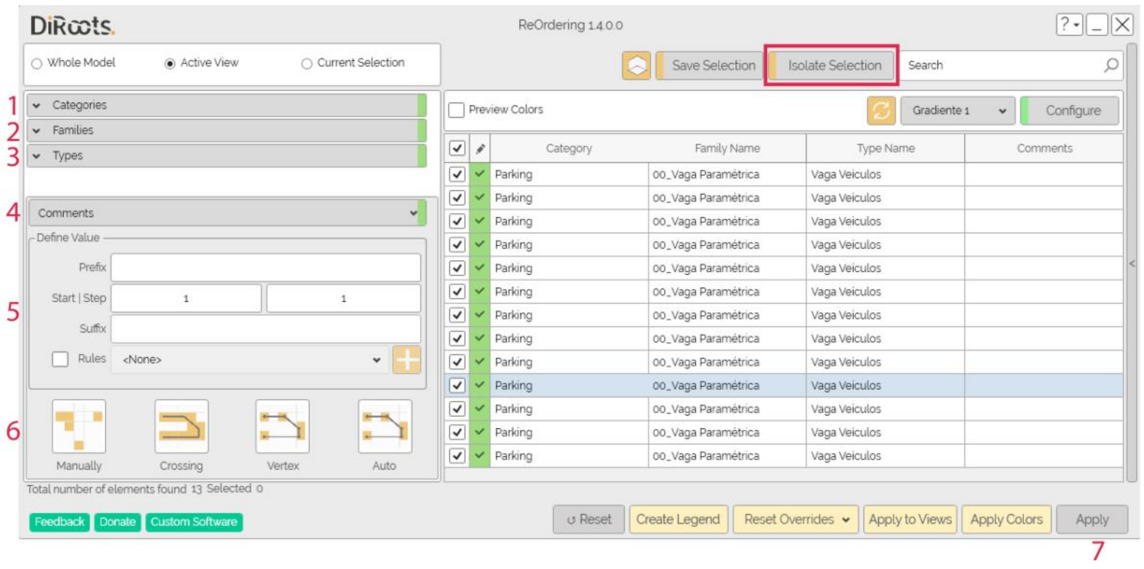


Figure 34 - DiRoots One Toolbar in Revit with the plug-in ReOrdering in highlight

The typical workflow shown in Figure 35 is to first select the elements by Categories (1), Families (2), and Types (3). With the "Isolate" (in red) button is possible to quickly visualize the selected elements. After is necessary to select an instance parameter (4) to renumber, such as Comments, Mark, Project, and others. Continuing is necessary to choose the rules to start renumbering (5) (Starting Number, Multiplier, Prefix, Suffix). After that, choose between 4 ways of renumbering: Automatically, Manually, Crossing, or Vertex (6). In the end, apply (7) to the chosen elements.



Start | Step: 1 | 1

Category	Family Name	Type Name	Comments
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	
Parking	00_Vaga Paramétrica	Vaga Veiculos	

Total number of elements found 13 Selected 0

Figure 35 – Workflow of ReOrdering

With Manually, is necessary to select one by one to update a parameter value – the changes are automatically applied when you click on the elements. When using Crossing and Vertex first is necessary to draw detailing line passing the elements that are going to be reordered. With Crossing, all elements that the detail line is passing are going to be reordered, and with Vertex, only the elements that a vertice of the line inside the element, are going to be reordered. The process to show how the ways of reordering, Automatically and Crossing works, will use as a base the parking lots incorrectly numbered as shown in 36.

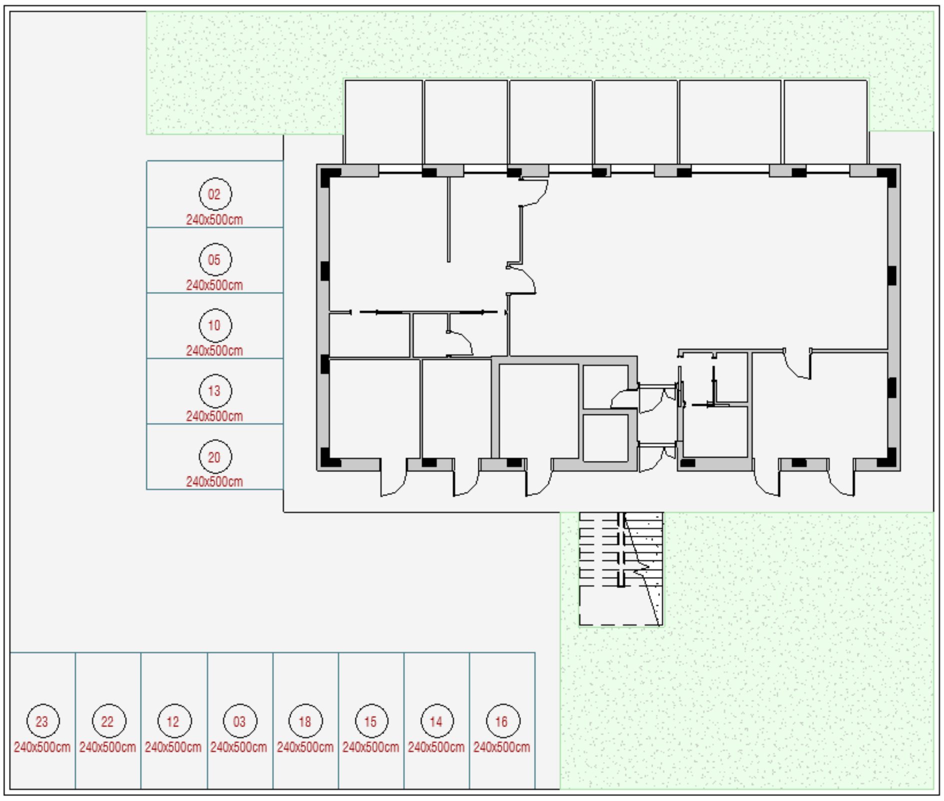


Figure 36 – Floor plan with the parking elements without ordering.

The plug-in's automated selection of numbers for the components while using the Auto function occasionally results in solutions that differ from what was expected. Figure 37 shows a floor plan with parking elements after the Auto function has been used. While the reordering process produced the desired results for items 01 to 05, and from 06 to 012, the Auto function did not result as intended since it didn't follow the standard convention of left-to-right reading orientation.

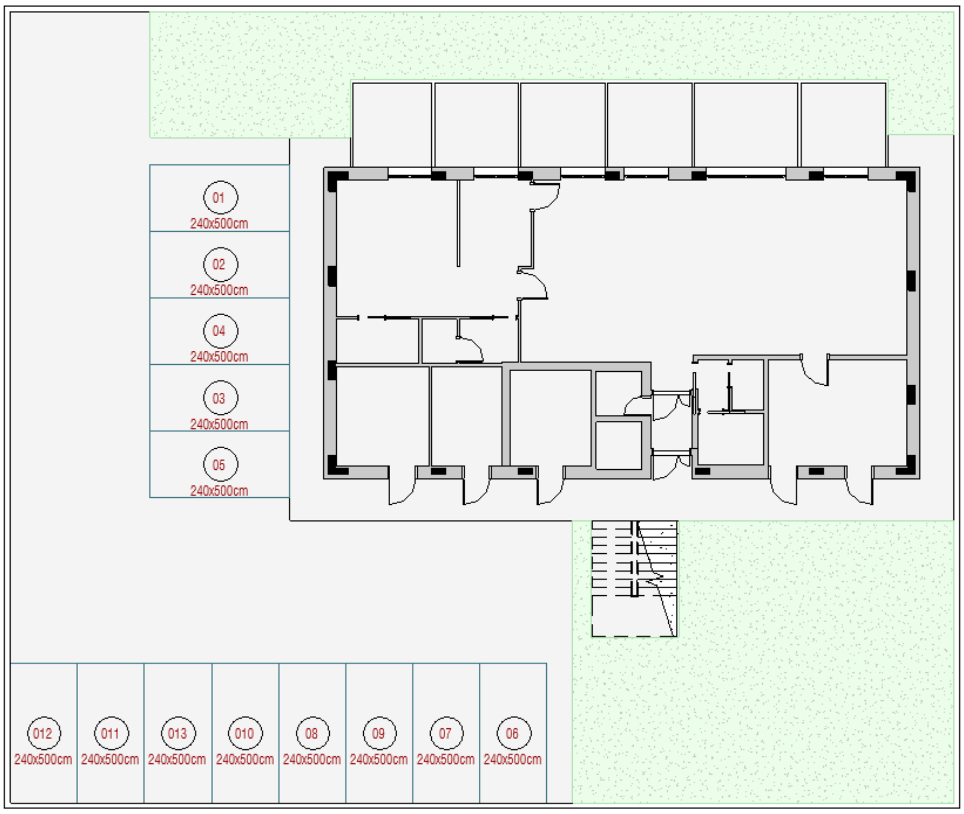


Figure 37 – Floor plan with the parking elements after applying the Auto function.

Through the application of the Crossing function, the plug-in follows the standard convention of a left-to-right and top-to-bottom reading orientation as shown in Figure 38. Remarkably, this alignment principle remains regardless of the direction in which lines were initially selected, whether ascending or descending. A recommended approach to get the expected result is therefore most effectively accomplished through the utilization of the Crossing Function.

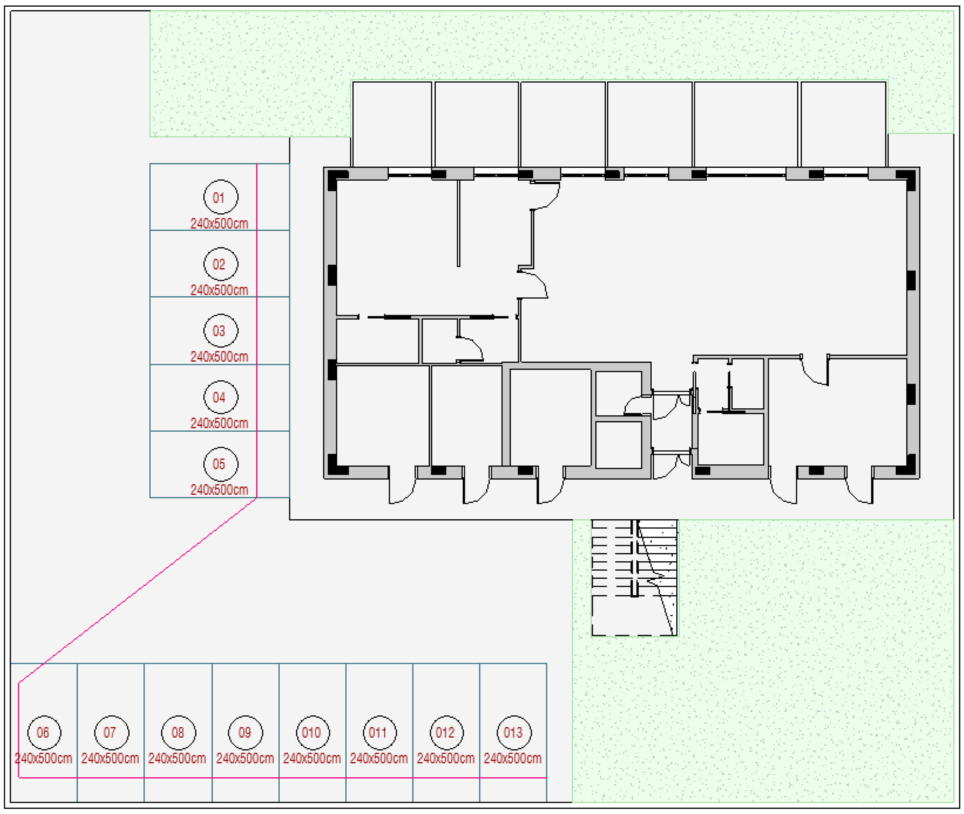


Figure 38 – Floor plan with the parking elements after applying the Crossing function

3.4. Sub-processes solved with Dynamo script

This subchapter elucidates the sub-processes that have been successfully resolved through the development of scripts in Dynamo, as explained before, they were made in Dynamo due to its connection with Revit, the BIM software used by NOZ Arquitectura. The subchapter will show the logic behind the script and how it works in a project.

3.4.1. Alignment of the view title of the viewport in a chosen position

The alignment of viewport names to a user-specified position within Revit through the utilization of Dynamo presents an approach to improve visual clarity and organization. By using the scripting capabilities of Dynamo, the sub-process of aligning viewport names can be automated and customized to follow specific design preferences. Upon executing the Dynamo script, the view names associated with each viewport are positioned as chosen, resulting in a coherent visual presentation that harmonizes with the overall design intent. This dynamic approach streamlines the process of managing view names, avoiding manual adjustments, and promoting a standardized aesthetic across the project.

The development of the script was made in two ways, as shown in Figure 42 one was getting all the viewports in the project, to align all of them, and the second was getting all the viewports from a view set. The part of the alignment was made with Python code and is the same for both cases.

The script's development occurred through two distinct approaches. Firstly, it involved gathering all viewports within the project to align them. The second involved gathering the viewports from a view set to align them. Python code was used to implement the alignment process, and this alignment method stayed the same for both circumstances. The run the script, the Python code asks for three inputs: the viewports, the offset distance from the viewport, and the position of alignment. The output is the list of the viewports that were aligned (Figure 39). In order to organize the script to be available in Nonica, the plug-in that makes scripts available in an easy way, the position of the alignment was set as a changeable input in both scripts, letting the user decide the desired position of the view title, and in the script that only aligns the view name of the viewports from a view set, the selection of the view set is also an input that can be chosen.

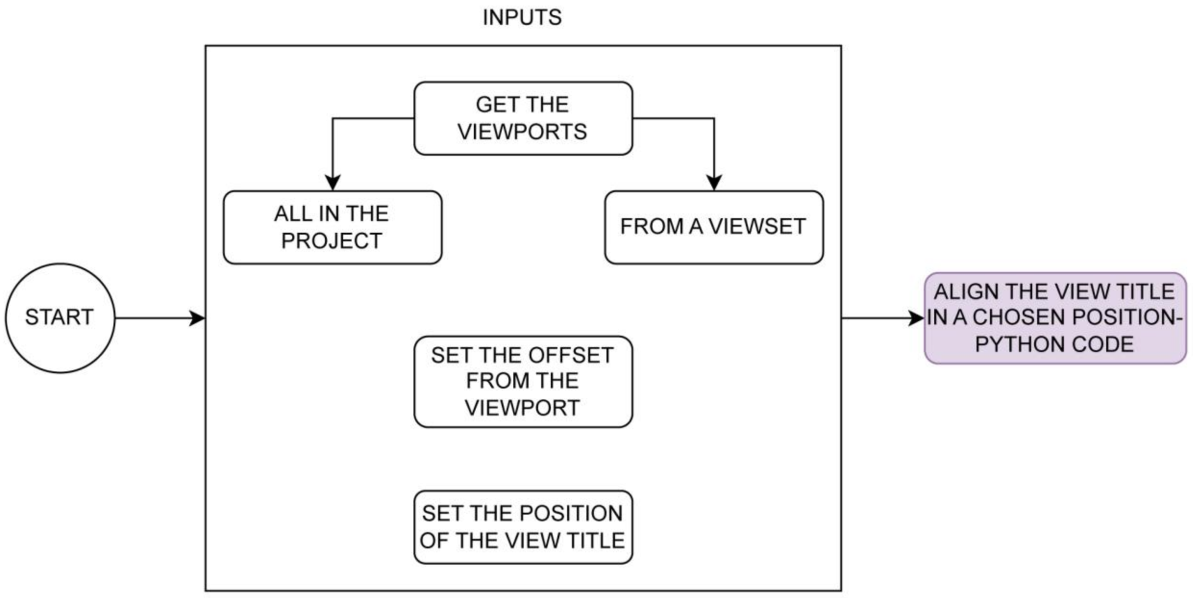

INPUTS
GET THE VIEWPORTS
ALL IN THE PROJECT
FROM A VIEWSET
START
SET THE OFFSET FROM THE VIEWPORT
SET THE POSITION OF THE VIEW TITLE
ALIGN THE VIEW TITLE IN A CHOSEN POSITION- PYTHON CODE

Figure 39 – Process Map of the development of the script to align the view title in chosen position

Figure 40 shows an example of two viewports in a sheet before running the script. The line of their view title is not aligned to the name. Figure 41 shows both viewports after running the script, the line of the view name also changes to fit the name. This approach streamlines the process of managing view names, avoiding manual adjustments, and promoting a standardized aesthetic across the project.

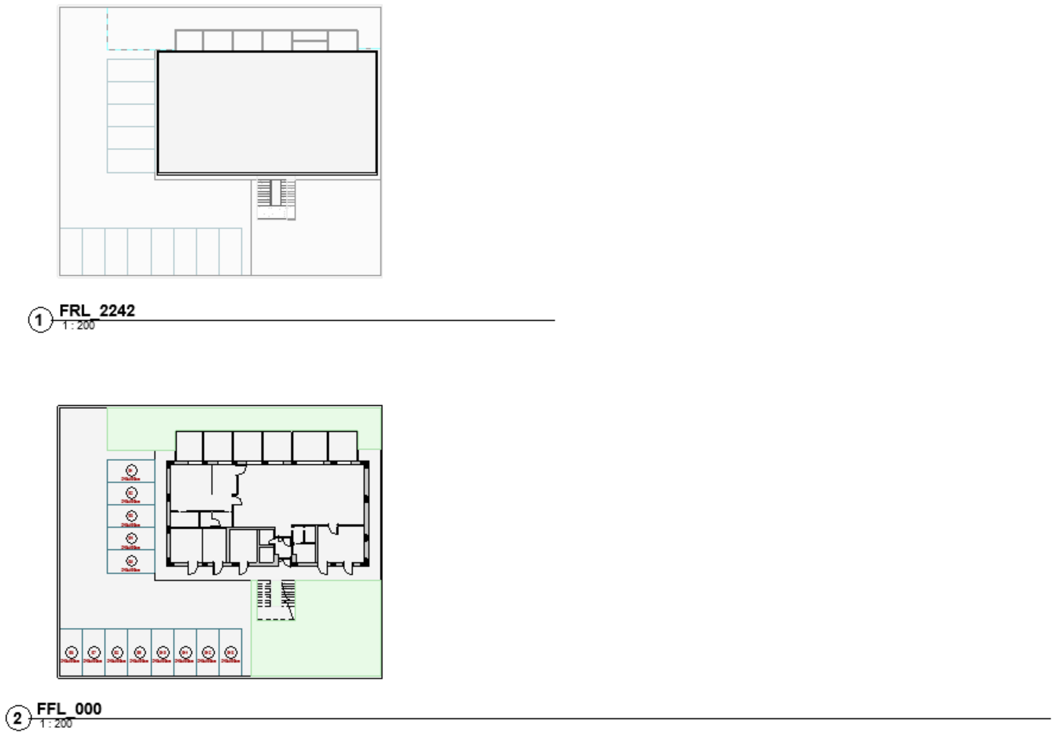


Figure 40 – Viewsports in a sheet before running the script

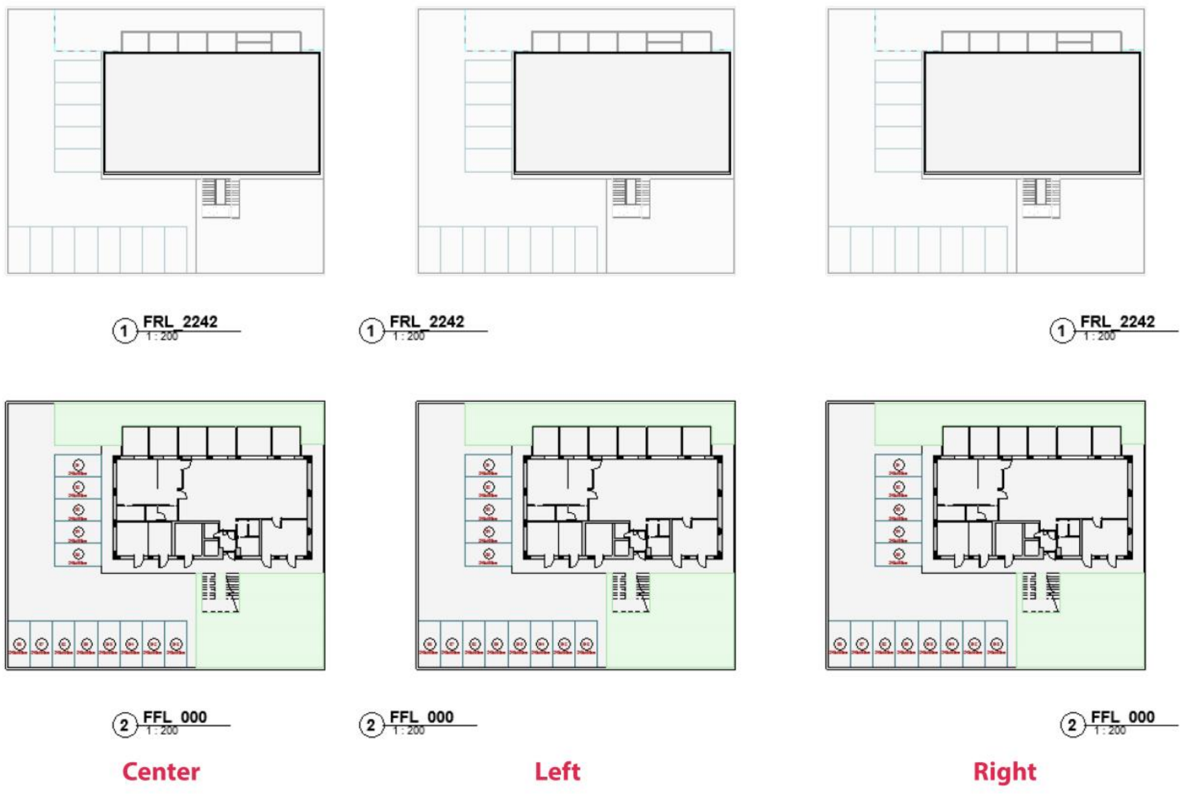


Figure 41 – Same viewports of Figure 41 in the sheets after running the script with the indicated position

3.4.2. Alignment of the view title with the viewport

Continuing the discussion on aligning view names, a different alignment method involves making use of the viewport as a reference. In this case, the drawing's bounds are really marked by an extension of the view name line to fit the viewport's size.

Similar to the approach used in the script from the preceding subsection, this script was also devised in two variations. The first approach is aligning all viewports within the project, while the second involves aligning viewports sourced from a view set. Within this script, a node from a package handles the alignment process, diverging from previous scripts focused on aligning view text to specific positions, that a Python code does the process. In this case, the node originates from the Rhythm package, and it only requires as input, the viewports that will be aligned (Figure 42). Figure 43 shows the view titles misaligned from the viewports and Figure 44 shows the same view titles after running the script, aligned to the viewport.

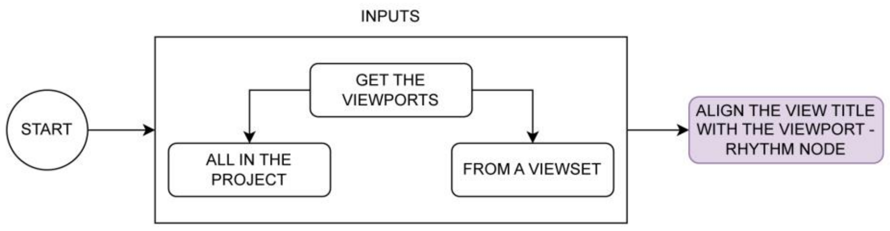


Figure 42 – Process Map of the development of the script to align the view title with the viewport

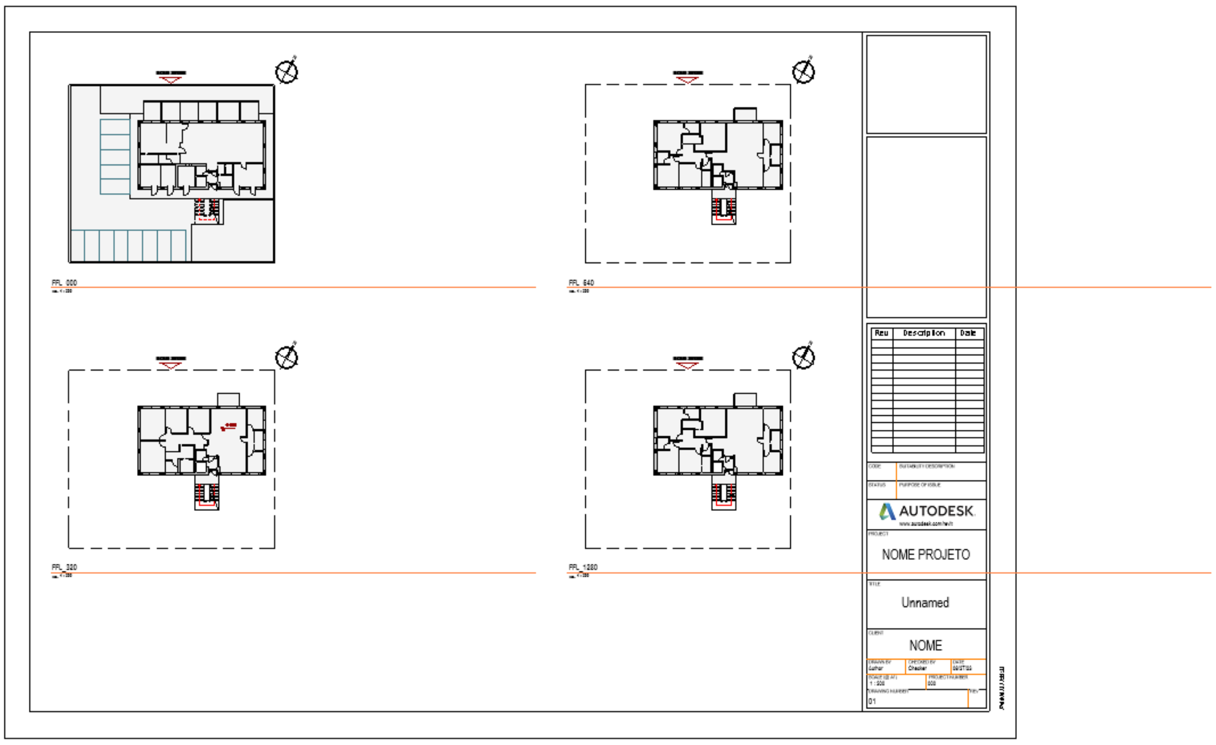


Figure 43 – Viewports in a sheet before running the script

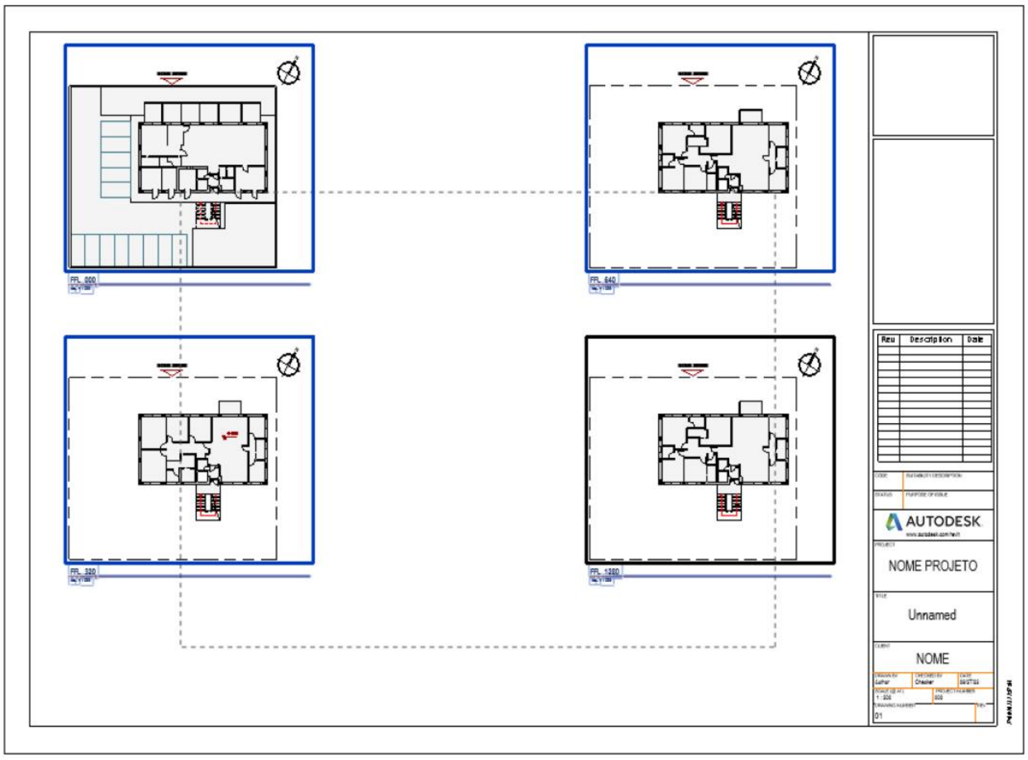


Figure 44 – Same viewports of Figure 43 in the sheets after running the script

3.4.3. Detection of overlapping information

Before delivering the drawings, an essential step in the documentation process is conducting a comprehensive quality check to ensure that all the necessary information is placed on it, legible, and accurately presented without conflicting elements. This review process is essential to maintaining the integrity, accuracy, and professionalism of the documentation.

The clash detection process is a crucial aspect of BIM that helps to identify possible mistakes, conflicts, and clashes between BIM models. However, it is important to note that clash detection only checks the geometric information and to check if the non-geometric information, annotations, dimensions, tags, and notes, are clashing is necessary to do it manually, using visual inspection.

Within this context, a script was developed with an emphasis on the overlapping of non-geometric information in the drawings, encompassing floor plans, sections, and elevations. However, it's worth noting that the script is versatile in its application, extending beyond its primary purpose to encompass the ability to assess both geometric and non-geometric information for potential clashes. This dual functionality empowers the script to comprehensively examine instances of both spatial and annotation-related conflicts, fostering a comprehensive quality evaluation of the documentation.

The script works in the current view opened in Revit, by collecting all the elements (geometric and non-geometric) on it. From that, it asks the user to select 2 categories of elements for checking if they are overlapping and after processing, it highlights by override with color the ones that are overlapping on the view (Figure 45).

The script, which operates within the active view in Revit, interacts by collecting all elements, regardless of their nature—be they geometric or non-geometric—that are in the current view. It then asks the user to choose two specific categories of elements, the ones that are in the current view, that can be different or the same, to undergo an assessment for potential overlaps. Upon this user input, the script undertakes a processing phase.

Upon completion of the assessment, the script employs a visual enhancement technique: it puts a selective array of elements that exhibit overlaps within the view with vibrant color overrides. This use of color serves as an indicator of the detected clashes, allowing them to be instantaneously identified and addressed.

In essence, this script serves as a dynamic tool, capable of efficiently isolating and showcasing instances of overlap between chosen element categories. It functions within the context of the active view in Revit, ensuring that potential conflicts are not only identified but also highlighted with visual clarity, ultimately contributing to a refined and conflict-free documentation process.

A complementary script was also developed to facilitate the effortless removal of color from elements that were identified as overlapping. This script serves the purpose of simplifying the procedure, ensuring that reverting the visual enhancements applied to the elements is a straightforward and streamlined sub-process.

Figure 46 shows the floor plan before applying the script, and Figure 47 shows after running to check if there is any overlapping between Room Tags and dimensions annotation.

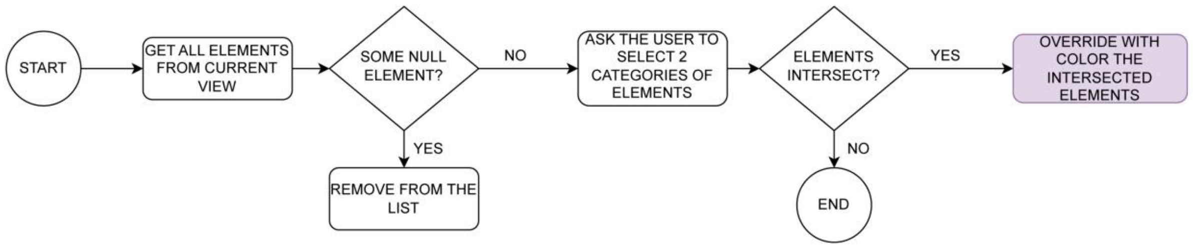

START
GET ALL ELEMENTS FROM CURRENT VIEW
SOME NULL ELEMENT?
NO
YES
REMOVE FROM THE LIST
ASK THE USER TO SELECT 2 CATEGORIES OF ELEMENTS
ELEMENTS INTERSECT?
YES
NO
OVERRIDE WITH COLOR THE INTERSECTED ELEMENTS
END

Figure 45 - Process Map of the development of the script to check for overlapping elements

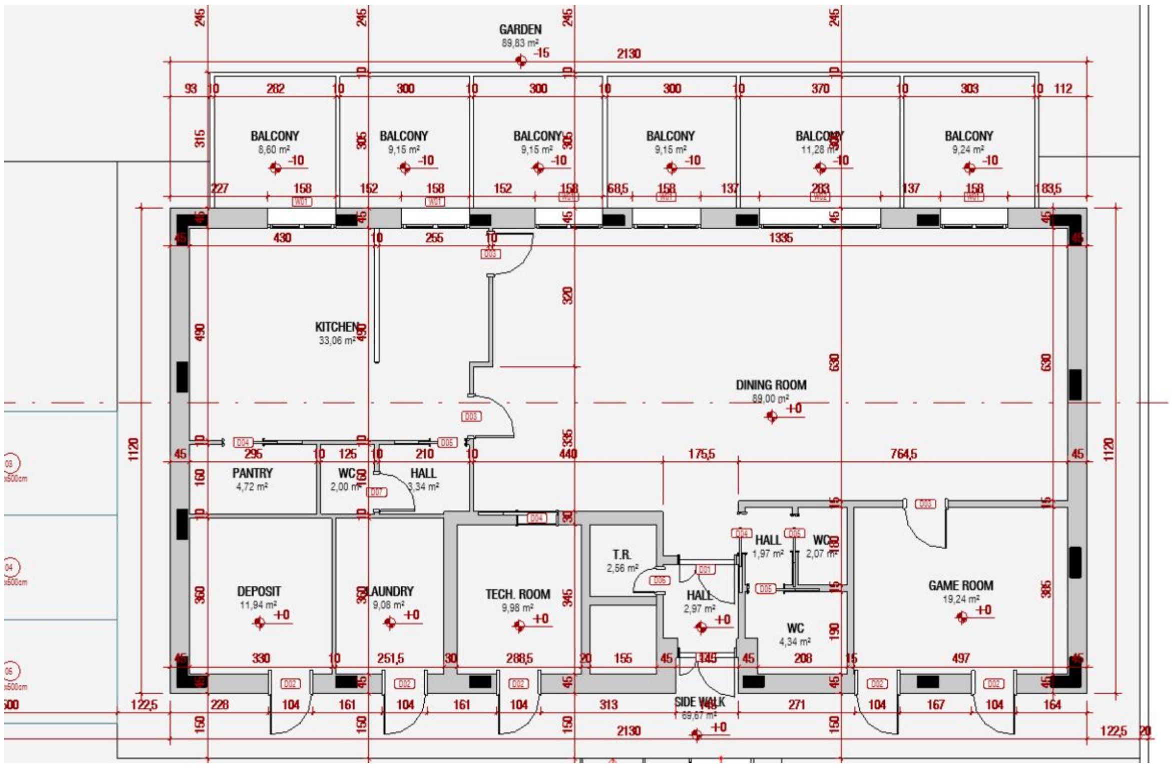


Figure 46 – Floor plan with annotations overlapping before running the script

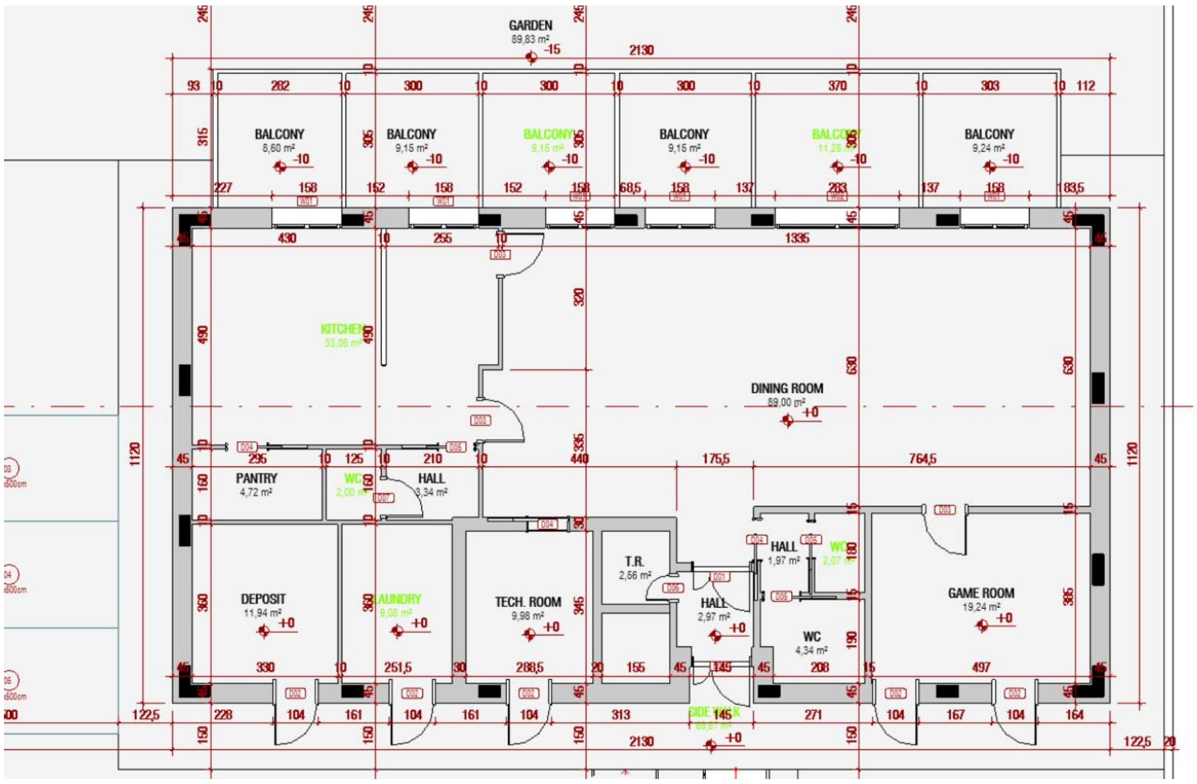
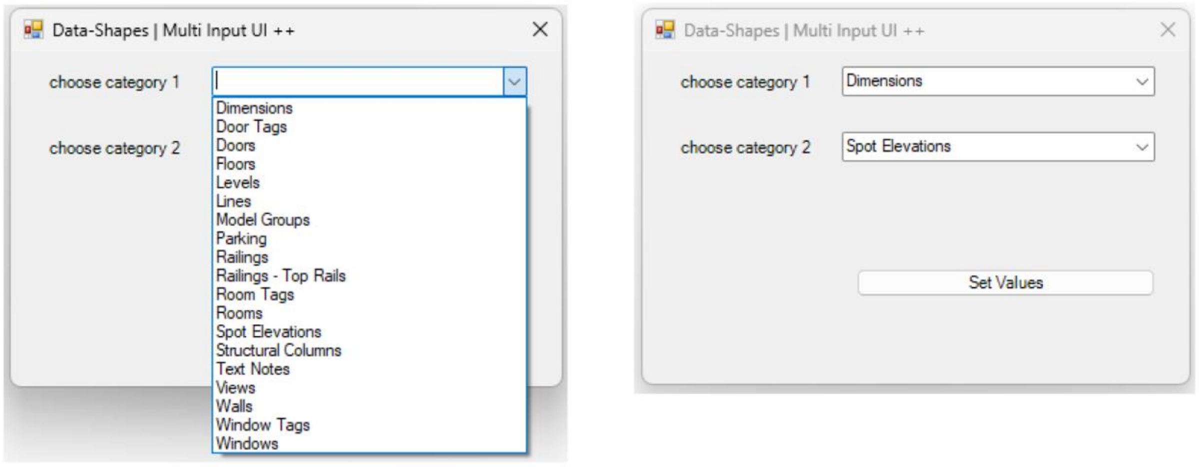


Figure 47 – Floor plan after running the script checking for overlapping between Room tags and dimensions annotations

Figure 48 shows the pop-up tab that the script opens to the user select the categories that wants to check the overlapping, in this case is checking for Dimensions and Spot Elevations, that is shown on Figure 49 before running the script and Figure 50 after.



Data-Shapes | Multi Input UI ++

choose category 1: Dimensions

choose category 2: Spot Elevations

Figure 48 – Pop-up Tab from the script to select the categories to check for overlapping

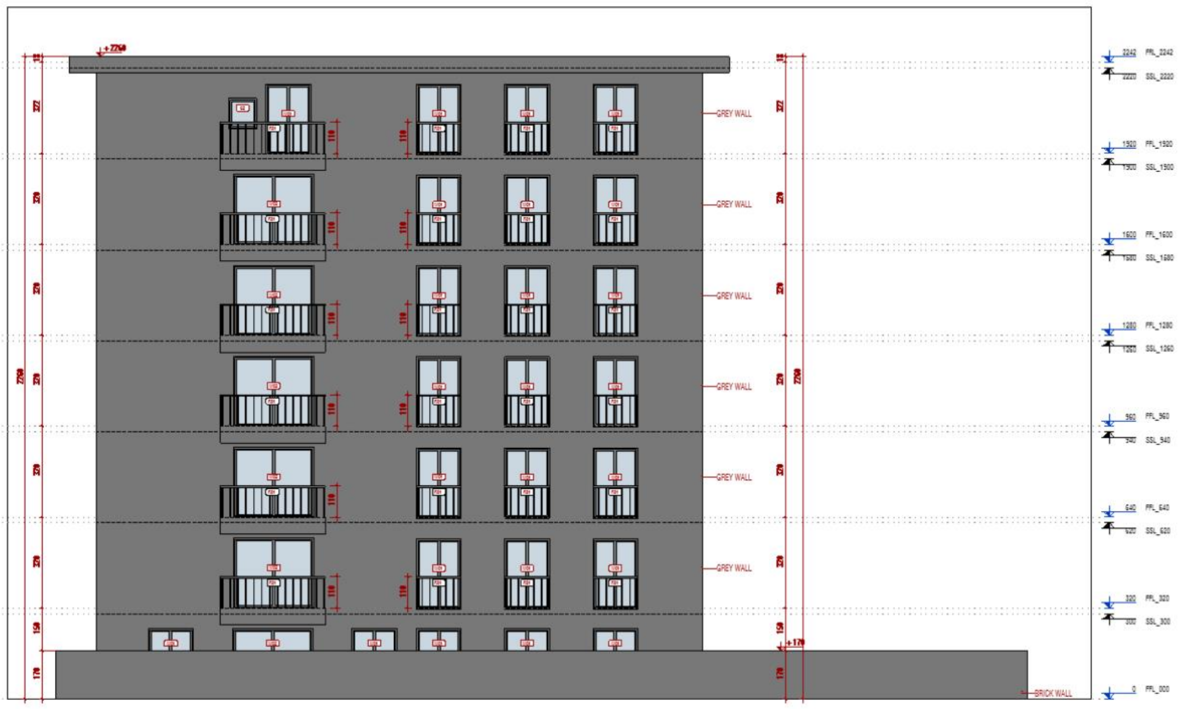


Figure 49 – Elevation view before running the script

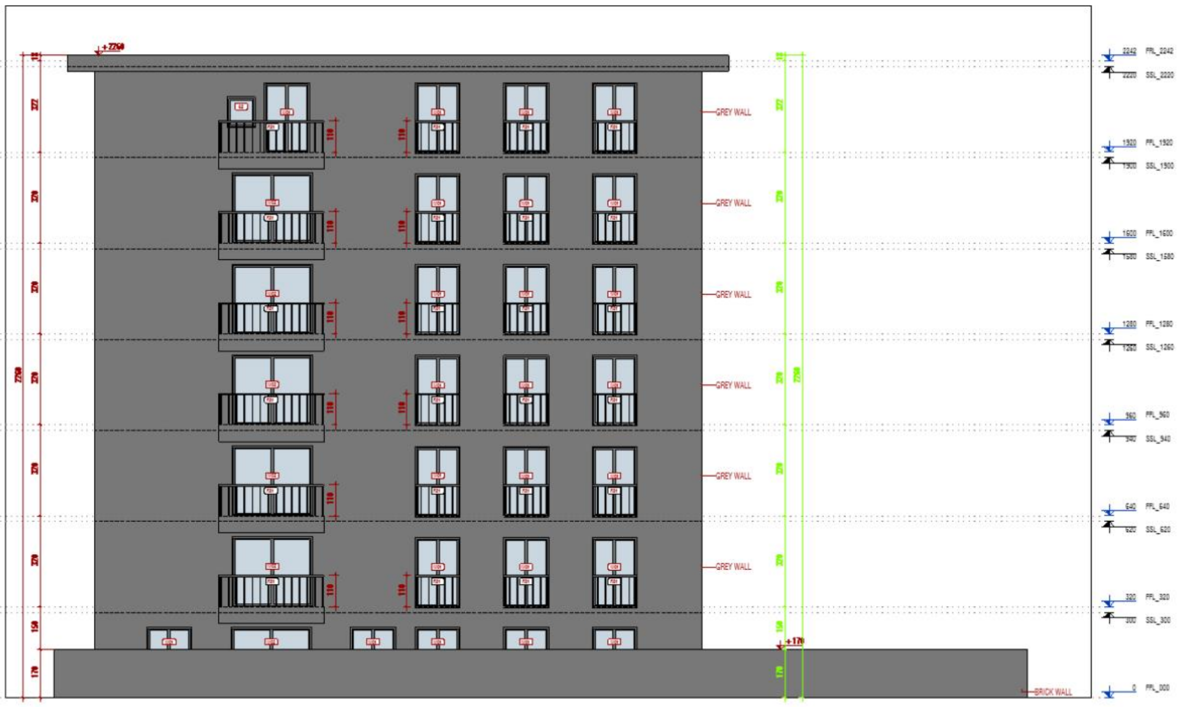


Figure 50 – Elevation view after running the script checking for overlapping between dimensions annotations and spot elevations

Figure 51 presents a sectional view prior to script execution, while Figure 52 reveals the outcome subsequent to detecting overlaps between Room Tags and Spot Elevations. Furthermore, Figure 53 demonstrates the script's capacity to identify clashes among three-dimensional components, exemplified by the Floors and Walls in this specific scenario.

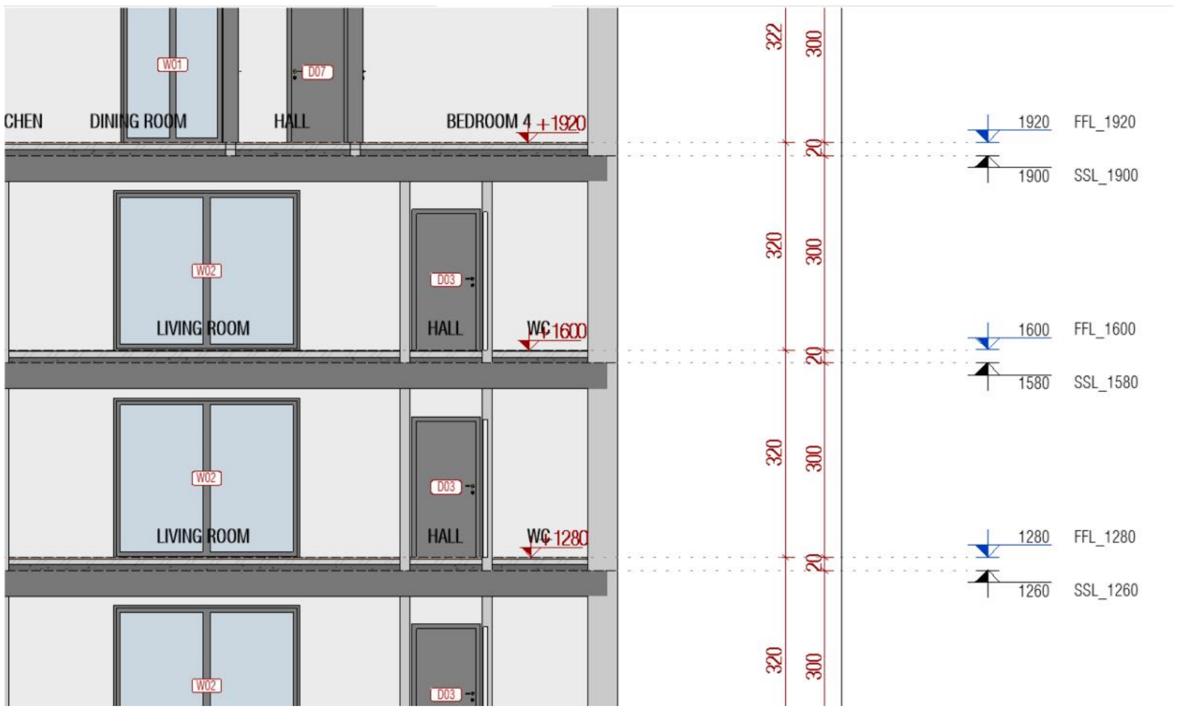


Figure 51 – Section view before running the script

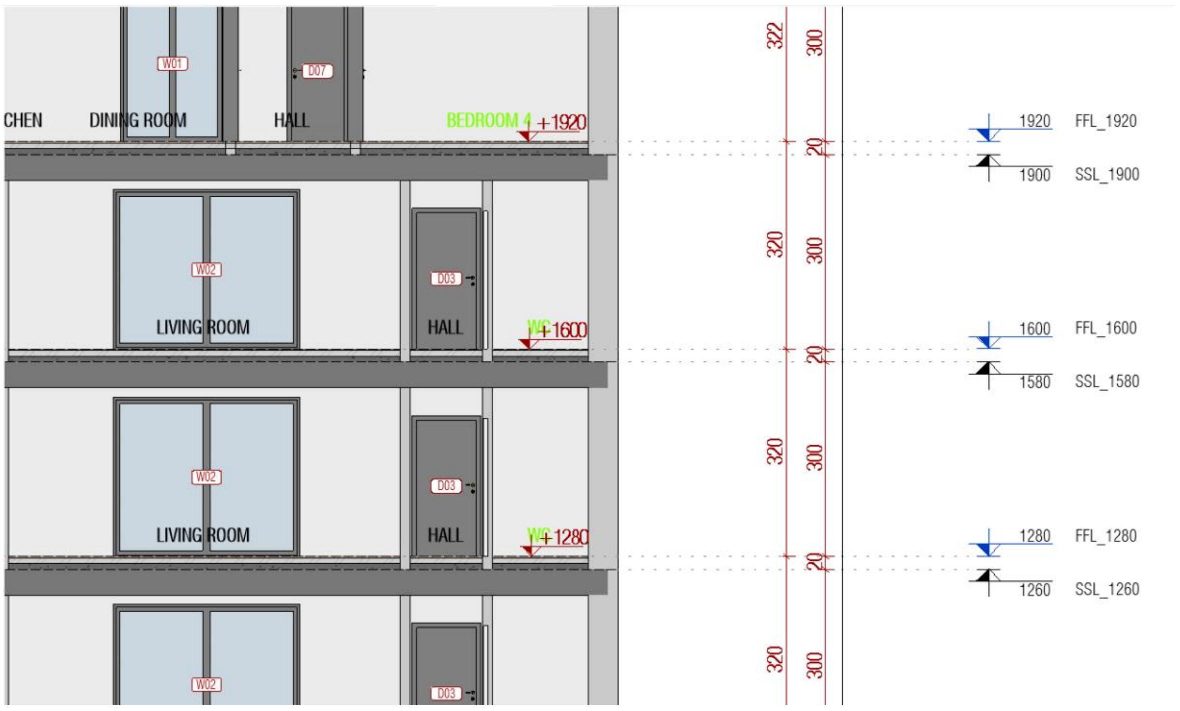


Figure 52- Section view after running the script checking for overlapping between Room tags and spot elevations

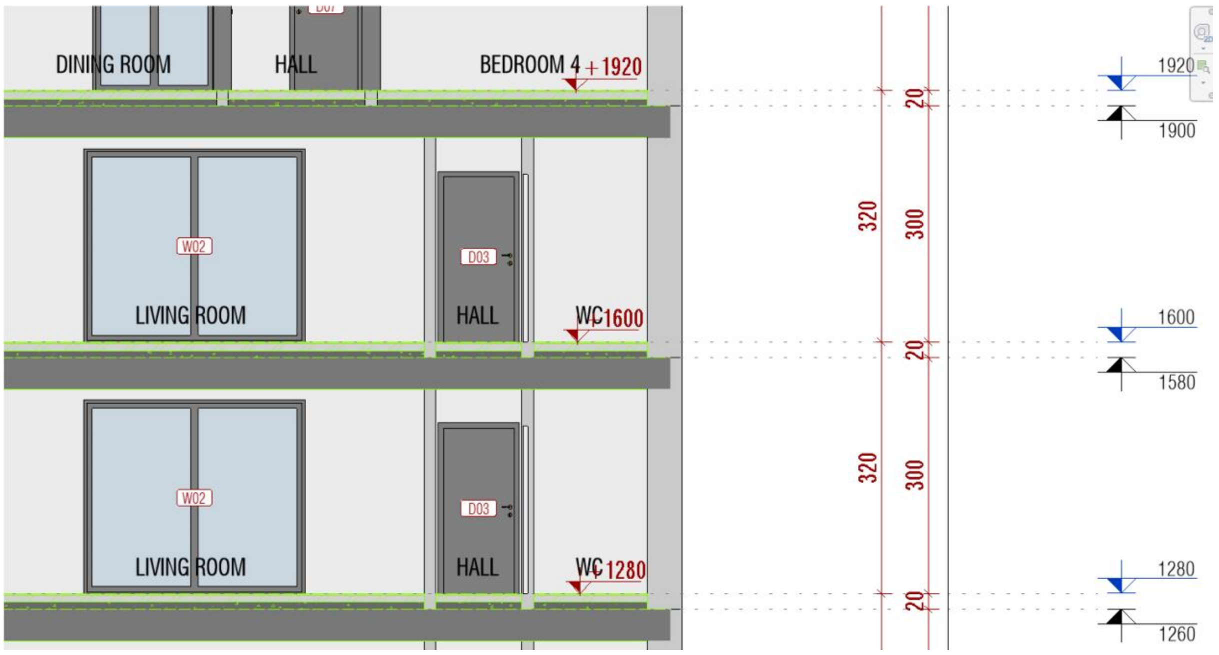


Figure 53 - Section view after running the script checking for clashing between floor and walls

3.4.4. Centralization of Rooms, Areas, Revit Spaces, and their respective tags

Within Revit, three distinct methods exist for spatializing a place: rooms, areas, and Revit spaces. These elements have a unique purpose within a project. While they possess distinct functions, a common characteristic is their demarcated boundaries and upon placement, each of these elements is accompanied by an area (with a unit) and a corresponding tag.

A room is a subdivision of space within a building model, based on elements such as walls, floors, roofs, and ceilings. These elements are defined as room-bounding. Revit refers to these room-bounding elements when computing the perimeter, area, and volume of a room. You can also use room separation lines to further subdivide space where no room-bounding elements exist (Autodesk, 2023). Typically, we refer to building areas as rooms, such as bedrooms, living rooms, hallways, bathrooms, offices, balconies, and more. A room is a 3D entity with borders in the floor plan and a height that may be used to compute its volume.

An area is a subdivision of space within a building model, typically on a larger scale than individual rooms. Model components are not always used as boundaries for areas. You may choose model components to use as borders or draw area limits. These areas may be used to calculate the areas of the floor, walls, windows, and other things. They are useful for quantification, budgeting, and compliance with building codes and regulations (Autodesk, 2023). The types of areas in a structure are often defined by regions, such as interior and exterior areas, private and public areas, moist and dry areas, and more. The only sort of floor plan in which areas may be produced is the area plan and they are a 2D element.

Revit Spaces depict how a room is used practically. They are used to specify a room's ventilation and temperature characteristics. Room-specific information, including thermal load, needed air volume, and other characteristics relating to comfort and environmental performance, may be found in spaces that can be connected to particular rooms (Autodesk, 2023). This is crucial for energy simulations and HVAC (Heating, Ventilation, and Air Conditioning) systems. Spaces can be made on all types of floor plans, and they are a 3D element because the volume is important data for the analysis.

For accuracy and visual clarity in the documentation of a project, it is important to centralize rooms, Revit spaces, areas, and their related tags. The users may make sure that the rooms, spaces, and regions, along with their corresponding tags, are uniformly positioned inside the design by using a script that can centralize them (Figure 54). This facilitates a unified visual representation and reduces design discrepancies. Additionally, it speeds up the design process by requiring less manual effort to relocate items and associated tags and considerably improves documentation quality by minimizing differences caused by hand placement. Is important to mention that the centralization of these elements is not possible to do it manually with precision. Placing them manually, to find the center is just an estimation because there is no reference point to put them. Figure 55, Figure 56, and Figure 57, show the ground floor plan with each one of the spaces, Room, Area, and Revit Space, in the respective order, and is possible to see that in all of them, the center of it and their tags are not centralized, before running the script.

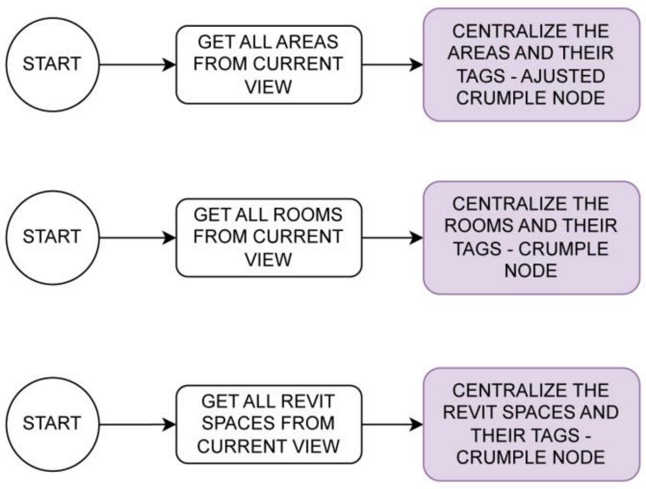

START
GET ALL AREAS FROM CURRENT VIEW
CENTRALIZE THE AREAS AND THEIR TAGS - AJUSTED CRUMPLE NODE
START
GET ALL ROOMS FROM CURRENT VIEW
CENTRALIZE THE ROOMS AND THEIR TAGS - CRUMPLE NODE
START
GET ALL REVIT SPACES FROM CURRENT VIEW
CENTRALIZE THE REVIT SPACES AND THEIR TAGS - CRUMPLE NODE

Figure 54 – Process Map of the development of the script to centralize Rooms, Areas and Revit Spaces

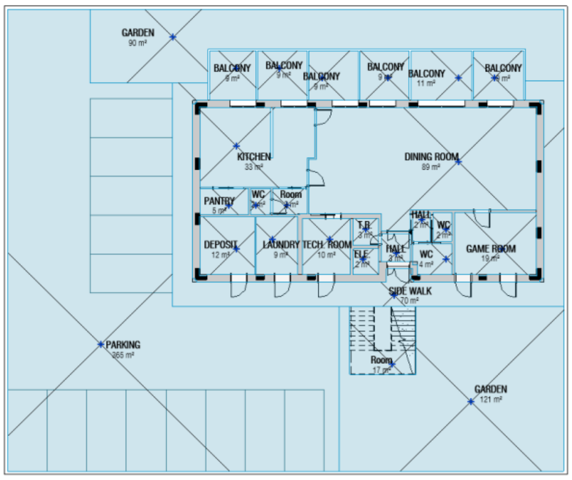


Figure 55 – Floor plan with Rooms in highlight before running the script

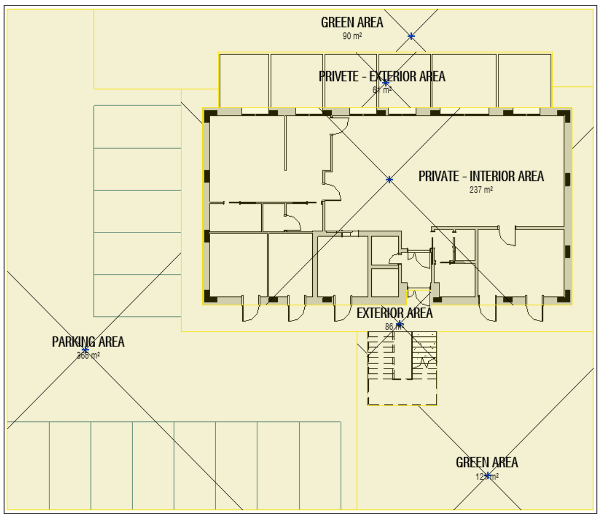


Figure 56 – Area plan with Areas in highlight before running the script

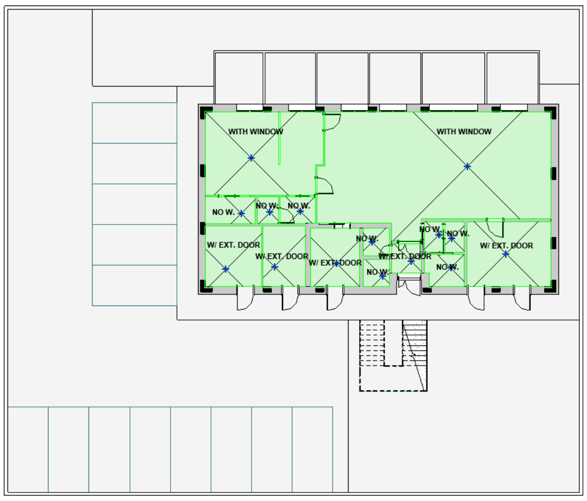
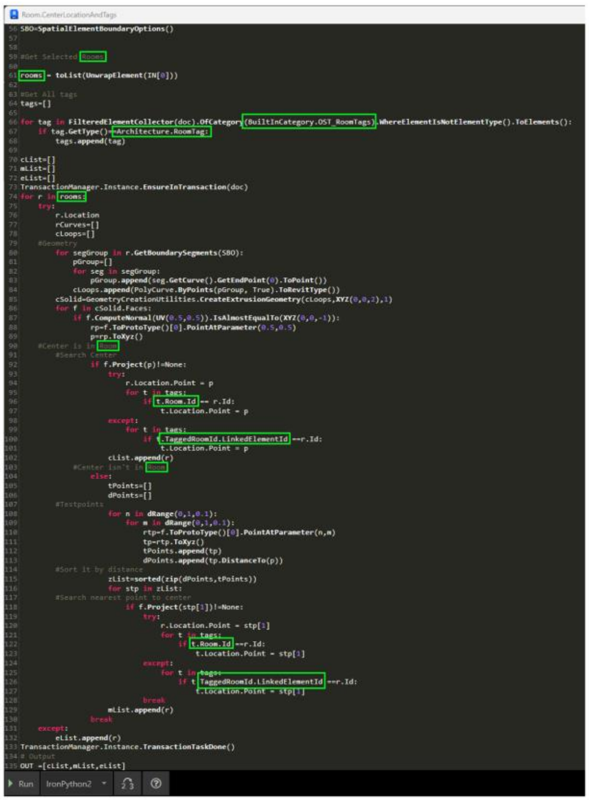


Figure 57 – Floor plan with Revit Spaces in highlight before running the script

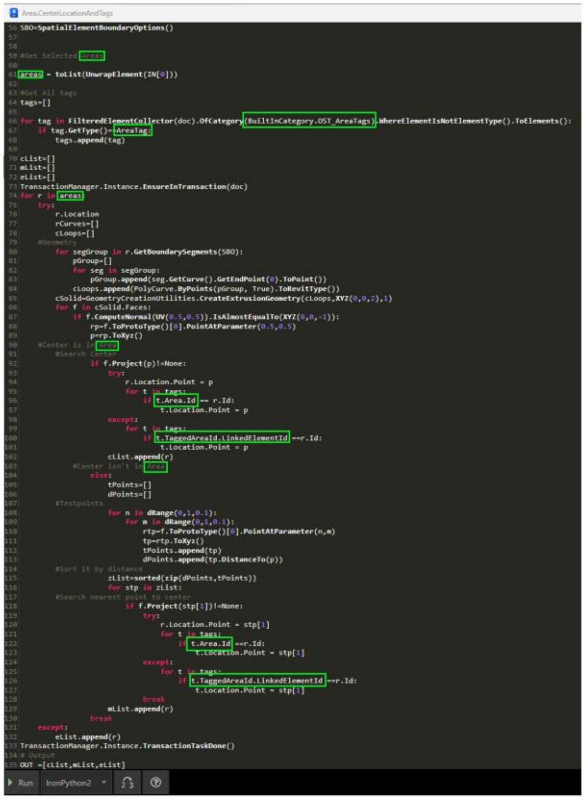
The initial script, aimed at centralizing these spaces, overlooked the variations in spatial configurations such as L-shapes or U-shapes. It failed to account for the intrinsic requirement of factoring in the center of such shapes, often leading to misplacement outside their boundaries.

After realizing this mistake, the development of the script to place the Room and the Revit Space in the center was easy to do since there is a node from the Modelical package that does this sub-process and asks as input the Rooms and Revit Spaces that are going to be centralized and then, automatically centralizes them and their tags. For the areas wasn't founded any node from any package that does this sub-process, but the nodes from Modelical that centralizes the Rooms and the Revit Spaces are custom node, and then it was possible to modify one of them, in the case the Room node, to be specific for areas. To modify it was necessary to change all the places that the code says room to Area, and also change the Python code.

The first attempt was to change all the places in the code that say Room to Area, but the node didn't work, because the Python code uses the Revit API to change from Room to Area wasn't that way. It was necessary to search in the documentation of Revit API what is the name Area in the Revit API, to change for the appropriate meaning. Figure 58 shows the Python code of both nodes and the places that were necessary to change.


Room.CenterLocationAndTags

```python
SBO=SpatialElementBoundaryOptions()


#Get Selected Rooms

rooms = toList(UnwrapElement(IN[0]))

#Get All tags
tags=[]

for tag in FilteredElementCollector(doc).OfCategory(BuiltInCategory.OST_RoomTags).WhereElementIsNotElementType().ToElements():
    if tag.GetType()==Architecture.RoomTag:
        tags.append(tag)

cList=[]
mList=[]
eList=[]
TransactionManager.Instance.EnsureInTransaction(doc)
for r in rooms:
    try:
        r.Location
        rCurves=[]
        cLoops=[]
    #Geometry
        for segGroup in r.GetBoundarySegments(SBO):
            pGroup=[]
            for seg in segGroup:
                pGroup.append(seg.GetCurve().GetEndPoint(0).ToPoint())
            cLoops.append(PolyCurve.ByPoints(pGroup, True).ToRevitType())
        cSolid=GeometryCreationUtilities.CreateExtrusionGeometry(cLoops,XYZ(0,0,2),1)
        for f in cSolid.Faces:
            if f.ComputeNormal(UV(0.5,0.5)).IsAlmostEqualTo(XYZ(0,0,-1)):
                rp=f.ToProtoType()[0].PointAtParameter(0.5,0.5)
                p=rp.ToXyz()
    #Center is in Room
        #Search Center
                if f.Project(p)!=None:
                    try:
                        r.Location.Point = p
                        for t in tags:
                            if t.Room.Id == r.Id:
                                t.Location.Point = p
                    except:
                        for t in tags:
                            if t.TaggedRoomId.LinkedElementId ==r.Id:
                                t.Location.Point = p
                    cList.append(r)
            #Center isn't in Room
                else:
                    tPoints=[]
                    dPoints=[]
        #Testpoints
                    for n in dRange(0,1,0.1):
                        for m in dRange(0,1,0.1):
                            rtp=f.ToProtoType()[0].PointAtParameter(n,m)
                            tp=rtp.ToXyz()
                            tPoints.append(tp)
                            dPoints.append(tp.DistanceTo(p))
        #Sort it by distance
                    zList=sorted(zip(dPoints,tPoints))
                    for stp in zList:
        #Search nearest point to center
                        if f.Project(stp[1])!=None:
                            try:
                                r.Location.Point = stp[1]
                                for t in tags:
                                    if t.Room.Id ==r.Id:
                                        t.Location.Point = stp[1]
                            except:
                                for t in tags:
                                    if t.TaggedRoomId.LinkedElementId ==r.Id:
                                        t.Location.Point = stp[1]
                            break
                    mList.append(r)
                break
    except:
        eList.append(r)
TransactionManager.Instance.TransactionTaskDone()
# Output
OUT =[cList,mList,eList]
```


Area.CenterLocationAndTags

```python
SBO=SpatialElementBoundaryOptions()


#Get Selected Areas

areas = toList(UnwrapElement(IN[0]))

#Get All tags
tags=[]

for tag in FilteredElementCollector(doc).OfCategory(BuiltInCategory.OST_AreaTags).WhereElementIsNotElementType().ToElements():
    if tag.GetType()==AreaTag:
        tags.append(tag)

cList=[]
mList=[]
eList=[]
TransactionManager.Instance.EnsureInTransaction(doc)
for r in areas:
    try:
        r.Location
        rCurves=[]
        cLoops=[]
    #Geometry
        for segGroup in r.GetBoundarySegments(SBO):
            pGroup=[]
            for seg in segGroup:
                pGroup.append(seg.GetCurve().GetEndPoint(0).ToPoint())
            cLoops.append(PolyCurve.ByPoints(pGroup, True).ToRevitType())
        cSolid=GeometryCreationUtilities.CreateExtrusionGeometry(cLoops,XYZ(0,0,2),1)
        for f in cSolid.Faces:
            if f.ComputeNormal(UV(0.5,0.5)).IsAlmostEqualTo(XYZ(0,0,-1)):
                rp=f.ToProtoType()[0].PointAtParameter(0.5,0.5)
                p=rp.ToXyz()
    #Center is in Area
        #Search Center
                if f.Project(p)!=None:
                    try:
                        r.Location.Point = p
                        for t in tags:
                            if t.Area.Id == r.Id:
                                t.Location.Point = p
                    except:
                        for t in tags:
                            if t.TaggedAreaId.LinkedElementId ==r.Id:
                                t.Location.Point = p
                    cList.append(r)
            #Center isn't in Area
                else:
                    tPoints=[]
                    dPoints=[]
        #Testpoints
                    for n in dRange(0,1,0.1):
                        for m in dRange(0,1,0.1):
                            rtp=f.ToProtoType()[0].PointAtParameter(n,m)
                            tp=rtp.ToXyz()
                            tPoints.append(tp)
                            dPoints.append(tp.DistanceTo(p))
        #Sort it by distance
                    zList=sorted(zip(dPoints,tPoints))
                    for stp in zList:
        #Search nearest point to center
                        if f.Project(stp[1])!=None:
                            try:
                                r.Location.Point = stp[1]
                                for t in tags:
                                    if t.Area.Id ==r.Id:
                                        t.Location.Point = stp[1]
                            except:
                                for t in tags:
                                    if t.TaggedAreaId.LinkedElementId ==r.Id:
                                        t.Location.Point = stp[1]
                            break
                    mList.append(r)
                break
    except:
        eList.append(r)
TransactionManager.Instance.TransactionTaskDone()
# Output
OUT =[cList,mList,eList]
```

Figure 58 – Python script with ajustments made using Revit API – small version - large version on Appendix 2.

The Figure 59, Figure 60 and Figure 61, shows the ground floor plan with each one of the spaces, Room, Area and Revit Space, after running the script. Is possible to see that sometimes the tag doesn't stay in a convenient place for the user, so is necessary to move manually.

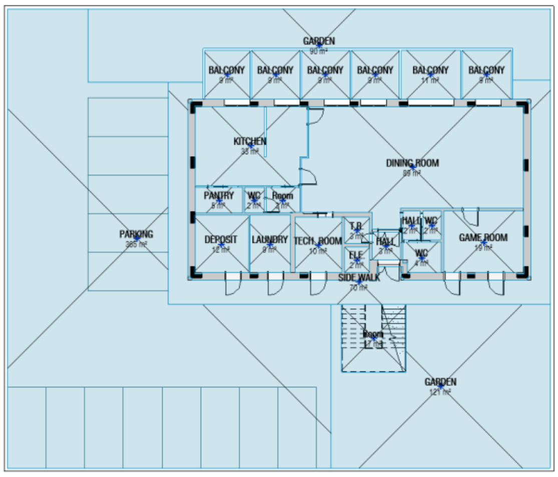


Figure 59 – Floor plan with Rooms in highlight after running the script

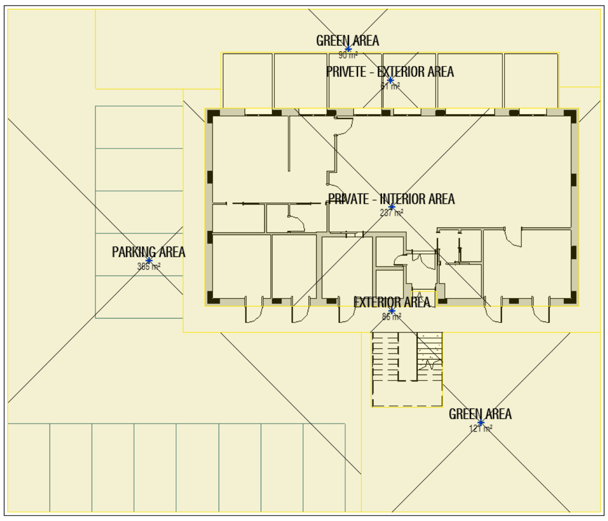


Figure 60 – Area plan with Areas in highlight after running the script

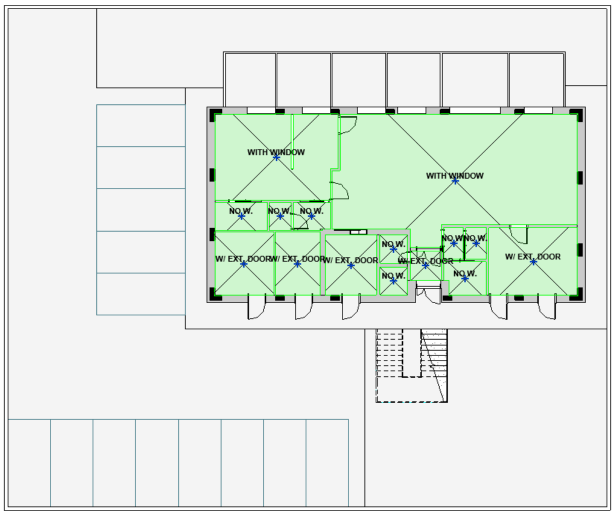


Figure 61 – Floor plan with Revit Spaces in highlight after running the script

3.4.5. Removal of not placed Rooms, Areas, and Revit Spaces

In the process of documentation in Revit, the utilization of tables containing information about Rooms, Revit Spaces, and Areas, is a widely adopted practice to get all the information about it. However, when some of these spaces are deleted in some view, they are actually not deleted from the project, they still appear in the table related to it. To delete them, was necessary to get all of this spaces from the project and check for those that have 0 as the location, and then delete them (Figure 62). Figure 63 shows the floor plan with all the Rooms placed and its table, and Figure 64 shows the floor plan with the Rooms name GARDEN deleted from it and the schedule with those rooms but with the information “not placed”.

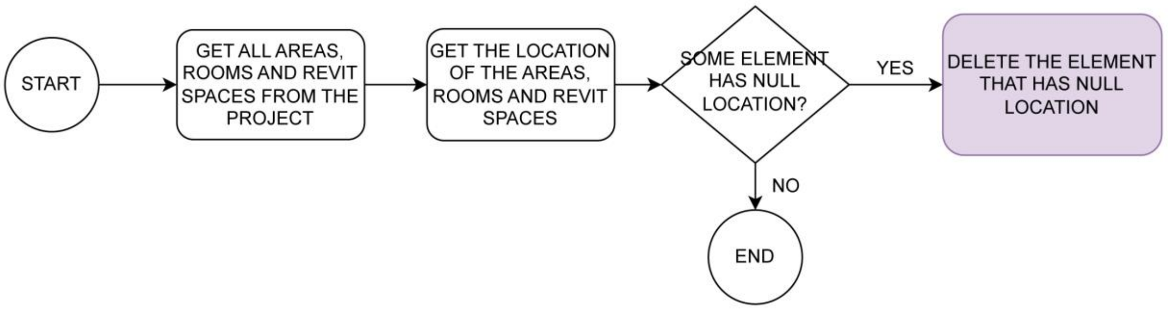
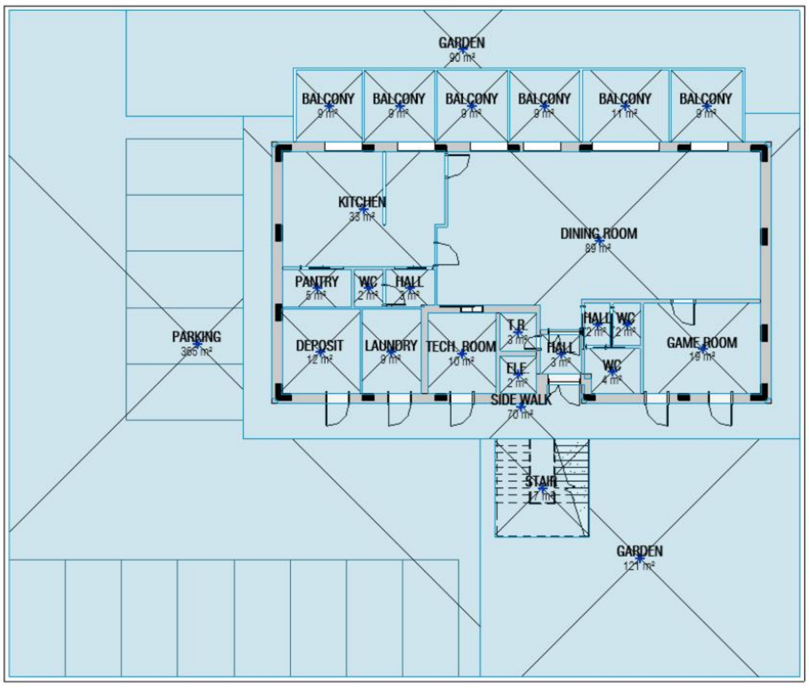
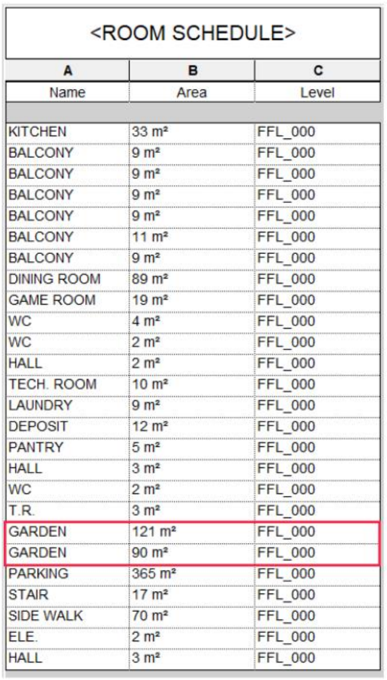

START
GET ALL AREAS, ROOMS AND REVIT SPACES FROM THE PROJECT
GET THE LOCATION OF THE AREAS, ROOMS AND REVIT SPACES
SOME ELEMENT HAS NULL LOCATION?
YES
DELETE THE ELEMENT THAT HAS NULL LOCATION
NO
END

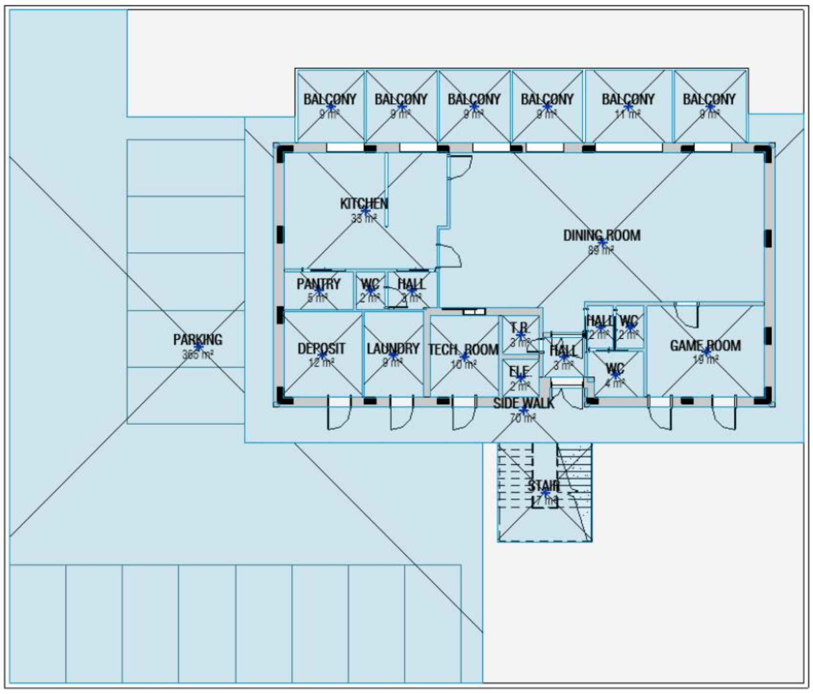
Figure 62 - Process Map of the development of the script to remove not placed Rooms, Areas, and Revit Spaces

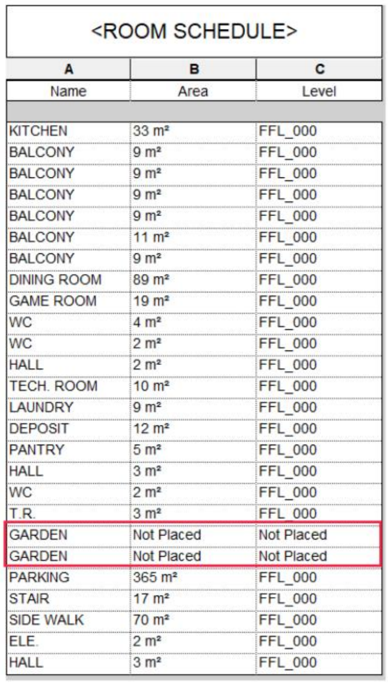



<ROOM SCHEDULE>

A	B	C
Name	Area	Level
KITCHEN	33 m²	FFL_000
BALCONY	9 m²	FFL_000
BALCONY	9 m²	FFL_000
BALCONY	9 m²	FFL_000
BALCONY	9 m²	FFL_000
BALCONY	11 m²	FFL_000
BALCONY	9 m²	FFL_000
DINING ROOM	89 m²	FFL_000
GAME ROOM	19 m²	FFL_000
WC	4 m²	FFL_000
WC	2 m²	FFL_000
HALL	2 m²	FFL_000
TECH. ROOM	10 m²	FFL_000
LAUNDRY	9 m²	FFL_000
DEPOSIT	12 m²	FFL_000
PANTRY	5 m²	FFL_000
HALL	3 m²	FFL_000
WC	2 m²	FFL_000
T.R.	3 m²	FFL_000
GARDEN	121 m²	FFL_000
GARDEN	90 m²	FFL_000
PARKING	365 m²	FFL_000
STAIR	17 m²	FFL_000
SIDE WALK	70 m²	FFL_000
ELE.	2 m²	FFL_000
HALL	3 m²	FFL_000

Figure 63 – Floor plan with Rooms in highlight and the Room Schedule before running the script




<ROOM SCHEDULE>

A	B	C
Name	Area	Level
KITCHEN	33 m²	FFL_000
BALCONY	9 m²	FFL_000
BALCONY	9 m²	FFL_000
BALCONY	9 m²	FFL_000
BALCONY	9 m²	FFL_000
BALCONY	11 m²	FFL_000
BALCONY	9 m²	FFL_000
DINING ROOM	89 m²	FFL_000
GAME ROOM	19 m²	FFL_000
WC	4 m²	FFL_000
WC	2 m²	FFL_000
HALL	2 m²	FFL_000
TECH. ROOM	10 m²	FFL_000
LAUNDRY	9 m²	FFL_000
DEPOSIT	12 m²	FFL_000
PANTRY	5 m²	FFL_000
HALL	3 m²	FFL_000
WC	2 m²	FFL_000
T.R.	3 m²	FFL_000
GARDEN	Not Placed	Not Placed
GARDEN	Not Placed	Not Placed
PARKING	365 m²	FFL_000
STAIR	17 m²	FFL_000
SIDE WALK	70 m²	FFL_000
ELE.	2 m²	FFL_000
HALL	3 m²	FFL_000

Figure 64 – Floor plan with Rooms in highlight, with the room garden removed and the Room Schedule with highlight in the Not Placed Room before running the script

This same thing happens with Area and Revit Spaces and some of them is deleted in a view. The only way to delete it from the whole project, is selecting the space that is unplaced in the table and go to tab

Modify Schedule/Quantities and then, delete it. In this example is just showing one level of a project, but when the project has a big scale and there is a lot of unplaced Rooms, Areas, and Revit Spaces, deleting one by one is a waste of time sub-process. With the script, is just necessary to run it and they will be deleted from the whole project. First it was developed three scripts, one for each type of space (Rooms, Areas, and Revit Spaces). However, recognizing that is more practicality have only one script, they were changed to be in just one, that deletes al of the unplaced spaces, Rooms, Areas, and Revit Spaces, as shown in the Figure 64.

3.4.6. Removal of value 0 from dimension annotations

Within the context of architectural design and documentation within Revit, the process of enhancing the clarity and precision of dimension representation often involves the removal of zero values that occasionally happens, usually when a dimension is copied and pasted from one view to another.

Through the application of Dynamo scripts, users can efficiently identify and eliminate zero values present within dimensions in the project. The script-driven procedure entails the selection of the dimension that has a 0 value, followed by the deletion of dimensions containing a value of zero.

This scripted approach offers several advantages. Firstly, it optimizes the visual legibility of architectural documentation by eliminating unnecessary zero values that can clutter dimension lines. Secondly, it contributes to a streamlined documentation workflow by automating the removal process, saving valuable time, and reducing manual effort. Moreover, it upholds the accuracy and professionalism of the design by ensuring that only relevant dimension values are displayed.

Figure 66 shows that the dimension has several occurrences of 0 values before running the script. This is a common occurrence when several elements share the same space and are mistakenly treated as reference points during the dimensioning process. When using automated dimensioning techniques or when measurements are copied and pasted between similar perspectives, such as in the case of duplicate floor plans within a building, this problem tends to come up more frequently. In Figure 67 shows the same dimension after running the script, with the 0 values deleted from it.

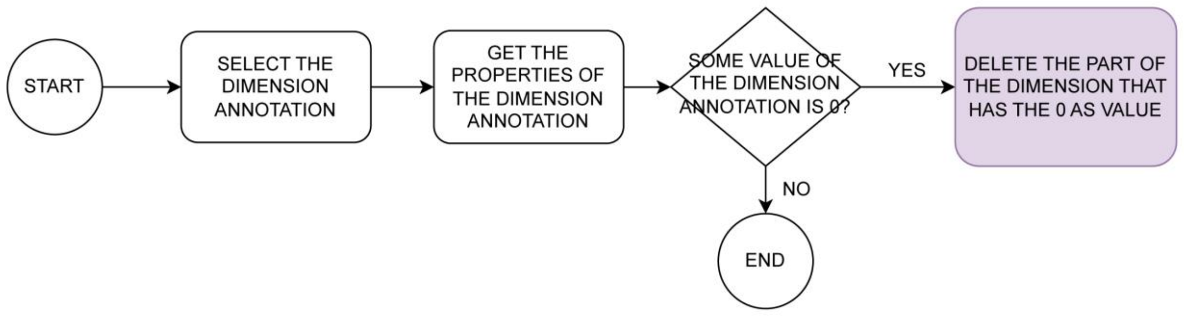

START
SELECT THE DIMENSION ANNOTATION
GET THE PROPERTIES OF THE DIMENSION ANNOTATION
SOME VALUE OF THE DIMENSION ANNOTATION IS 0?
YES
DELETE THE PART OF THE DIMENSION THAT HAS THE 0 AS VALUE
NO
END

Figure 65 - Process Map of the development of the script to remove 0 value from dimension annotations

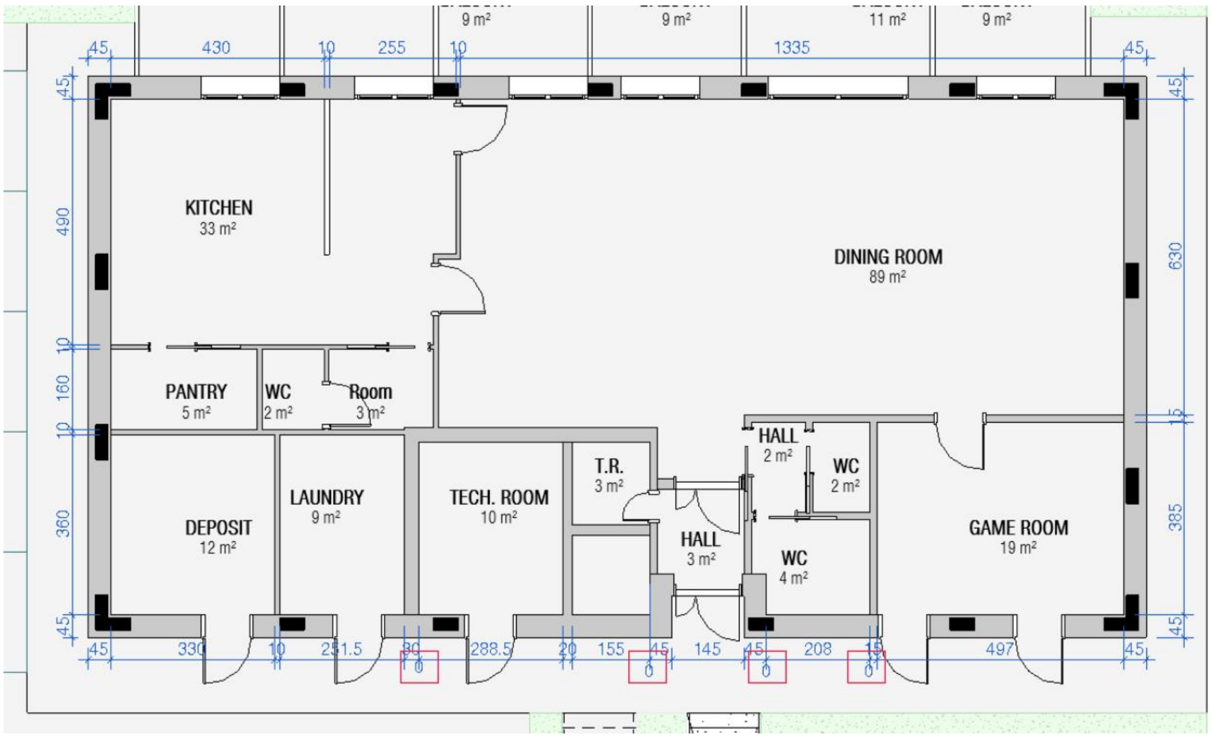


Figure 66 – Floor plan with the dimension annotation with 0 value in highlight before running the script

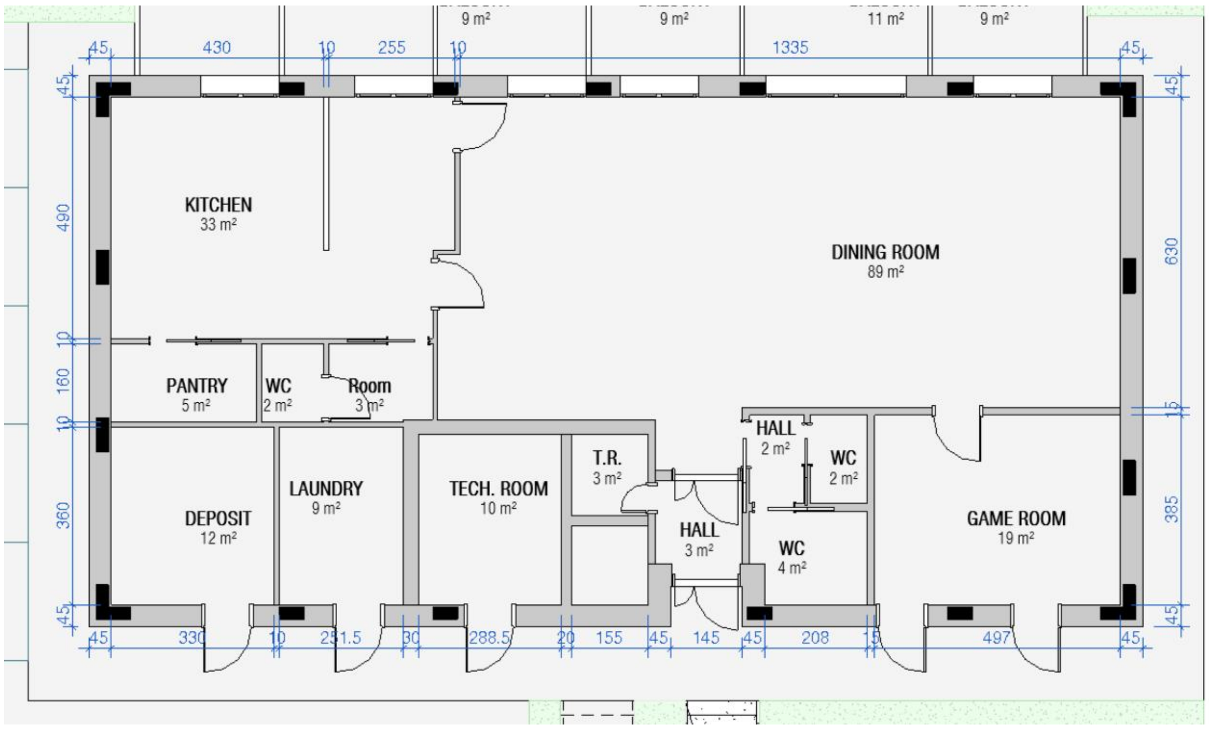


Figure 67 – Floor plan with dimension annotations after running the script

3.4.7. Insertion of light switch referenced to a door

It is common to specify the placement of electrical equipment during the design stage of an architectural project. A recognized standard specifies the location of many fixtures, such as light switches, inside a room. Usually, the light switch is placed on the wall of the door, parallel to the opening side of the door, at the room entry. It is positioned 30 cm away from the entrance and 110 cm above the floor. Placing light switches manually at each door in a project with many doors becomes a standardized and repetitive sub-process.

The developed Dynamo script offers a solution by allowing the selection of a light switch family type and subsequently placing it on all doors associated with the chosen family within the project. By permitting the selection of a light switch family type and then automatically putting it on all doors connected with the selected family inside the project, the produced Dynamo script provides a solution. Although using a nested family was initially under discussion, it was decided that the families would be better off being unconnected. this decision avoids issues with Quantity Take-off (QTO) extraction, potential deletion needs, and changes to the height and distance requirements.

For this task, using a Dynamo script has several benefits. First, it ensures a cohesive and organized arrangement of light switches that aligns perfectly with doors, enhancing design coherence. Additionally, by accelerating design operations, this automated method reduces the need for manual intervention. Furthermore, this approach reinforces design accuracy by decreasing the potential errors that can happen from manual positioning.

Based on the guidelines followed by the NOZ Arquitectura team, the script already specifies the door's height and distance, which are 110 cm from the floor and 30 cm from the door.

The three main elements that make up the script are doors, the walls that go with them, and light switches. Filtering the precise doors—more specifically, the inner doors—to which the light switches should be placed was the first step. This decision stems from the fact that interior doors often follow the specifications covered in this section. Additionally, because they are located inside the structure, the location of the light switches stays constant. On the other hand, since external doors frequently lead outside, it is unusual to find light switches on the outside of the building.

To implement this, the script starts by filtering interior doors and from that, extracts the essential data required to determinate the exact location of the point to install the light switch. The Dynamo node responsible for placing elements presents two options, both require a location point (coordinates) as input. Consequently, accurately determining the point for the element's placement is of extreme importance.

The procedure of the development of the script is based on 3 main elements, the doors, the wall of the doors, and the light switch. First was necessary to filter what doors to put the light switches in, in this case, the interior doors. The reason is that the interior doors use more the standard discussed in this chapter, and since they are interior, the light switch will always stay inside of the building. The exterior doors, usually open to the outside, and is unusual that a light switch is placed outside. It is important to emphasize that the script exclusively addresses the use of swinging doors, excluding sliding doors, due to the fact that when we have a sliding door in a project, the point of placement of the light switch is not on the same wall as the door, but on a perpendicular to it. Therefore, should sliding doors be required, a special and noticeably more complicated script would need to be created to handle this particular situation.

So, the script first filters the interior doors, and from them, gets all the information necessary to get the point to place the light switch. The node from dynamo to place an element has two options, both ask for a point as input, that’s why getting the point where to place the element is important.

To determine the precise location for light switch placement, the script follows this sequence: Initially, it identifies the midpoint of the door. Given that it's the midpoint, the relocation involves moving half of the door's width in addition to the 30cm offset. This point now resides within the door wall's interior, positioned at its midpoint. Consequently, an additional adjustment is needed to align it with the wall's surface, necessitating it to be moved by half of the wall's width.

Once the desired point has been reached, the next steps involve choosing the light switch family and determining the levels for its placement. There is a particular node created to retrieve this information, and the chosen level matches the level of the doors. Following this, the light switch is positioned on the selected level. To match it with the standard height of 110 cm an elevation modification is essential. The final phase is aligning the light switch with the wall's surface, which is accomplished by using a specific node for that (Figure 68).

The distance and height of the light switch were not placed as an input that could be changed. This decision was because they are following the most used standard and because it was a request from NOZ Arquitectura. The focus was on creating a script that necessitated minimal user interaction for practicality, as scripts demanding extensive input are less efficient.

It is important to point out that a previous script was developed involved a more interactive process. In that version, users were required to select the door for light switch placement, specify the distance from the door and the height, and align the position with the door. This process demanded numerous steps and was applicable to a single door, ultimately consuming more time than manually positioning the light switch.

Figure 69 shows the floor plan with the interior sliding doors selected before running the script, and Figure 70 shows the same floor plan after running the script.

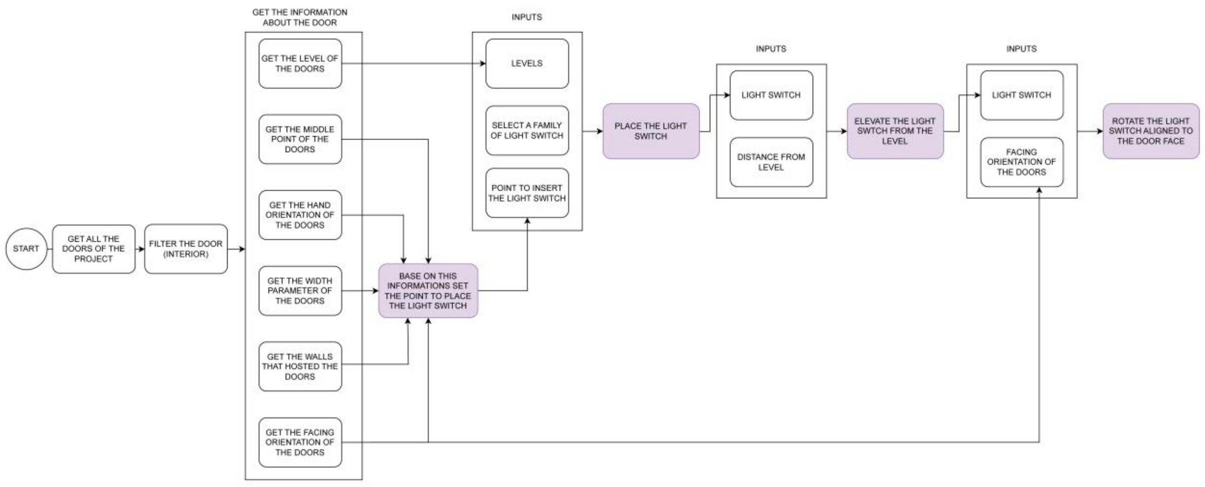

START
GET ALL THE DOORS OF THE PROJECT
FILTER THE DOOR (INTERIOR)
GET THE INFORMATION ABOUT THE DOOR
GET THE LEVEL OF THE DOORS
GET THE MIDDLE POINT OF THE DOORS
GET THE HAND ORIENTATION OF THE DOORS
GET THE WIDTH PARAMETER OF THE DOORS
GET THE WALLS THAT HOSTED THE DOORS
GET THE FACING ORIENTATION OF THE DOORS
BASE ON THIS INFORMATIONS SET THE POINT TO PLACE THE LIGHT SWITCH
INPUTS
LEVELS
SELECT A FAMILY OF LIGHT SWITCH
POINT TO INSERT THE LIGHT SWITCH
PLACE THE LIGHT SWITCH
INPUTS
LIGHT SWITCH
DISTANCE FROM LEVEL
ELEVATE THE LIGHT SWITCH FROM THE LEVEL
INPUTS
LIGHT SWITCH
FACING ORIENTATION OF THE DOORS
ROTATE THE LIGHT SWITCH ALIGNED TO THE DOOR FACE

Figure 68 - Process Map of the development of the script to insert the light switch in all internal opening doors

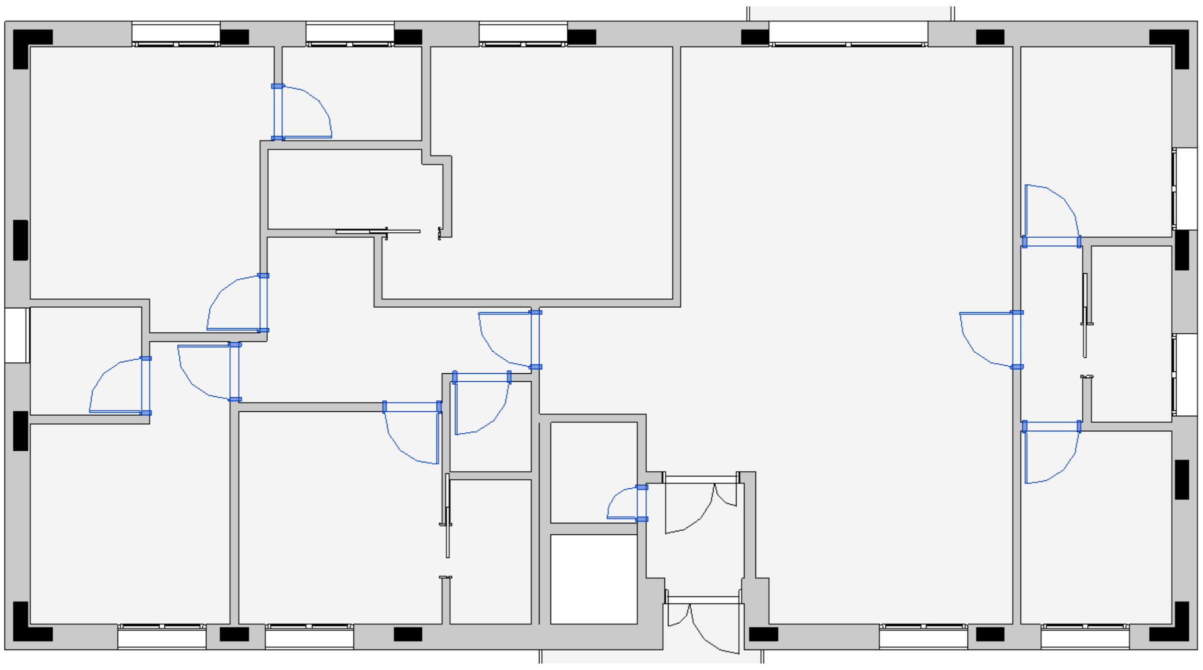


Figure 69 - The floor plan before running the script – highlight in the interior doors

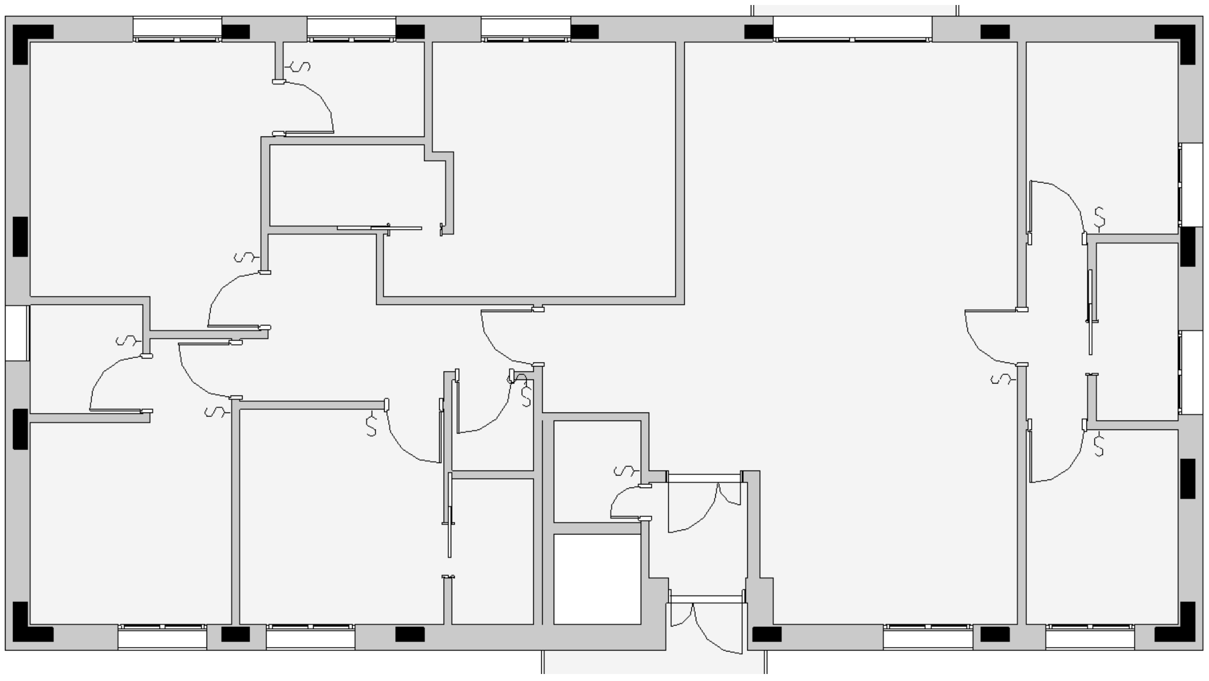


Figure 70 – The floor plan after running the script

Figure 71 shows the light switch with the door and the dimensions in a floor plan and in an elevation after running the script.

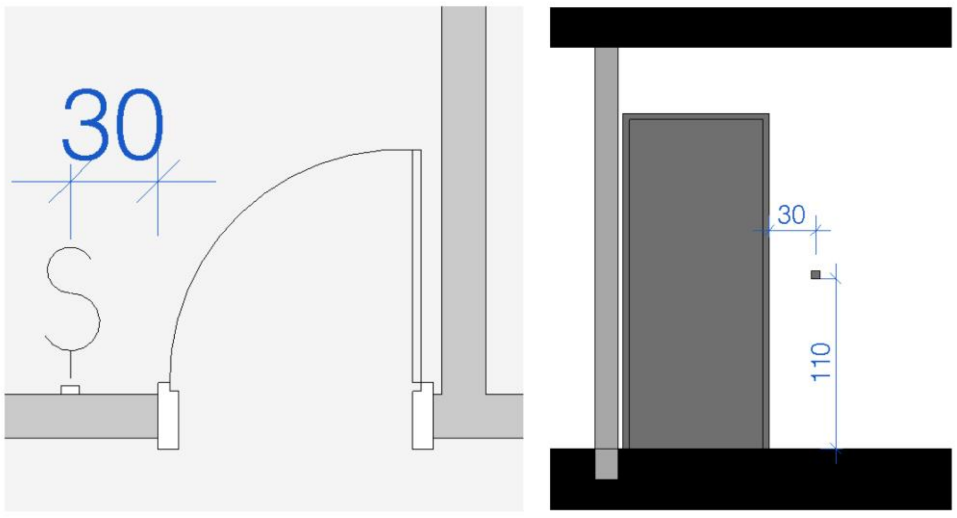


Figure 71 – Floor plan and elevation with the dimensions values used in the script

One of the problems found after running the script is that when there is no space on the wall to place the light switch, the script still runs and places the element, but in a place that is not convenient to the project. In this case, as shown in Figure 72 the user needs to change or delete manually.

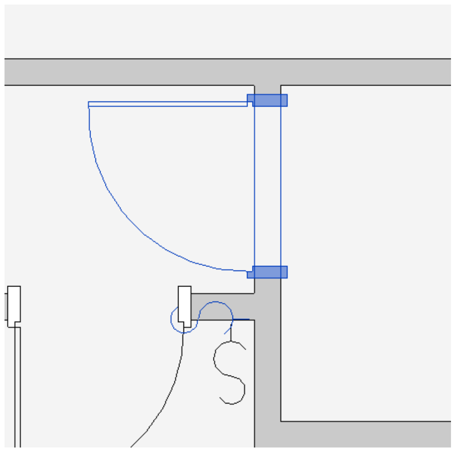


Figure 72 – Floor plan with highlight in the door and light switch with a problm

The distance and height of the light switch were not placed as an input that could be changed. This decision was because they are following the most used standard and because it was a request from NOZ Arquitectura. The focus was on creating a script that necessitated minimal user interaction for practicality, as scripts demanding extensive input are less efficient.

3.4.8. Insertion of spot elevation in a view aligned to the center of all Rooms

A spot elevation in Revit is a graphical annotation that displays the actual elevation of a selected point. Within architectural and construction drawings, it is frequently utilized to express vertical dimensions and elevations of floors, ramps, roads, toposolids, and stair landings. Spot elevations provide height values with a visual representation, making it easier to communicate important information to project stakeholders. Without automation it's only possible to place spot elevations manually on plan views, sections, and elevations, there is no command of placing spot elevation in all elements selected as there is for tagging doors, windows, parking, and more.

With the script of Dynamo is possible to place spot elevation in a selected place, in this case was selected to place it below all the room tags in the current view of the user. It was decided on this place because during the creation of the 2D documentation, usually is the place where the spot elevation is placed. Figure 41 illustrates the floor plan's state before the script is executed, whereas Figure 75 displays the floor plan after the script has been run. The accuracy and efficiency of the documentation process can be greatly improved by this automation.

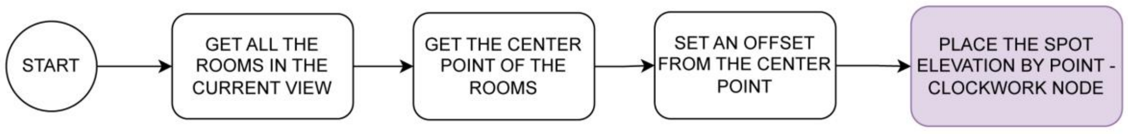

START
GET ALL THE ROOMS IN THE CURRENT VIEW
GET THE CENTER POINT OF THE ROOMS
SET AN OFFSET FROM THE CENTER POINT
PLACE THE SPOT ELEVATION BY POINT - CLOCKWORK NODE

Figure 73 - Process Map of the development of the script to insert spot elevation in a view aligned to the center of all Rooms

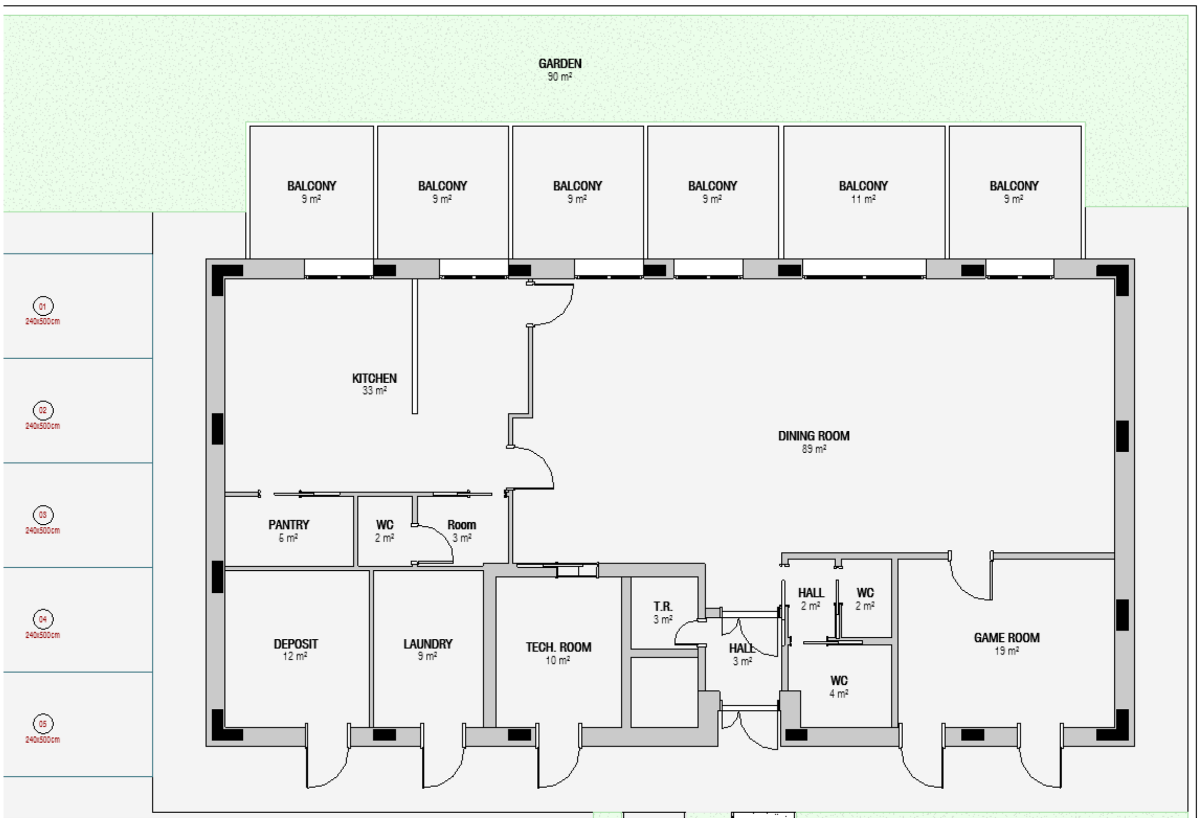


Figure 74 – Flor plan before running the script

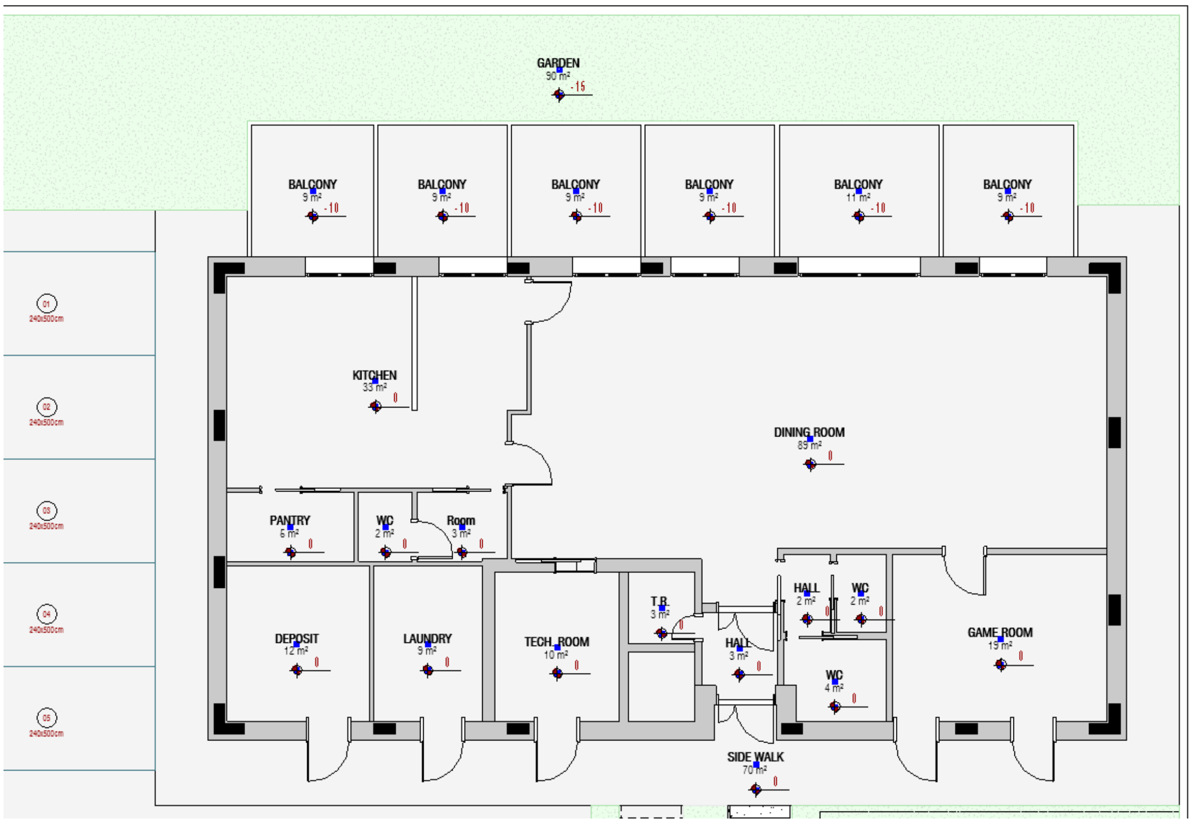


Figure 75 – Floor plan after running the script

By employing the script, users achieve a cohesive floor plan where spot elevations remain aligned with room tags. While not all spaces might necessitate spot elevations, this approach allows for easier removal of undesired instances that were placed. This convenience prevents the laborious task of inserting them one by one, guaranteeing a uniform and consistent result.

3.4.9. Insertion of stair path

Automating the insertion of stair paths through scripting is a significant advantage, streamlining users' workflows by eliminating repetitive manual tasks. Stair paths are pivotal in delineating the direction of stairs and are essential for consistency across all floor plans. The process is efficiently executed using a Crumple package node that prompts for input the stairs requiring the path, the path type, and specific views, in this case, all floor plans.

Illustratively, Figure 77 shows the floor plans' status before script execution, while Figure 78 shows the same floor plans post-execution of the script. This automation ensures that each staircase adheres to the same type of stair path, enhancing both efficiency and uniformity in the project.

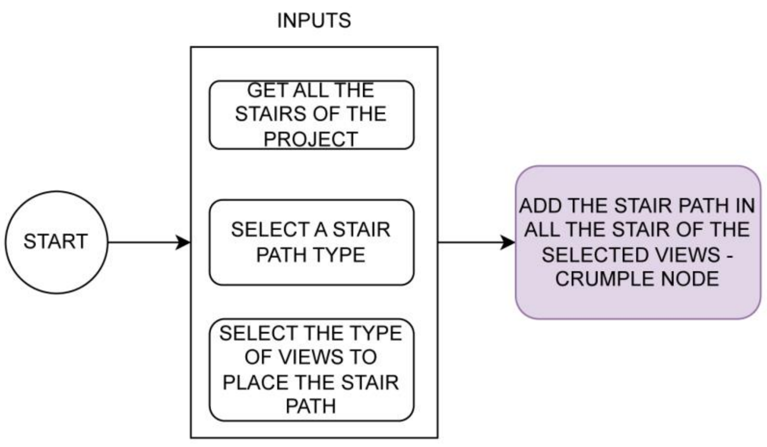

INPUTS
GET ALL THE STAIRS OF THE PROJECT
START
SELECT A STAIR PATH TYPE
ADD THE STAIR PATH IN ALL THE STAIR OF THE SELECTED VIEWS - CRUMPLE NODE
SELECT THE TYPE OF VIEWS TO PLACE THE STAIR PATH

Figure 76 - Process Map of the development of the script to insert stair path

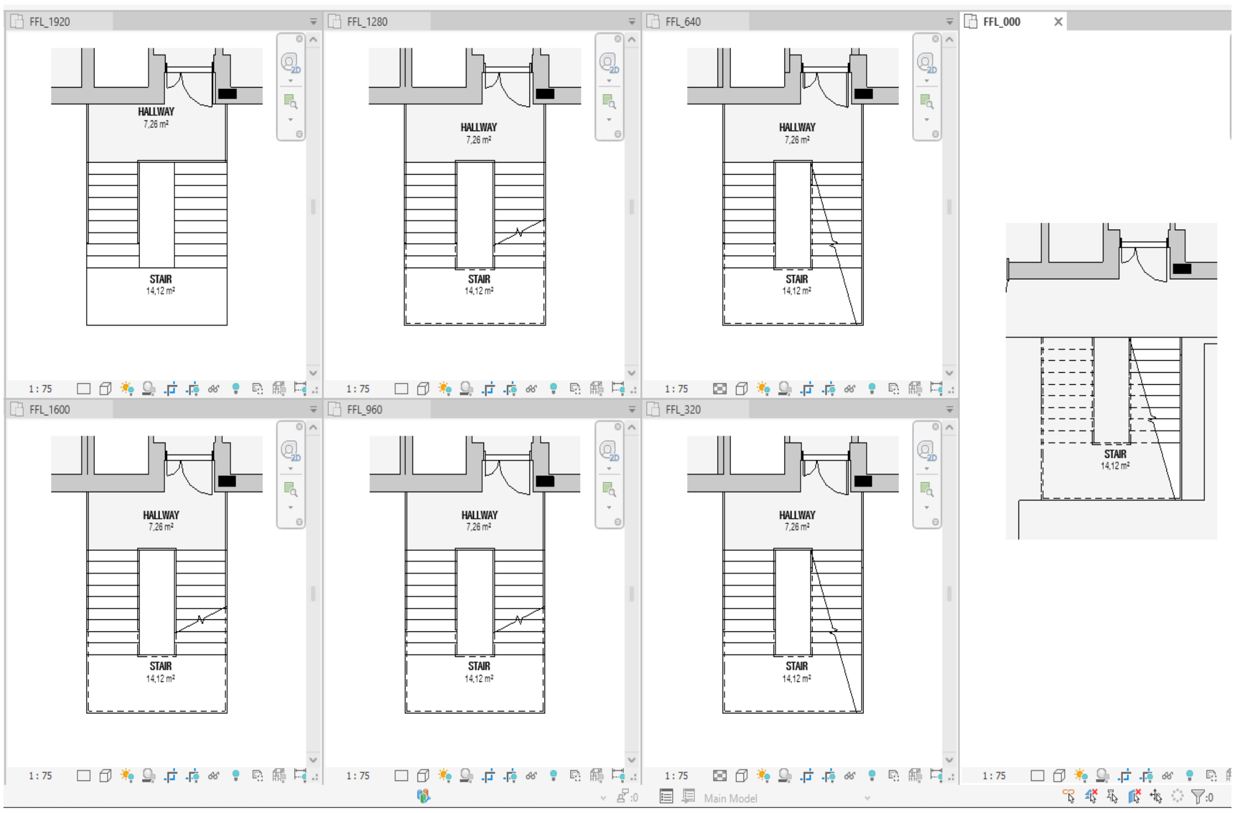


Figure 77 – View of all stair of the project before running the script

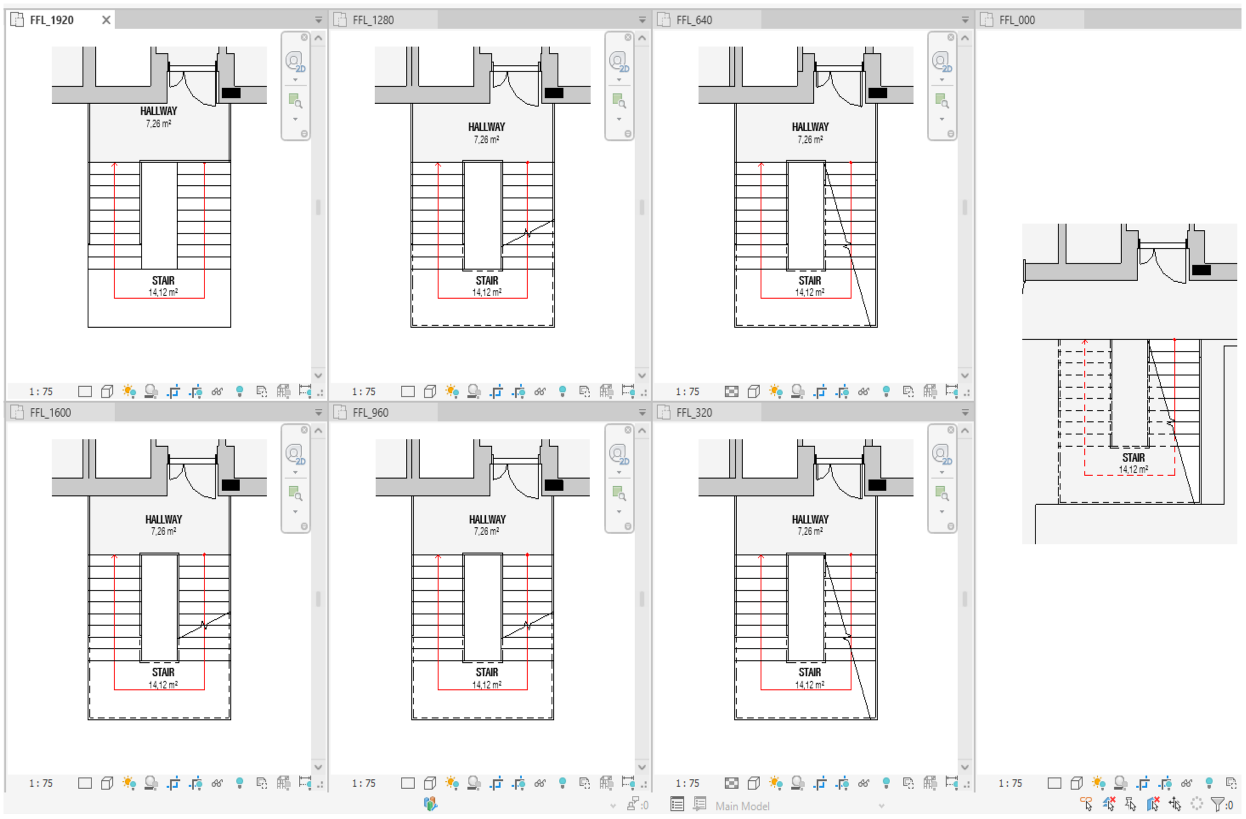


Figure 78 - View of all stair of the project after running the script with the stair path in all of them

3.4.10. Insertion of north symbol in all floor plans

A North symbol in Revit is a graphical element that indicates the direction of the geographic north on a floor plan. It helps viewers quickly understand the orientation of the building in relation to the cardinal directions. (Autodesk, 2023)

Adding North symbols is an important task in architectural documentation, as it provides crucial information about the building's alignment and orientation on-site. Adding them automatically, firstly, ensures consistent placement of north symbols across all floor plans, promoting design coherence and aiding orientation for users. Secondly, the automation accelerates the documentation process, minimizing manual effort and expediting the workflow. Additionally, this approach maintains accuracy by ensuring that the north symbols are aligned precisely with the design intent.

When manually placing north symbols, a distinct instance is required for each view, each necessitating a rotation to align with the project's true north direction. For a more efficient manual approach, the process requires placing and rotating one symbol initially, then employing Revit's "Aligned to Selected Views" function to copy it across a selection of views.

However, this procedure can be further streamlined through scripting. This automation offers a notable advantage, as it requires only two inputs. The first input involves selecting the desired view type for symbol placement. The second input is to determine the project's north angle. The designated point for North symbol placement is the upper right corner of the site. The script uses the property line as a reference, necessitating its prior placement in the project. Once these inputs are provided, the script seamlessly deploys the north symbol across all selected views, ensuring uniform positioning and precise orientation. This not only simplifies the process but also significantly enhances its speed and accuracy.

Visualizing the process, Figure 80 portrays the initial state of the floor plans. Figure 81 illustrates the Nonica pop-up interface, which emerges upon initiating the script. This interface prompts users to input the North angle and the desired view type to apply the symbol. Upon execution, the script dynamically applies the North symbol placement and rotation to the selected views. This transformation is captured in Figure 82, where the floor plans now showcase the aligned North symbols.

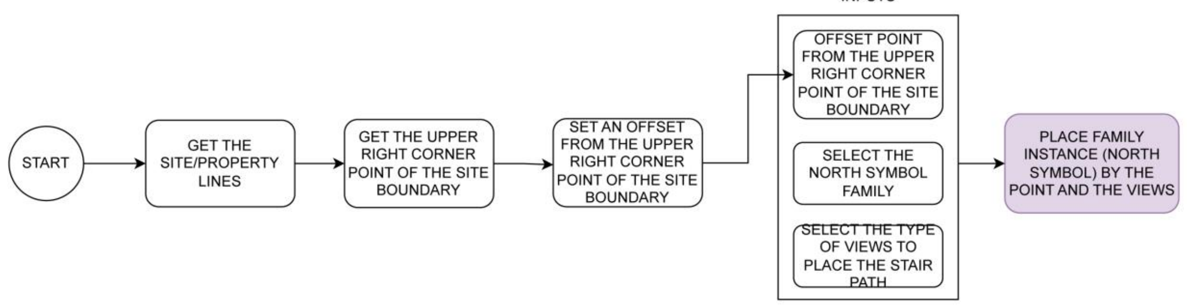

INPUTS
START
GET THE SITE/PROPERTY LINES
GET THE UPPER RIGHT CORNER POINT OF THE SITE BOUNDARY
SET AN OFFSET FROM THE UPPER RIGHT CORNER POINT OF THE SITE BOUNDARY
OFFSET POINT FROM THE UPPER RIGHT CORNER POINT OF THE SITE BOUNDARY
SELECT THE NORTH SYMBOL FAMILY
SELECT THE TYPE OF VIEWS TO PLACE THE STAIR PATH
PLACE FAMILY INSTANCE (NORTH SYMBOL) BY THE POINT AND THE VIEWS

Figure 79 - Process Map of the development of the script to insert north symbol in all floor plans

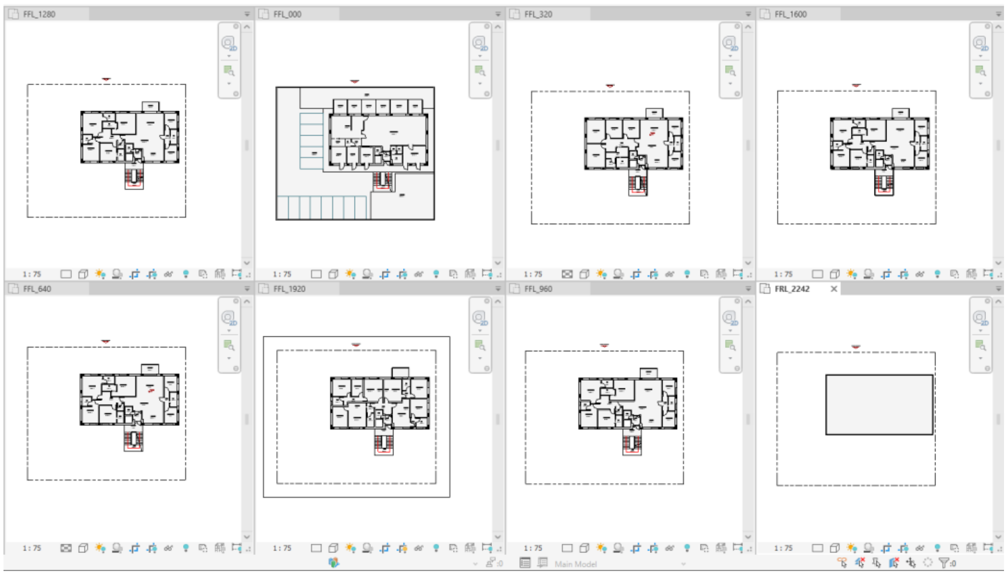
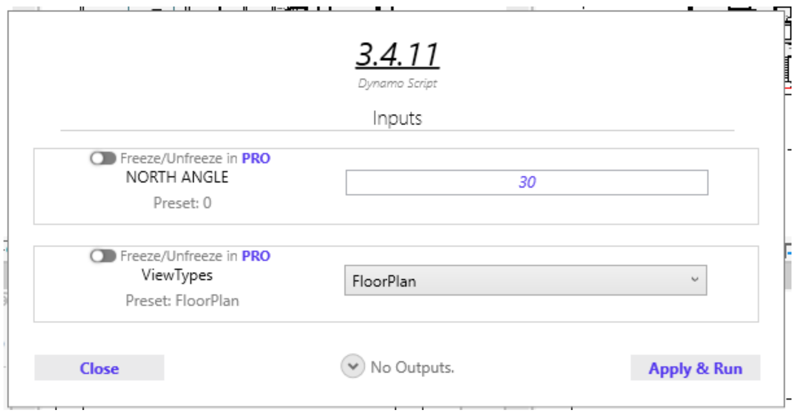


Figure 80 – View with all floor plans of the project before running the script



3.4.11

Dynamo Script

Inputs

Freeze/Unfreeze in PRO
NORTH ANGLE
Preset: 0
30

Freeze/Unfreeze in PRO
ViewTypes
Preset: FloorPlan
FloorPlan

Close
No Outputs.
Apply & Run

Figure 81 – Nonica pop-up to insert the inputs of the script

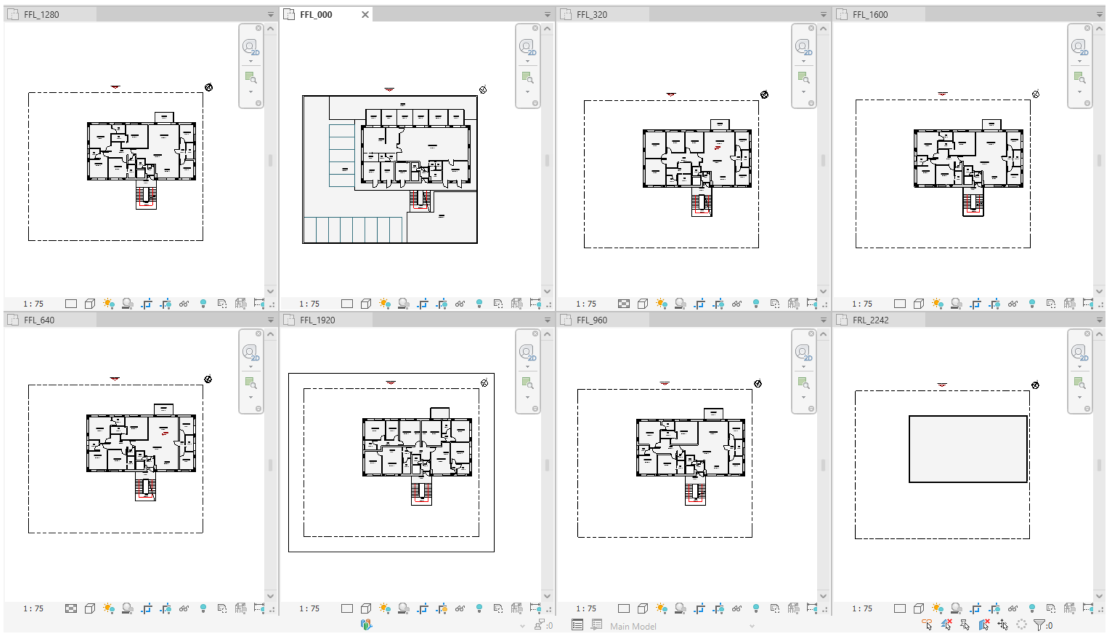
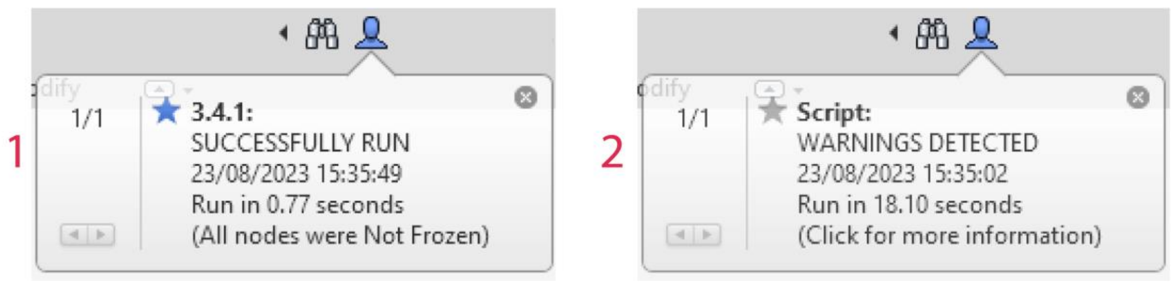


Figure 82 - View with all floor plans of the project after running the script with the north symbol in all of them

3.4.11. Make all the automation scripts available in an easy to use

A specific request from Noz Arquitectura was about the accessibility of the automation scripts developed within this dissertation through a user-friendly mechanism. Initial research into achieving this objective involved the Dynamo Player, an intrinsic facet of the Dynamo platform. However, upon subjecting it to practical assessment, an observation came to the fore—the elapsed duration required for the Dynamo Player to load and subsequently execute the script. The script doesn't stay charged in the player, necessitating every time that it needs to be run, to charge it again.

A solution that effectively addresses this challenge is the Nonica plug-in. This tool offers the capacity to integrate scripts into the toolbar. Once a script is uploaded and executed initially, it becomes archived within the Add-on. Subsequent executions of the same script are expedited due to this archival mechanism. Upon executing the script, it promptly displays a success message or alerts the user if any issues arise (Figure 83). In the latter case, users can click on the message and Dynamo will open to check and fix the problem. In the context of this dissertation, the free version of the Nonica plug-in was used, facilitating the inclusion of up to 12 distinct scripts within its toolbar. The scripts that are incorporated in the toolbar are those that were created for this dissertation, as discussed in the following subchapter (Figure 84).



3.4.1:
SUCCESSFULLY RUN
23/08/2023 15:35:49
Run in 0.77 seconds
(All nodes were Not Frozen)

Script:
WARNINGS DETECTED
23/08/2023 15:35:02
Run in 18.10 seconds
(Click for more information)

Figure 83 – Nonica pop-up messages after running a script

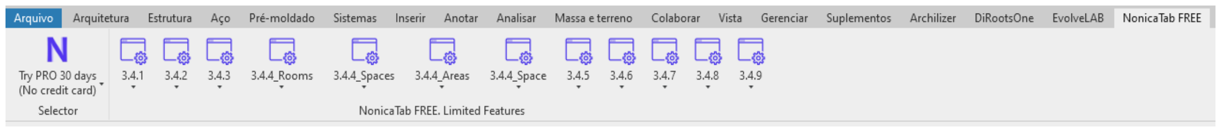


Figure 84 – Nonica Tab Toolbar with all the scripts developed in this dissertation inserted on it

4. CASE STUDY

This chapter investigate the practical application of the Add-ons, Plug-ins, and Scripts that were tested and developed in Chapter 3. These tools are now being put to use in a case study in a project that has already the BIM model modelled, and its functionalities have been adjusted to perform the specific set of activities.

4.1. Description

The use of automation during the design stage in BIM is the main target of this case study, and it consists of proving that by using automation the time spent on doing some tasks can be reduced, and the consistency of the drawing gets more accurate. This evaluation is juxtaposed against the traditional manual methodology of manually inputting information into the drawings. According to the literature review conducted for this study, there has been a lot of work done on BIM automation in the design stage, demonstrating the growing interest in and investment in using digital tools and technologies to improve architectural and engineering design processes within the AEC industry.

To facilitate a comprehensive comparison, a list of tasks has been formulated and they are based in the sub-processes that were researched for automation in the Chapter 3. These tasks are intended to be executed manually as well as through the implementation of automation within the project, using the add-ons, plug-ins and scripts of chapter 3.

To measure the time taken for each task, and to complete all of them will be recorded in an Excel spreadsheet. A number of proactive steps have been done within the file to simplify the job execution process. To be more specific, all Dynamo scripts will be easily available on the Nonica Tab, all project sheets will be set up in advance, all Rooms, Revit Spaces, and Areas will already be placed in the project, and some Rooms, Areas and Revit Spaces were deleted to perform the task about removing the not placed ones.

4.2. Workflow of the tasks

Following is the list of tasks to be performed:

1. Centralization of Rooms, Areas, Revit Spaces and their tags.
2. Removal of not placed Rooms, Areas, and Revit Spaces.
3. Renumbering the parking lots.
4. Placement of tags on all windows and doors, followed by renumbering based on size.
5. Placement of light switches in all interior opening doors.
6. Application of spot elevation in all floor plans.
7. Placement of stair paths across the project's staircases.
8. Placement of north symbols in all floor plans.
9. Dimensioning annotation across all floor plans.
10. Removal of zero values from dimension annotations, if present.

11. Verification of overlapping information in all floor plans
12. Arrangement of all floor plans onto distinct sheets, and alignment of the viewports.
13. Precise alignment of view titles in the left corner of the view port.
14. Incorporation of three revisions within all sheets.

To accomplish these tasks, it was decided to work with floor plans due to their more detailed and need more information on it. The case study will be done in a total of 12 floor plans, consisting of 8 project floor plans, 2 area floor plans, and 2 Revit Space floor plans. The project encompasses a total of 7 stories (Levels: 0, 3,2m, 6,4m, 9,60m, 12,80m, 16m, 19,2m), with 4 of them featuring an identical layout (Levels: 6,4m, 9,60m, 12,80m, 16m), resulting in a total of 4 distinct floor plans.

4.3. Brief information about the project used in the Case Study

The project used in the Case Study encompasses a residential building spanning 7 stories and encompassing a total area of 1670 m², as shown in Figure 85. The BIM model has been modelled employing compound wall and floor families. Noteworthy modifications have been introduced to enhance the project's applicability within the thesis context. These include the addition of extra floors and an expansion of the ground floor to accommodate parking lots.

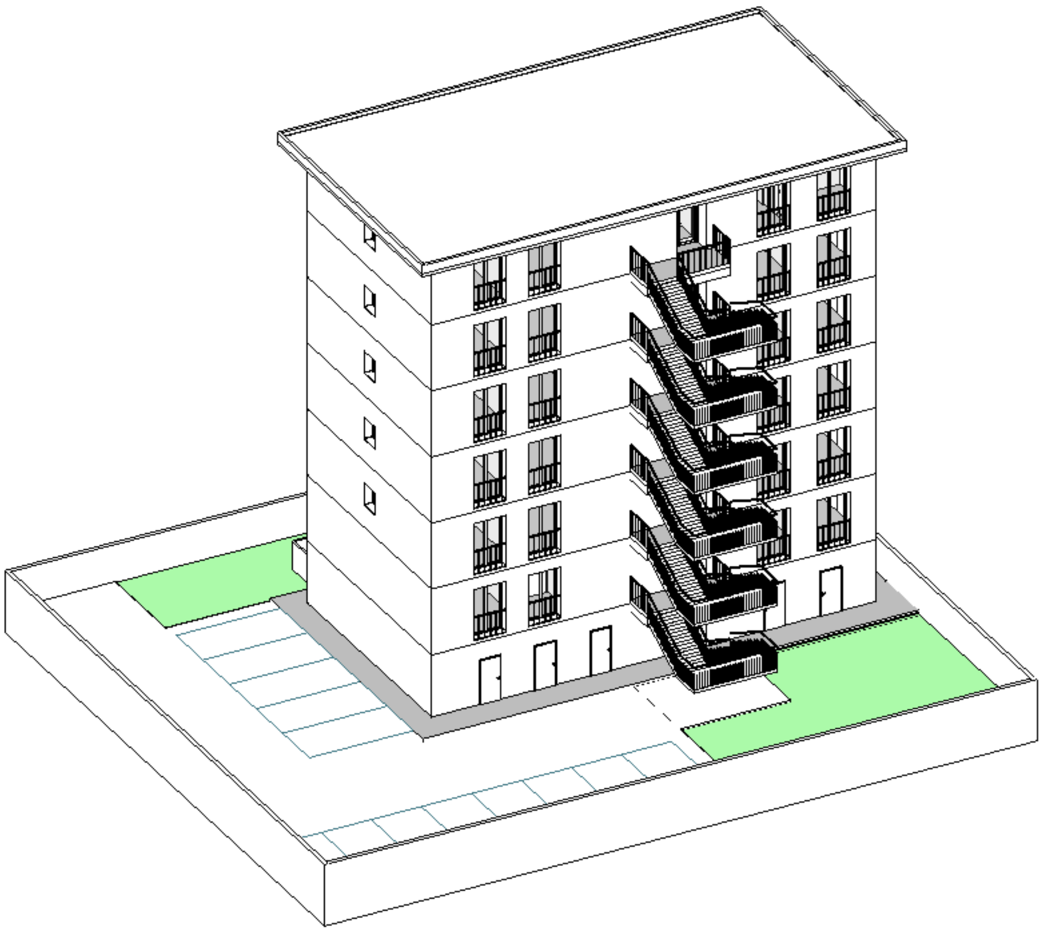


Figure 85 – 3D image of the project used in the case study

4.4. Manual

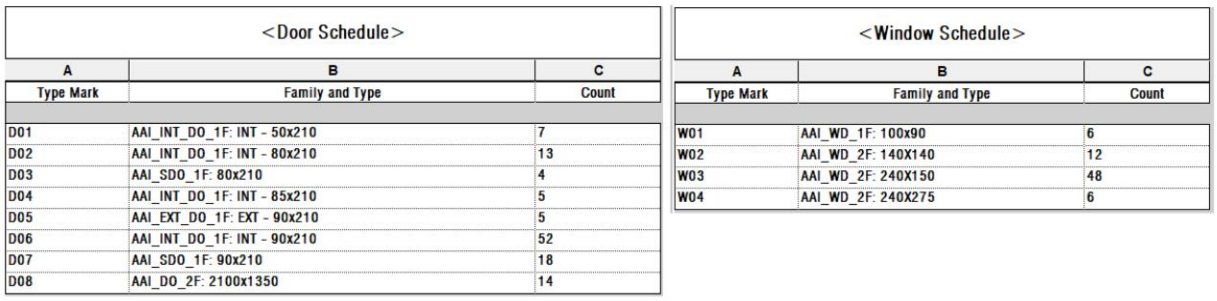
The first case study was doing the tasks with no automation besides some features that Revit has to help, for example, past aligned to current view and past aligned to selected views and tag all elements from a category.

To centralize Rooms, Areas, Revit Spaces, and their corresponding tags in the absence of reference points, an intuitive approach was required. This involved placing them at what was deemed to be the center of each space. To streamline this process, it was essential to enable the visibility of rooms, which is typically turned off, to facilitate selection and movement. Each floor had to be individually addressed. On floors with identical layouts (Levels: 6.4m, 9.60m, 12.80m, 16m), the task was simplified by centralizing them on one floor (Level: 6.4m), remove the rooms that were already in the other levels (Levels: 9.60m, 12.80m, 16m), and then copying and pasting them aligned to those specific levels. This method ensured that, given their identical layouts, the Rooms and tags remained consistently positioned.

The second task, which involved removing unplaced Rooms, Areas, and Revit Spaces, required the creation of a table for each category. Subsequently, identify and select the unplaced elements within these tables, and then delete them from the project.

Renumbering the parking lots in this project proved to be a straightforward task due to the limited count of only 13 parking lots, all situated on the same floor. The process simply involved adjusting each number one by one.

In the subsequent task of tagging all windows and doors and renumbering them based on their size, initially expedited the tagging process by employing Revit's "Tag All" feature for each floor. To facilitate the renumbering based on size, was necessary to create tables for both windows and doors, listing the family types present in the project along with their respective sizes. With this size information, was possible to adjust the Type Mark, which corresponds to the tag value, within the tables (Figure 86).



<Door Schedule>

A	B	C
Type Mark	Family and Type	Count
D01	AAI_INT_DO_1F: INT - 50x210	7
D02	AAI_INT_DO_1F: INT - 80x210	13
D03	AAI_SDO_1F: 80x210	4
D04	AAI_INT_DO_1F: INT - 85x210	5
D05	AAI_EXT_DO_1F: EXT - 90x210	5
D06	AAI_INT_DO_1F: INT - 90x210	52
D07	AAI_SDO_1F: 90x210	18
D08	AAI_DO_2F: 2100x1350	14

<Window Schedule>

A	B	C
Type Mark	Family and Type	Count
W01	AAI_WD_1F: 100x90	6
W02	AAI_WD_2F: 140X140	12
W03	AAI_WD_2F: 240X150	48
W04	AAI_WD_2F: 240X275	6

Figure 86 – Door Schedule and Window Schedule used to renumber the Type Mark

The task of positioning light switches on all interior doors necessitated a one-by-one approach for each door. However, on levels sharing an identical layout, a similar process to centralize Rooms and tags was employed. Initially, this was carried out on one level (Level: 6.4m) and then replicated across the others (Levels: 9.60m, 12.80m, 16m). It's worth noting that during the light switch placement, the elevation value defaults to 0. To follow the standard height of 1.10m, once all switches were in place, the "Select All Instances In Entire Project" command was utilized to swiftly adjust them to the desired height of 1.10m.

The process of applying spot elevations to all floor plans involved a systematic approach, where each floor plan was addressed individually. However, on levels sharing an identical layout, this task was optimized by creating it once and then duplicating it in alignment with the others. Placing the spot elevations in a reference position below the room tags was an intuitive endeavor, as the spot elevation element doesn't rely on references during placement.

To place the stair path for all staircases in the project, even when they occupy identical positions, Revit does not allow copying and pasting this specific element. Consequently, it was necessary to proceed methodically, going through each level individually and placing the stair path within each respective staircase.

Task 8, involving the placement of the north symbol on all floor plans, was executed by initially positioning in the upper right coner and orienting it to match the project's true north in one floor plan. Subsequently, this symbol was copied and pasted in all the remaining floor plans.

One of the most time-consuming tasks during the project involved placing the dimension annotations across all the floor plans. This task demanded a substantial amount of time due to the necessity of accurately referencing the wall positions and carefully positioning the dimensions in appropriate locations, avoiding placement in the middle of spaces. Even when selecting wall faces as reference points for dimensioning, there were instances where the selection defaulted to the wall's midpoint, necessitating additional time to adjust it to the intended face. Once more, for floors sharing identical layouts, the task was initially completed on one of them and subsequently replicated across the others. Notably, because the process was executed manually, it ensured that the dimensions didn't have any value equal 0. Consequently, there was no need to undertake Task 10.

The verification of overlapping information in all the floor plans was made through visual inspection in Revit involves a review of the project to ensure that annotations, text, and dimensions are positioned in a way to avoid overlapping with other elements, ensuring that all information remains legible and comprehensible. In this task, a visual inspection was made and then some elements were moved of place, ensuring consistence on the drawings.

When arranging all the floor plans onto separate sheets and aligning the viewports, a specific technique was employed. After initially placing all the floor plans onto their respective sheets, achieving uniform viewport placement necessitated the creation of two detail lines (Figure 87). These lines were positioned at the corners of the Property Lines on the first sheet, but just an estimation, because is not possible to use as reference any element on the drawing and served as reference points for subsequent sheets. Because these detail lines were established within the sheets themselves, positioning the viewports was based on estimation, as selecting elements directly on the drawings was not possible.

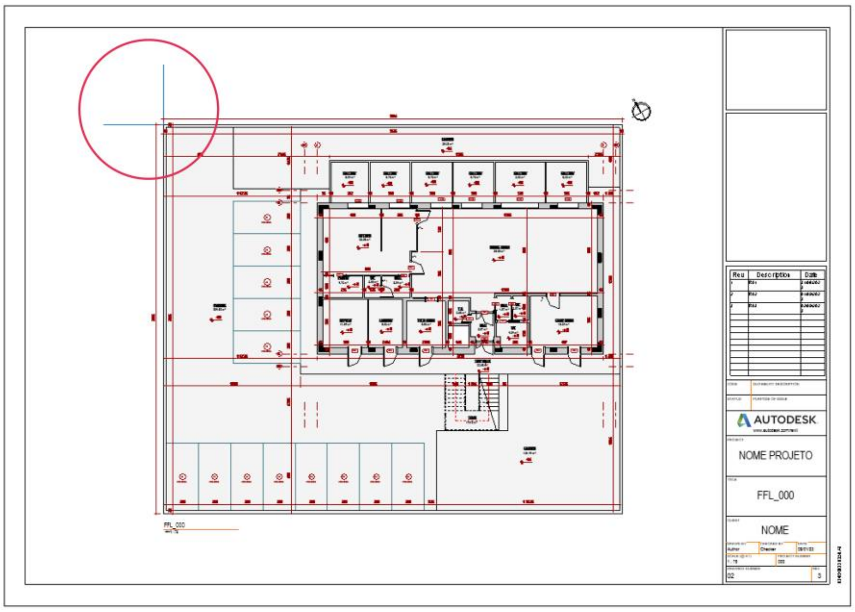


Figure 87 - Sheet with a floor plan view place on it. Highlight to the detail lines made to use as reference to align the views of the other sheets

The same approach of the previous task was used to align the view titles in the left corner. This involved the creation of detail lines to serve as rough reference points for positioning the view titles. Each view title had to be placed individually, and as an estimation using this method (Figure 88).

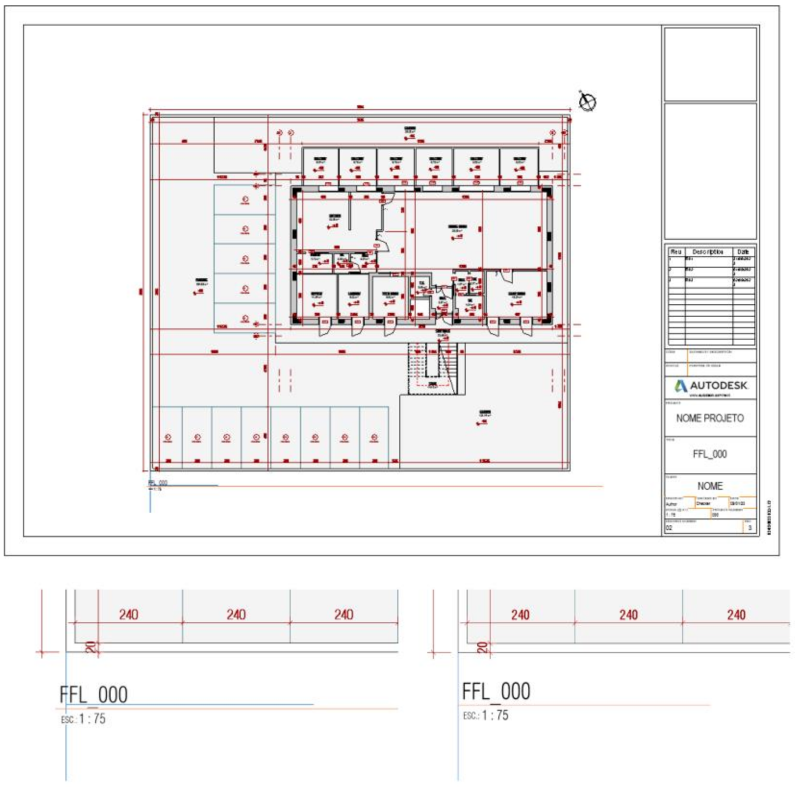


Figure 88 - Sheet with a floor plan view place on it. Highlight to the detail lines made to use as reference to align the view title

During the final task, which utilized Revit 2022, the revision process required a sheet-by-sheet approach. This was necessitated by the absence of a feature in this version that allows simultaneous selection and application of revisions across multiple sheets.

4.5. Automated

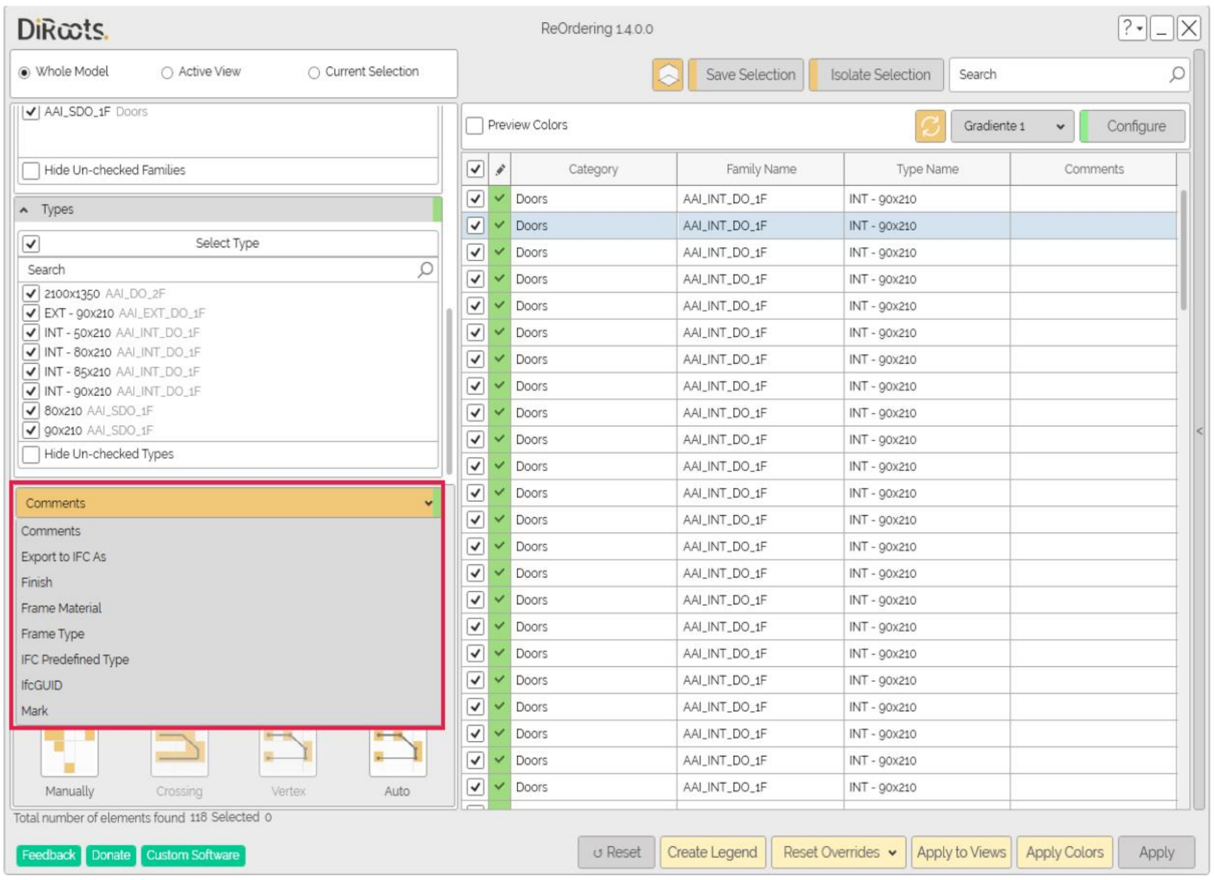
The second case study was doing the tasks with the plug-ins, add-ons and scripts identified and developed in the Chapter 3. Also, some features that Revit has to help, for example, past aligned to current view and past aligned to selected views and tag all elements from a category were used.

When centralizing Rooms, Areas, Revit Spaces, and their corresponding tags, it was necessary to execute the script individually for each floor plan, area plane, and space plan. For those floor plans sharing the same layout, it was easier and quicker to run the script in each of them, than to copy and paste from one that the script already ran.

To remove unplaced Rooms, Areas, and Revit Spaces, a different approach was adopted, diverging from the manual method. In this instance, there was no requirement to create tables for deletion. Instead, a single execution of the script proved sufficient to accomplish this task.

For renumbering the parking lots, the DiRoots Renumbering tool was employed, utilizing the Crossing mode. This approach involved the creation of detail lines spanning across all the parking lots, following the desired numbering direction. Despite the initial step of drawing the detail lines, this method expedited the task, resulting in a swift completion.

For the task of tagging all windows and doors and renumbering them based on their size, initially was done like the manual way, by expedited the tagging process by employing Revit's "Tag All" feature for each floor. To renumber the parking lots based on their size, the initial plan was to utilize the Renumbering tool from DiRoots. However, it was not possible as the tags relied on the Type Mark property for their values, and the tool did not offer the option to renumber based on Type Mark, as shown in Figure 89. Consequently, the approach had to be done as it was in the manual method, using the tables of doors and windows to renumber them.



✓ AAI_SDO_1F Doors

Hide Un-checked Families

✓ 2100x1350 AAI_DO_2F
✓ EXT - 90x210 AAI_EXT_DO_1F
✓ INT - 50x210 AAI_INT_DO_1F
✓ INT - 80x210 AAI_INT_DO_1F
✓ INT - 85x210 AAI_INT_DO_1F
✓ INT - 90x210 AAI_INT_DO_1F
✓ 80x210 AAI_SDO_1F
✓ 90x210 AAI_SDO_1F

Hide Un-checked Types

Comments
- Comments
- Export to IFC As
- Finish
- Frame Material
- Frame Type
- IFC Predefined Type
- IfcGUID
- Mark

✓		Category	Family Name	Type Name	Comments
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	
✓	✓	Doors	AAI_INT_DO_1F	INT - 90x210	

Total number of elements found 118 Selected 0

Figure 89 – DiRoots ReOrdering with the parameters options to re-ordering in highlight

In Task 5, which involved placing light switches in all interior opening doors, the script proved highly effective. A single execution of the script successfully positioned the light switches across all floor plans within the project with the standard distances, 30cm from the door and 110cm from the floor.

Applying spot elevations to all floor plans necessitated executing the script within each individual floor plan. These spot elevations were positioned using references derived from room tags. In projects, there is no requirement to place spot elevations in every single room; they are typically only needed for rooms with elevations different from the level of the floor plan. However, it proved more efficient to initially place spot elevations across all rooms, aligning them to a reference, and subsequently removing the unnecessary ones, as opposed to manually inserting them without a clear reference point.

To implement the stair path across all staircases within the project, a single execution of the script sufficed, ensuring that all stairs received the necessary stair path. However, as the script automatically added the type of stair path loaded in the Revit model, it inserted the "UP" text at the beginning of each path. Typically, using this text is unnecessary since the line itself indicates the stair's direction. Therefore, following the placement, it was necessary to rectify this by selecting all instances using the "Select All Instances In Entire Project" command and deselecting the "UP" text option.

In the subsequent task of placing the north symbol in all floor plans, the initial step involved knowing the angle of the project's north direction, which was subsequently input into the script. With this information in hand, the script was executed just once, and it positioned the north symbol across all floor plans.

To add dimension annotations to all floor plans, the Glyph plugin from EvolveLab was employed, utilizing the line mode. The strategy involved dimensioning the floor plans with distinct layouts individually and, for those with identical layouts, completing the task on one plan and then copying and pasting it to the other levels. The use of the plugin resulted in some dimensions displaying as "0," necessitating the use of a script to remove these zero values individually from each affected dimension. This was done prior to copying and pasting the dimensions onto floor plans with similar layouts.

To check for conflicting information, it was preferable to visually identify the most problematic elements and then use those elements as references to run the script. Initially, this process was carried out using dimensions and spot elevations, followed by dimensions and window tags, and finally dimensions with door tags. By conducting these checks, it was possible to create a cleaner and more legible drawing.

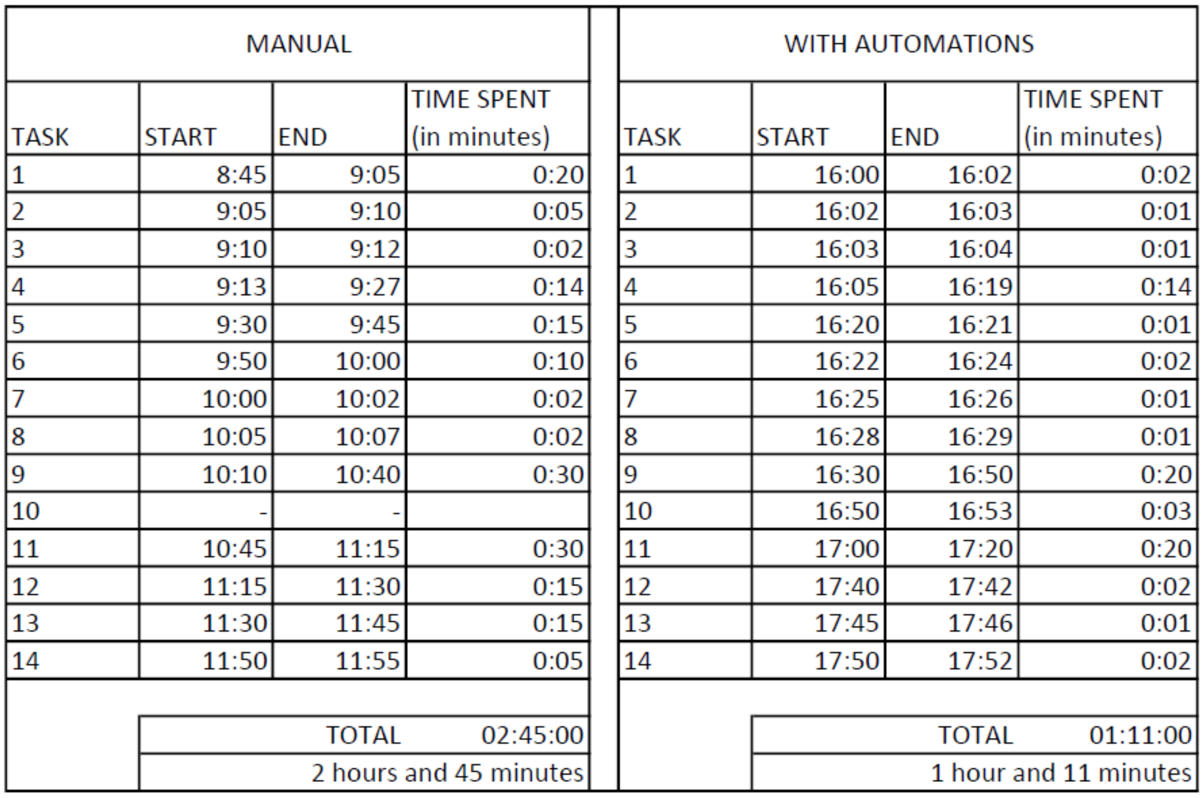
To arrange the drawings onto the sheets and align them across all floor plans, the process was straightforward. All that was required was to place the drawings onto their respective sheets and then utilize the Tiny Tools plugin to align them, using the drawing of the first sheet as the reference.

To align the view titles uniformly in the left corner of the viewports using the script, a single execution proved sufficient. This single action resulted in all the view titles aligning precisely in the same position and maintaining consistent spacing from their respective viewports.

In the final task, which involved adding three revisions to all sheets, was used the DiRoots SheetGen plugin. This powerful tool easily incorporated all the necessary revisions across the sheets designated by the user, streamlining the revision process efficiently.

4.6. Comparison between manual and automated

After conducting case studies, both manually and using automation, the Excel spreadsheet was filled with the time data for each task, resulting in the outcomes shown in Figure 90. This table allows for a comparison of the time taken to complete tasks manually versus the time taken when automation was applied, facilitating an assessment of task efficiency.



MANUAL				WITH AUTOMATIONS			
TASK	START	END	TIME SPENT (in minutes)	TASK	START	END	TIME SPENT (in minutes)
1	8:45	9:05	0:20	1	16:00	16:02	0:02
2	9:05	9:10	0:05	2	16:02	16:03	0:01
3	9:10	9:12	0:02	3	16:03	16:04	0:01
4	9:13	9:27	0:14	4	16:05	16:19	0:14
5	9:30	9:45	0:15	5	16:20	16:21	0:01
6	9:50	10:00	0:10	6	16:22	16:24	0:02
7	10:00	10:02	0:02	7	16:25	16:26	0:01
8	10:05	10:07	0:02	8	16:28	16:29	0:01
9	10:10	10:40	0:30	9	16:30	16:50	0:20
10	-	-		10	16:50	16:53	0:03
11	10:45	11:15	0:30	11	17:00	17:20	0:20
12	11:15	11:30	0:15	12	17:40	17:42	0:02
13	11:30	11:45	0:15	13	17:45	17:46	0:01
14	11:50	11:55	0:05	14	17:50	17:52	0:02
		TOTAL	02:45:00			TOTAL	01:11:00
			2 hours and 45 minutes				1 hour and 11 minutes

Figure 90 - Filled Excel Spreedsheet with the time spent in every task and the total time

As Figure 83 illustrates, the time required to complete the task using automations was significantly less than when performed manually. Manual execution took 2 hours and 45 minutes, while using automations reduced it to just 1 hour and 11 minutes, resulting in a remarkable 57%-time savings. This significant reduction in time not only increases productivity but also minimizes the chance of mistakes that can occur during the tasks manually.

The tasks that experienced the most significant time reduction were as follows: Task 1, which involved centralizing Rooms, Areas, and Revit Spaces; Task 5, focused on placing light switches next to all interior opening doors; Task 6, where spot elevations were applied across all floor plans; Task 12, which dealt with organizing all floor plans onto sheets with aligned viewports; and Task 13, dedicated to aligning view titles to a specific position.

Conversely, several tasks exhibited minimal time differences between manual and automated execution. These tasks included Task 3, the renumbering of parking lots, main reason because they were all in the same level and were just a few of them; Task 7, placing stair paths in all project staircases also because there is just a few of them in the project; Task 8, inserting north symbols in all views, because with the option of Copy and Past In Selected views this task manually becomes easy; and Task 14, incorporating revisions, because there was juts a few sheets in the project.

It is noteworthy that this case study was conducted in the context of a small project, which may account for the relatively small time differences observed in certain tasks. In larger-scale projects, the time-saving benefits of automation become even more pronounced, making it a valuable asset for streamlining project workflows and optimizing efficiency.

When it comes to the quality of drawings, it's evident that automation enhances the consistency of the relationship between annotation elements. Automation allows for the precise placement of annotations using references, whereas manual methods often rely on estimations and approximations, leading to less consistent results.

5. CONCLUSIONS

The objectives of this research have revolved around evaluating the advantages, potential, and challenges of implementing automation within the BIM processes, with a specific emphasis on its applicability to the architectural context, especially during the design phase. This thesis presented a study about the actual automations developed in this context and the existent possibilities to be developed.

In the conduction of the research, the dissertation went through the processes that involves the design stage in order to identify manual time-intensive processes, devise strategies for simplifying them, and determine the most effective approaches for expediting the design process.

Automation emerges as a promising solution to reduce time wastage, eliminate inaccuracies stemming from manual work, and enhance the quality of 2D drawings. Automation is not a new concept in the AEC industry, having been utilized since the early days of computer-aided modeling and simulation. Its goal is to shift tasks from humans to computers and machines, resulting in more consistent and efficient design processes. Moreover, automation is an enabler, uncovering new ideas and project solutions while reducing the burden on designers to ensure comprehensive information in construction and fabrication drawings.

This dissertation demonstrates the advantages of task-specific automation, resulting in considerable time savings for collaborative teams and creating 2D shop drawings that stand out for their accuracy and clarity. The research on existing automations tools and the developed scripts were made to attend the necessities of tasks time consuming during the design stage, especially in the architecture field. The work relied on the collaboration with the architectural firm Noz Arquitectura as an object of study.

The sub-processes identified to be automated were: alignment of the floor plan views in the same place in different sheets; insertion of revisions on sheets automatically; creation of dimension annotations automatically; element renumbering; make all the automation scripts available in an easy to use; alignment of the view name of the viewport in a chosen position; alignment of the view name with the viewport; detection of overlapping information; centralization of rooms, spaces, areas, and their respective tags; removal of not placed rooms, spaces, areas, and their respective tags; removal of value 0 from dimension; insertion of light switch referenced to a door; insertion of spot elevation in a view aligned to all room tags; insertion of the stair path; and insertion of north symbol in all floor plans.

For solving them with automation, an investigating of currently available automated options was made, and for those that a solution was not founded in plug-ins or add-ons, a creation of Dynamo routines were made. After that was performed a comparative study case between manual and automated way of perform a workflow of tasks related to the sub-processes.

During the case study conducted as part of this research, remarkable results were achieved, demonstrating a remarkable 57% reduction in time spent when utilizing automation in the design phase for specific tasks outlined in this dissertation. These findings underscore the substantial potential for automation to revolutionize and streamline architectural processes, paving the way for greater efficiency, precision, and innovation within the AEC industry.

This dissertation has successfully accomplished the primary objective of comparing the utilization of automation tools during the design phase within the architecture field. The research journey has yielded valuable insights into the profound impact that automation can have on various aspects of architectural design and documentation.

In conclusion, this research contributes to an enriched understanding of automation's role in architectural practice within the context of BIM implementation. By methodically exploring automation functionalities, market offerings, and customized scripting, this study underscores the potential advantages of automation, particularly in terms of time efficiency and the quality of project documentation. It confirms the idea that, in the AEC sector, automation serves as an engine for innovation and efficiency rather than merely a tool.

5.1. Future developments

While this dissertation has effectively accomplished its core objectives and produced valuable insights into the application of automation in architectural design during the conceptual phase, it is crucial to acknowledge that there are areas and opportunities for further exploration and development.

The add-ons, plug-ins, and scripts could be tested in an across a broader spectrum of project types and sizes. While the research was conducted on a small-scale project, the applicability of these automation tools to larger, more complex projects remains unexplored. Investigating their effectiveness and scalability in various contexts, such as commercial buildings, multi-use developments, or even public infrastructure projects, would provide a more comprehensive understanding of their potential impact.

In architectural field, many automations can be developed, especially with the use of artificial intelligence (AI) and machine learning (ML) that can allow architects to easily generate multiple design layout and façade options and optimize them based on specific performance criteria.

Expanding the application of automation tools beyond architectural design to encompass other disciplines within the AEC industry is another promising avenue for future development. Structural engineering and MEP (Mechanical, Electrical, Plumbing) design are integral components of building projects. Integrating automation into these disciplines during the design stage could lead to more seamless collaboration and coordinated designs. Further research could explore the development of cross-disciplinary automation tools and workflows that foster integration and enhance overall project efficiency.

The usability and accessibility of automation tools are critical factors in their successful adoption within the industry. Future developments could focus on refining the user interfaces (UI) of these tools to make them more intuitive and user-friendly than the one used in this thesis. Additionally, efforts could be directed towards creating user guides, tutorials, and training resources to facilitate the adoption of automation among architectural professionals, regardless of their proficiency in scripting or programming.

Given the diverse nature of architectural projects and design approaches, enabling greater customization and adaptability of automation scripts is essential. Future iterations of these tools could incorporate more flexible parameters and options, allowing architects and designers to fine-tune automation processes to align with their specific project requirements and design philosophies.

In conclusion, while this dissertation has laid a solid foundation for understanding the benefits of automation in architectural design, it also points the way forward for future developments and refinements. As the AEC industry continues to evolve, embracing automation as an integral part of the design process holds the promise of greater efficiency, creativity, and precision. The path ahead involves continuous innovation, collaboration, and adaptation to meet the evolving needs and challenges of architectural practice.

REFERENCES

Aghabayli, A. (2021). *Machine Learning Applied to Building Information Models*. Universidade do Minho.

Aish, R. (2011). *DesignScript: origins, explanation, illustration*.

Aish, R. (2013). *DesignScript*.

An, S., Martinez, P., Al-Hussein, M., & Ahmad, R. (2020). BIM-based decision support system for automated manufacturability check of wood frame assemblies. *Automation in Construction*, *111*. https://doi.org/10.1016/j.autcon.2019.103065

Asl, M. R., Zarrinmehr, S., Bergin, M., & Yan, W. (2015). BPOpt: A framework for BIM-based performance optimization. *Energy and Buildings*, *108*, 401–412.

Autodesk. (2023a). *Autodesk Platform Services*. https://aps.autodesk.com/developer/overview/revit

Autodesk. (2023b). *Tiny Tools*. https://apps.autodesk.com/RVT/en/Detail/Index?id=314878267117975121&appLang=en&os=Win64

BIMe. (2019). *BIM Dictionary*. https://bimdictionary.com/en/bim-workflow/1

Bråthen, K., No, K., Moum, A., & No, A. M. (2015). *Bridging the gap: taking BIM to the construction site*.

Brell-Cokcan, S., & Braumann, J. (2014). *ROBOTIC PRODUCTION IMMANENT DESIGN CREATIVE TOOLPATH DESIGN IN MICRO AND MACRO SCALE*.

Building Smart. (2023). *Building Information Modeling*. https://buildingsmart.pt/o-que-e/

Choi, J., Lee, Y., & Kim, I. (2018). *Development of Application for Generation of Automatic 2D Drawings based on openBIM*.

Czmoch, I., & Pękala, A. (2014). Traditional Design versus BIM Based Design. *Procedia Engineering*, *91*, 210–215. https://doi.org/https://doi.org/10.1016/j.proeng.2014.12.048

Davies, R., & Harty, C. (2013). Implementing "site BIM": A case study of ICT innovation on a large hospital project. *Automation in Construction*, *30*, 15–24. https://doi.org/10.1016/j.autcon.2012.11.024

Davis, D., & Peters, B. (2013). Design Ecosystems: Customising the Architectural Design Environment with Software Plug-ins. *Architectural Design*, *83*(2), 124–131. https://doi.org/10.1002/ad.1567

Deng, M. (2018). *BIM-based Automatic Generation of Fabrication Drawings for Building Façades*.

DiRoots. (2023a). *ReOrdering*. https://diroots.com/revit-plugins/renumber-revit-elements-using-a-prefix-a-suffix-and-a-multiplier-with-reordering/

DiRoots. (2023b). *SheetGen*. https://diroots.com/revit-plugins/batch-create-revit-sheets-and-place-views-with-sheetgen/

Disney, O., Roupé, M., Johansson, M., & Domenico Leto, A. (2022). Embracing BIM in its totality: a Total BIM case study. *Smart and Sustainable Built Environment*. https://doi.org/10.1108/SASBE-06-2022-0124

Dixon, J. R. (1995). Knowledge-Based Systems for Design. *Journal of Mechanical Design*, *117*, 11–16. https://api.semanticscholar.org/CorpusID:121897939

Dynamo Primer. (2023). *The Dynamo Primer*. https://primer.dynamobim.org/en/index.html

Ermolenko, E. (2020). *Algorithm-aided Information Design: Hybrid Design approach on the edge of Associative Methodologies in AEC*. Universidade do Minho.

EvolveLab. (2023). *Glyph*. https://www.evolvelab.io/glyph

Frits, E. (2023). *EF-Tools*. https://github.com/ErikFrits/EF-Tools

Groover, M. (2023). *Automation*. https://www.britannica.com/technology/automation

Kensek, K. M. (2014). Integration of Environmental Sensors with BIM: Case studies using Arduino, Dynamo, and the Revit API. *Informes de La Construccion*, *66*(536). https://doi.org/10.3989/ic.13.151

Kim, Y., Chin, S., & Choo, S. (2022). BIM data requirements for 2D deliverables in construction documentation. *Automation in Construction*, *140*, 104340. https://doi.org/10.1016/J.AUTCON.2022.104340

Kreaker, M. G., & Ramírez Verdyguer, I. (2019). *BIM-WHAT IS BEYOND 2D DRAWINGS A Review of the Design Phase and the Perspective of an Industry without 2D Drawings Master's thesis in Design and Construction Project Management*.

Leitão, A., & Santos, L. (2011). Programming Languages for Generative design. *Respecting Fragile Places: 29th ECAADe*, 549–557.

Leśniak, A., Górka, M., & Skrzypczak, I. (2021). Barriers to bim implementation in architecture, construction, and engineering projects—the Polish study. *Energies*, *14*(8). https://doi.org/10.3390/en14082090

Miettinen, R., & Paavola, S. (2014). Beyond the BIM utopia: Approaches to the development and implementation of building information modeling. *Automation in Construction*, *43*, 84–91. https://doi.org/10.1016/j.autcon.2014.03.009

Monteiro, A. (2016). *Visual Programming Language for Creating BIM Models with Level of Development 400*. www.dharmasistemas.wix.com/

Mukkavaara, J. (2018). *Structures for Supporting BIM-Based Automation in The Design Process*. www.ltu.se/shb

Murvold, V., Vestermo, A., Svalestuen, F., Lohne, J., & Laedre, O. (2016). *EXPERIENCES FROM THE USE OF BIM-STATIONS*. www.iglc.net

NBS. (2021). *What is BIM?* https://www.thenbs.com/knowledge/what-is-building-information-modelling-bim

Nonica. (2023). *Autodesk App Store*. https://apps.autodesk.com/RVT/en/Detail/Index?id=2476142006549788030&appLang=en&os=Win64

Oliveira, M. A. R. de. (2022). *Integrated BIM tools for Digital Building Permit*. Politecnico di Milano.

pyRevit. (2023). *pyRevit*. https://pyrevitlabs.notion.site/

Python.NET. (2023). *Python.NET*. http://pythonnet.github.io/

RIBA. (2020). *RIBA Plan of Work 2020 Overview*. www.ribaplanofwork.com

Robert Aish. (2011). DesignScript: Origins, Explanation, Illustration. *Computational Design Modelling*, 1–8.

Rossi, G. (2020). *The role of Computational Design in BIM*. Univerzav Ljubljani.

Sandberg, M., Gerth, R., Lu, W., Jansson, G., Mukkavaara, J., Se, J. M., & Olofsson, T. (2016). *Design automation in construction-an overview*.

Sena, P. C. P. de. (2019). *AUTOMAÇÃO DE PROCESSOS DE PROJETO E PROGRAMAÇÃO EM BIM: DYNAMO, PYTHON E C#*.

Succar, B. (2009). Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in Construction*, *18*(3), 357–375. https://doi.org/10.1016/j.autcon.2008.10.003

Sundquist, V., Leto, A., Gustafsson, M., Johansson, M., & Roupé, M. (2020, September). *BIM in Construction Production: Gains and hinders for firms, projects and industry*.

Terzidis, K. (2006). *Algorithmic Architecture*.

TestFit. (2023). *TestFit*. https://testfit.io/

Vanegas, J. (2022). *Automatic Rule Verification for Digital Building Permits*. Universidade do Minho.

Vieira, L. L. (2020). *Coordinated specification and quantity take-off through digital modelling*. Universidade do Minho.

Zhang, C., Zou, Y., & Dimyadi, J. (n.d.). *A Systematic Review of Automated BIM Modelling for Existing Buildings from 2D Documentation*.

Zhang, C., Zou, Y., & Dimyadi, J. (2021). A Systematic Review of Automated BIM Modelling for Existing Buildings from 2D Documentation. *Proceedings of the International Symposium on Automation and Robotics in Construction*, *2021-November*, 220–226. https://doi.org/10.22260/isarc2021/0032

LIST OF ACRONYMS AND ABBREVIATIONS

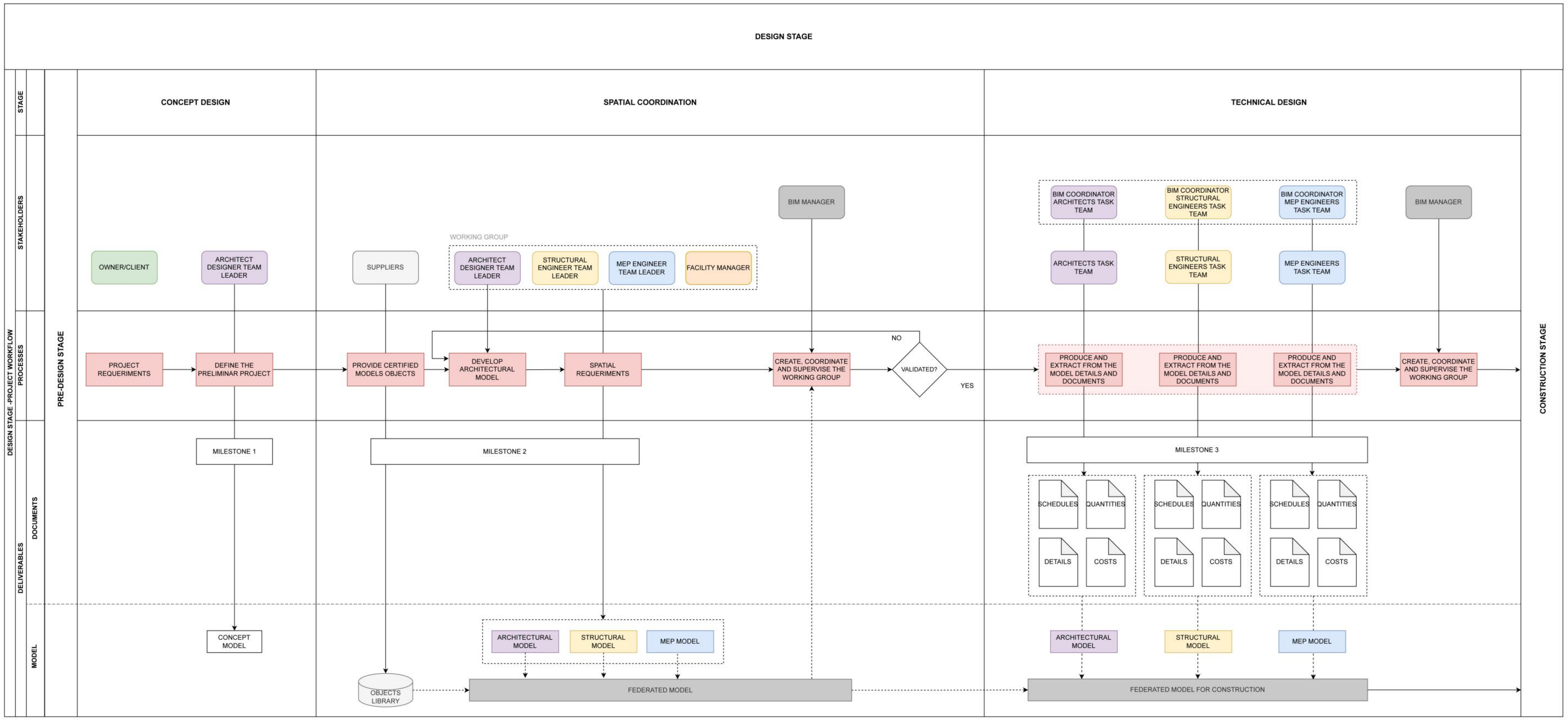
They are to be listed transparently, logically separated, in column (tabular) form. Acronyms / abbreviations / translations that are not in the public domain should be indicated. Acronyms / abbreviations / translations in the list are to be listed in alphabetical order. Acronyms and abbreviations are to be indicated in the List of Acronyms and Abbreviations.

Example:

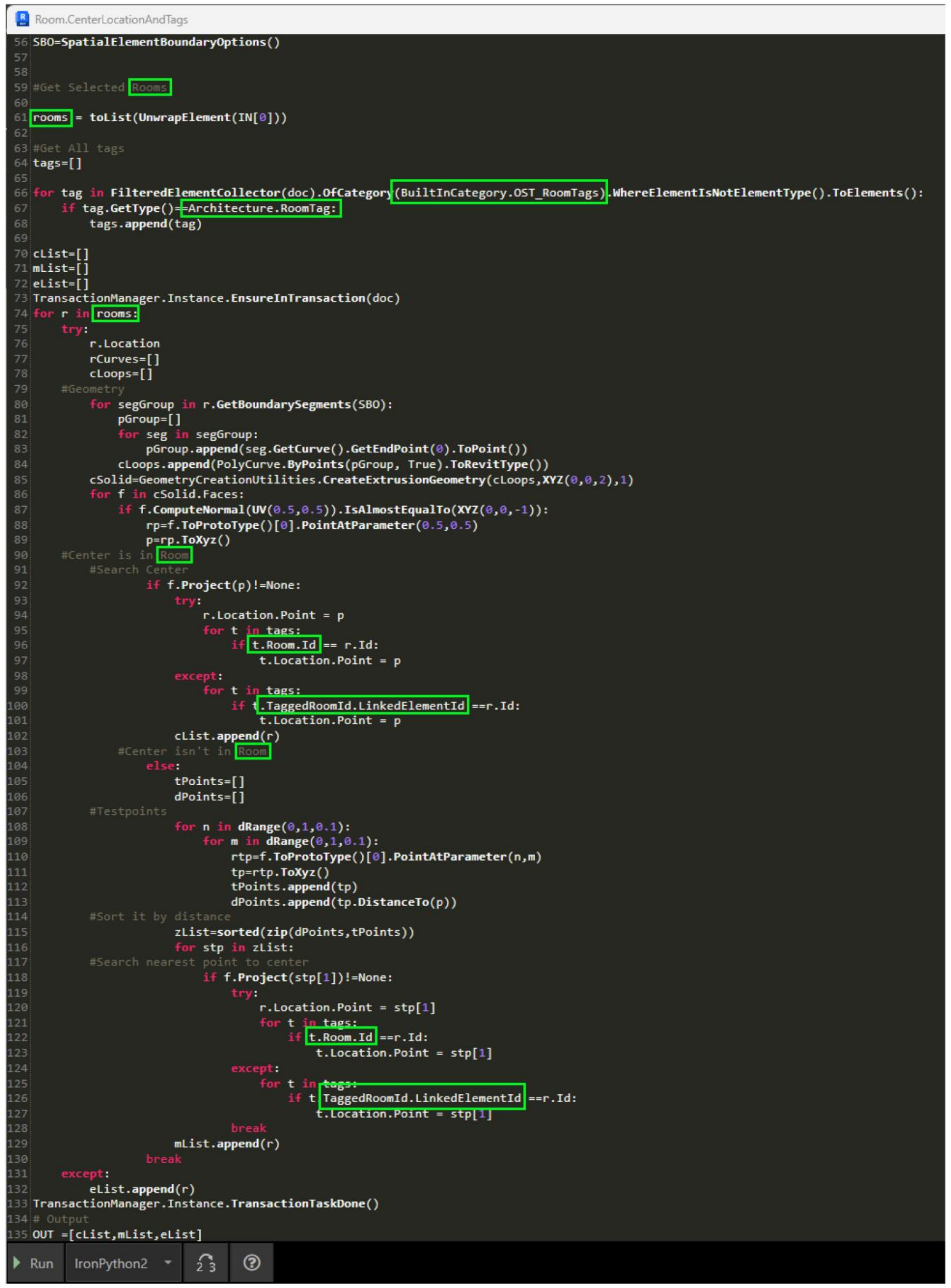
2D	Two Dimension
3D	Three Dimension
AEC	Architectural, Engineering, and Construction
API	Application Programming Interface
BEP	Building Execution Plan
BIM	Building Information Modelling
CAD	Computer-aided-Design
CDE	Common Data Environment
CLR	Common Language Runtime
DLL	Dynamic Link Libraries
DXF	Drawing Exchange Format
EIR	Exchange Information Requirements
EIR	Employers Information Requirements
GUI	Graphical User Interface
IDM	Information Delivery Method
IFC	Industry Foundation Classes
ISO	International Organization for Standardization
LOD	Level of Development
PL	Programming Language
RAD	Rapid Application Prototyping
TPL	Textual Programming Language
VPL	Visual Programming Language

APPENDICES

APPENDIX 1: ARCHITECTURAL PROJECT WORKFLOW IN THE DESIGN STAGE

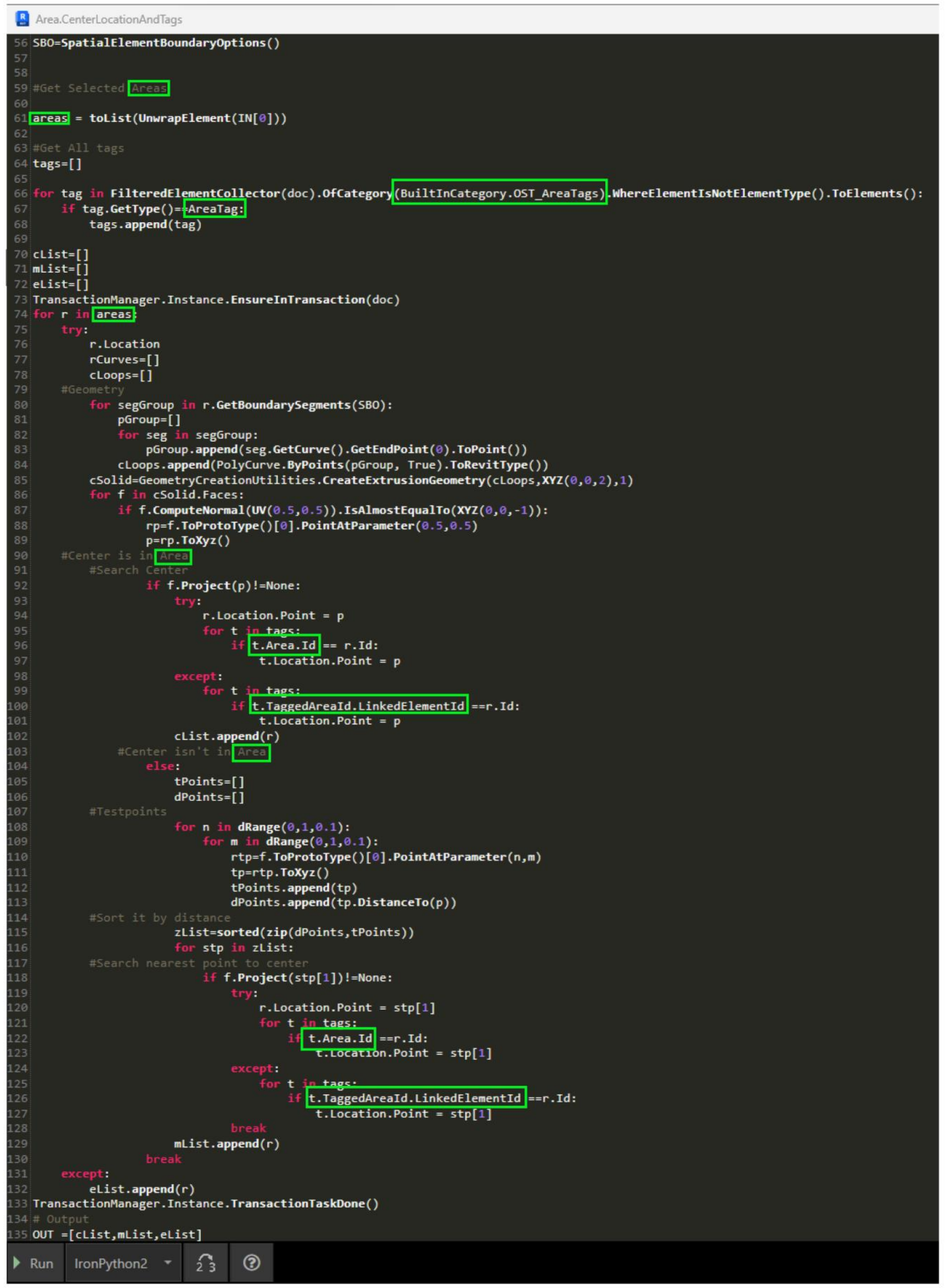

DESIGN STAGE
DESIGN STAGE -PROJECT WORKFLOW
STAGE
STAKEHOLDERS
PROCESSES
DELIVERABLES
DOCUMENTS
MODEL
PRE-DESIGN STAGE
CONCEPT DESIGN
SPATIAL COORDINATION
TECHNICAL DESIGN
CONSTRUCTION STAGE
OWNER/CLIENT
ARCHITECT DESIGNER TEAM LEADER
SUPPLIERS
WORKING GROUP
ARCHITECT DESIGNER TEAM LEADER
STRUCTURAL ENGINEER TEAM LEADER
MEP ENGINEER TEAM LEADER
FACILITY MANAGER
BIM MANAGER
BIM COORDINATOR ARCHITECTS TASK TEAM
BIM COORDINATOR STRUCTURAL ENGINEERS TASK TEAM
BIM COORDINATOR MEP ENGINEERS TASK TEAM
BIM MANAGER
ARCHITECTS TASK TEAM
STRUCTURAL ENGINEERS TASK TEAM
MEP ENGINEERS TASK TEAM
PROJECT REQUERIMENTS
DEFINE THE PRELIMINAR PROJECT
PROVIDE CERTIFIED MODELS OBJECTS
DEVELOP ARCHITECTURAL MODEL
SPATIAL REQUERIMENTS
CREATE, COORDINATE AND SUPERVISE THE WORKING GROUP
NO
VALIDATED?
YES
PRODUCE AND EXTRACT FROM THE MODEL DETAILS AND DOCUMENTS
PRODUCE AND EXTRACT FROM THE MODEL DETAILS AND DOCUMENTS
PRODUCE AND EXTRACT FROM THE MODEL DETAILS AND DOCUMENTS
CREATE, COORDINATE AND SUPERVISE THE WORKING GROUP
MILESTONE 1
MILESTONE 2
MILESTONE 3
SCHEDULES
QUANTITIES
DETAILS
COSTS
SCHEDULES
QUANTITIES
DETAILS
COSTS
SCHEDULES
QUANTITIES
DETAILS
COSTS
CONCEPT MODEL
ARCHITECTURAL MODEL
STRUCTURAL MODEL
MEP MODEL
OBJECTS LIBRARY
FEDERATED MODEL
ARCHITECTURAL MODEL
STRUCTURAL MODEL
MEP MODEL
FEDERATED MODEL FOR CONSTRUCTION

APPENDIX 2: PYTHON CODE USED IN THE SCRIPT OF CENTRALIZING THE AREAS AND THEIR TAGS



Room.CenterLocationAndTags

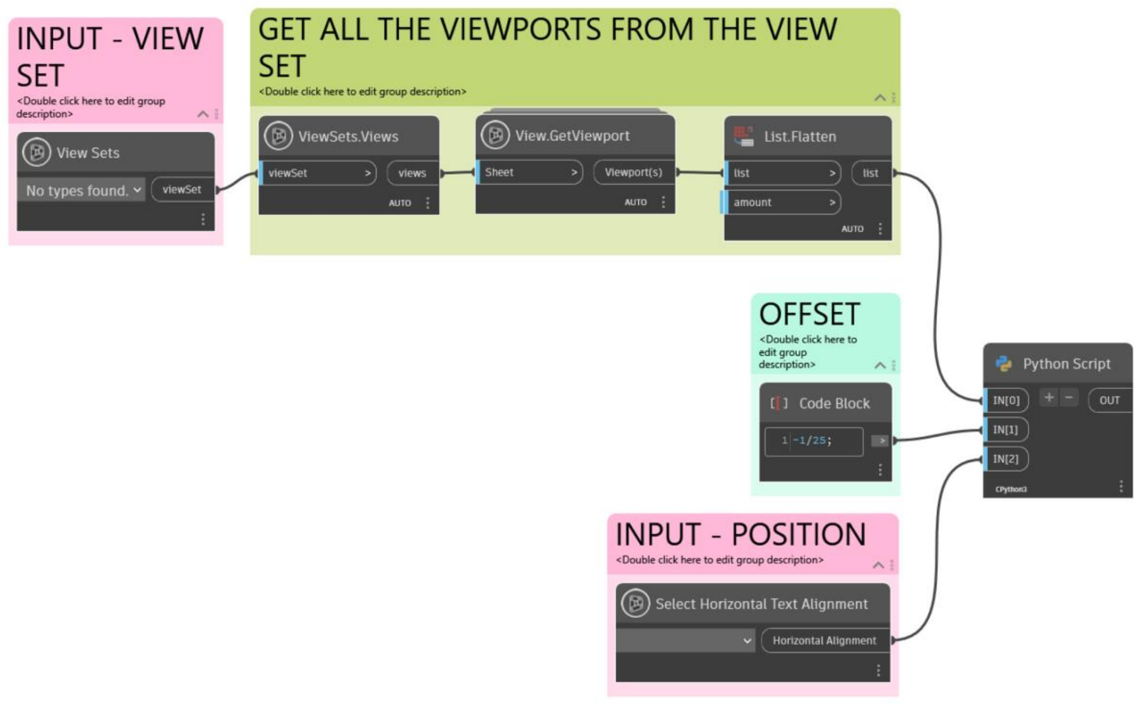
```python
SBO=SpatialElementBoundaryOptions()


#Get Selected Rooms

rooms = toList(UnwrapElement(IN[0]))

#Get All tags
tags=[]

for tag in FilteredElementCollector(doc).OfCategory(BuiltInCategory.OST_RoomTags).WhereElementIsNotElementType().ToElements():
    if tag.GetType()==Architecture.RoomTag:
        tags.append(tag)

cList=[]
mList=[]
eList=[]
TransactionManager.Instance.EnsureInTransaction(doc)
for r in rooms:
    try:
        r.Location
        rCurves=[]
        cLoops=[]
    #Geometry
        for segGroup in r.GetBoundarySegments(SBO):
            pGroup=[]
            for seg in segGroup:
                pGroup.append(seg.GetCurve().GetEndPoint(0).ToPoint())
            cLoops.append(PolyCurve.ByPoints(pGroup, True).ToRevitType())
        cSolid=GeometryCreationUtilities.CreateExtrusionGeometry(cLoops,XYZ(0,0,2),1)
        for f in cSolid.Faces:
            if f.ComputeNormal(UV(0.5,0.5)).IsAlmostEqualTo(XYZ(0,0,-1)):
                rp=f.ToProtoType()[0].PointAtParameter(0.5,0.5)
                p=rp.ToXyz()
    #Center is in Room
        #Search Center
                if f.Project(p)!=None:
                    try:
                        r.Location.Point = p
                        for t in tags:
                            if t.Room.Id == r.Id:
                                t.Location.Point = p
                    except:
                        for t in tags:
                            if t.TaggedRoomId.LinkedElementId ==r.Id:
                                t.Location.Point = p
                    cList.append(r)
            #Center isn't in Room
                else:
                    tPoints=[]
                    dPoints=[]
        #Testpoints
                    for n in dRange(0,1,0.1):
                        for m in dRange(0,1,0.1):
                            rtp=f.ToProtoType()[0].PointAtParameter(n,m)
                            tp=rtp.ToXyz()
                            tPoints.append(tp)
                            dPoints.append(tp.DistanceTo(p))
        #Sort it by distance
                    zList=sorted(zip(dPoints,tPoints))
                    for stp in zList:
        #Search nearest point to center
                        if f.Project(stp[1])!=None:
                            try:
                                r.Location.Point = stp[1]
                                for t in tags:
                                    if t.Room.Id ==r.Id:
                                        t.Location.Point = stp[1]
                            except:
                                for t in tags:
                                    if t.TaggedRoomId.LinkedElementId ==r.Id:
                                        t.Location.Point = stp[1]
                            break
                    mList.append(r)
                break
    except:
        eList.append(r)
TransactionManager.Instance.TransactionTaskDone()
# Output
OUT =[cList,mList,eList]
```

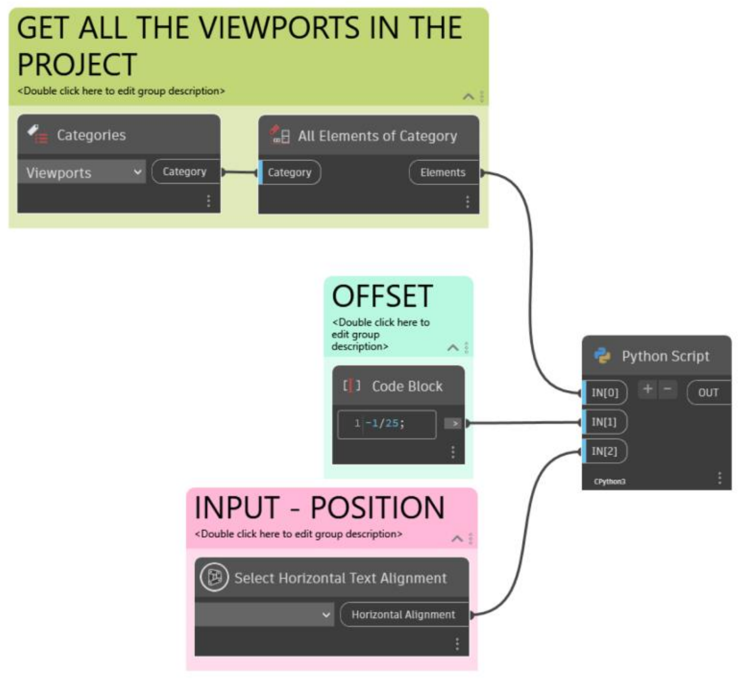


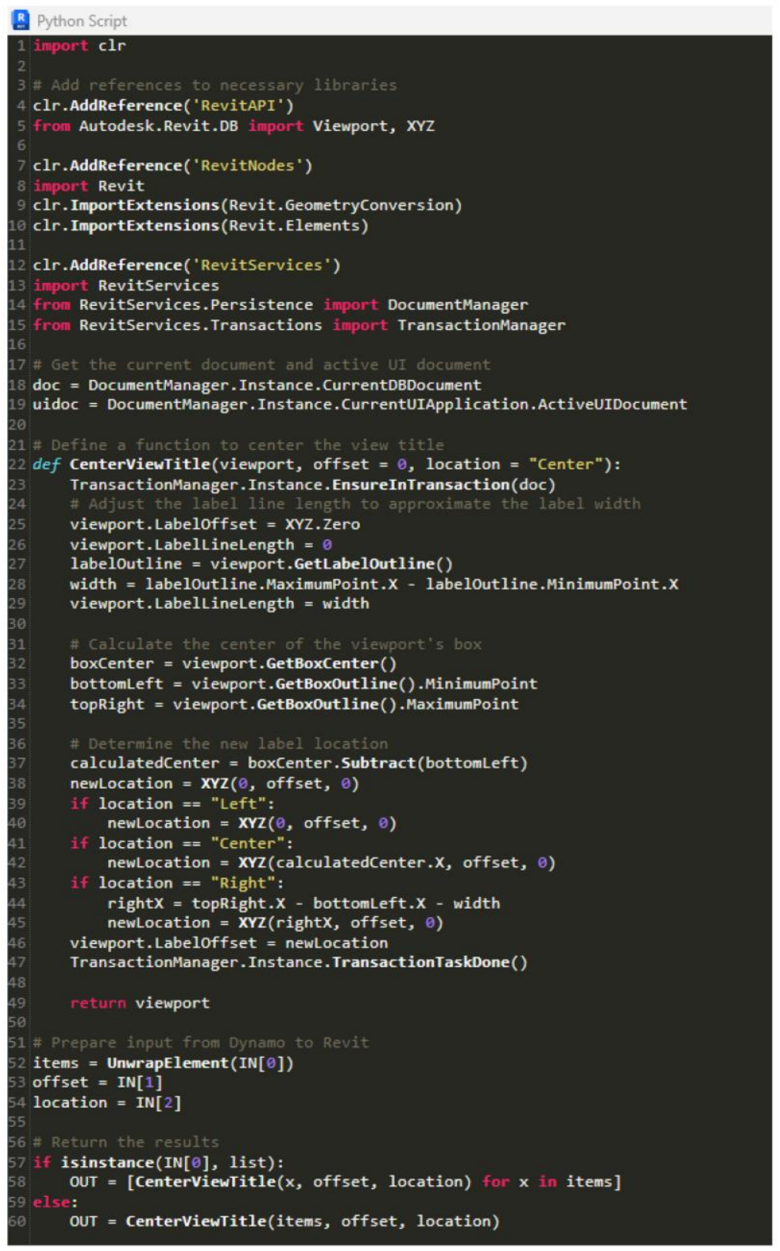
Area.CenterLocationAndTags

```python
SBO=SpatialElementBoundaryOptions()


#Get Selected Areas

areas = toList(UnwrapElement(IN[0]))

#Get All tags
tags=[]

for tag in FilteredElementCollector(doc).OfCategory(BuiltInCategory.OST_AreaTags).WhereElementIsNotElementType().ToElements():
    if tag.GetType()==AreaTag:
        tags.append(tag)

cList=[]
mList=[]
eList=[]
TransactionManager.Instance.EnsureInTransaction(doc)
for r in areas:
    try:
        r.Location
        rCurves=[]
        cLoops=[]
    #Geometry
        for segGroup in r.GetBoundarySegments(SBO):
            pGroup=[]
            for seg in segGroup:
                pGroup.append(seg.GetCurve().GetEndPoint(0).ToPoint())
            cLoops.append(PolyCurve.ByPoints(pGroup, True).ToRevitType())
        cSolid=GeometryCreationUtilities.CreateExtrusionGeometry(cLoops,XYZ(0,0,2),1)
        for f in cSolid.Faces:
            if f.ComputeNormal(UV(0.5,0.5)).IsAlmostEqualTo(XYZ(0,0,-1)):
                rp=f.ToProtoType()[0].PointAtParameter(0.5,0.5)
                p=rp.ToXyz()
    #Center is in Area
        #Search Center
                if f.Project(p)!=None:
                    try:
                        r.Location.Point = p
                        for t in tags:
                            if t.Area.Id == r.Id:
                                t.Location.Point = p
                    except:
                        for t in tags:
                            if t.TaggedAreaId.LinkedElementId ==r.Id:
                                t.Location.Point = p
                    cList.append(r)
            #Center isn't in Area
                else:
                    tPoints=[]
                    dPoints=[]
        #Testpoints
                    for n in dRange(0,1,0.1):
                        for m in dRange(0,1,0.1):
                            rtp=f.ToProtoType()[0].PointAtParameter(n,m)
                            tp=rtp.ToXyz()
                            tPoints.append(tp)
                            dPoints.append(tp.DistanceTo(p))
        #Sort it by distance
                    zList=sorted(zip(dPoints,tPoints))
                    for stp in zList:
        #Search nearest point to center
                        if f.Project(stp[1])!=None:
                            try:
                                r.Location.Point = stp[1]
                                for t in tags:
                                    if t.Area.Id ==r.Id:
                                        t.Location.Point = stp[1]
                            except:
                                for t in tags:
                                    if t.TaggedAreaId.LinkedElementId ==r.Id:
                                        t.Location.Point = stp[1]
                            break
                    mList.append(r)
                break
    except:
        eList.append(r)
TransactionManager.Instance.TransactionTaskDone()
# Output
OUT =[cList,mList,eList]
```

APPENDIX 3: DYNAMO AND PYTHON SCRIPTS FOR ALIGNMENT OF THE VIEW TITLE OF THE VIEWPORT IN A CHOSEN POSITION

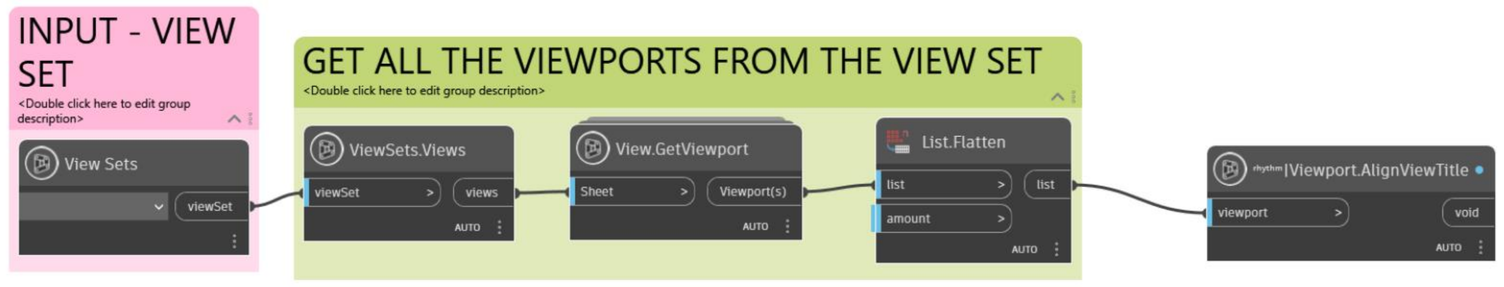

INPUT - VIEW SET
<Double click here to edit group description>
View Sets
No types found.
viewSet
GET ALL THE VIEWPORTS FROM THE VIEW SET
<Double click here to edit group description>
ViewSets.Views
viewSet
views
View.GetViewport
Sheet
Viewport(s)
List.Flatten
list
amount
list
OFFSET
<Double click here to edit group description>
Code Block
-1/25;
INPUT - POSITION
<Double click here to edit group description>
Select Horizontal Text Alignment
Horizontal Alignment
Python Script
IN[0]
IN[1]
IN[2]
OUT
CPython3

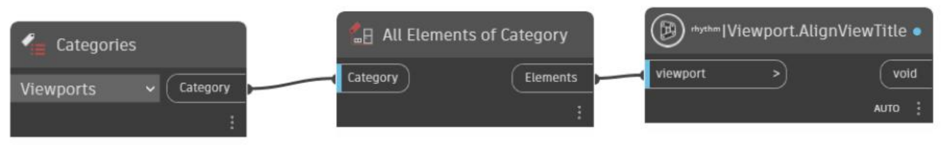

GET ALL THE VIEWPORTS IN THE PROJECT
<Double click here to edit group description>
Categories
Viewports
Category
All Elements of Category
Category
Elements
OFFSET
<Double click here to edit group description>
Code Block
-1/25;
INPUT - POSITION
<Double click here to edit group description>
Select Horizontal Text Alignment
Horizontal Alignment
Python Script
IN[0]
IN[1]
IN[2]
OUT
CPython3


Python Script

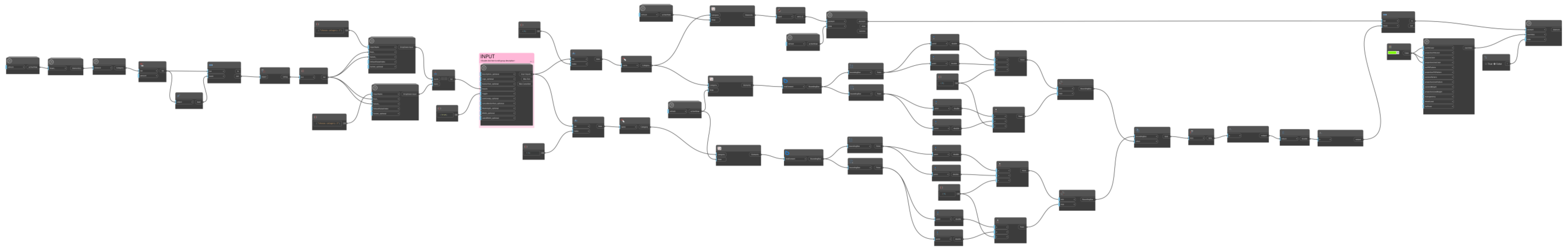
```python
import clr

# Add references to necessary libraries
clr.AddReference('RevitAPI')
from Autodesk.Revit.DB import Viewport, XYZ

clr.AddReference('RevitNodes')
import Revit
clr.ImportExtensions(Revit.GeometryConversion)
clr.ImportExtensions(Revit.Elements)

clr.AddReference('RevitServices')
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager

# Get the current document and active UI document
doc = DocumentManager.Instance.CurrentDBDocument
uidoc = DocumentManager.Instance.CurrentUIApplication.ActiveUIDocument

# Define a function to center the view title
def CenterViewTitle(viewport, offset = 0, location = "Center"):
    TransactionManager.Instance.EnsureInTransaction(doc)
    # Adjust the label line length to approximate the label width
    viewport.LabelOffset = XYZ.Zero
    viewport.LabelLineLength = 0
    labelOutline = viewport.GetLabelOutline()
    width = labelOutline.MaximumPoint.X - labelOutline.MinimumPoint.X
    viewport.LabelLineLength = width

    # Calculate the center of the viewport's box
    boxCenter = viewport.GetBoxCenter()
    bottomLeft = viewport.GetBoxOutline().MinimumPoint
    topRight = viewport.GetBoxOutline().MaximumPoint

    # Determine the new label location
    calculatedCenter = boxCenter.Subtract(bottomLeft)
    newLocation = XYZ(0, offset, 0)
    if location == "Left":
        newLocation = XYZ(0, offset, 0)
    if location == "Center":
        newLocation = XYZ(calculatedCenter.X, offset, 0)
    if location == "Right":
        rightX = topRight.X - bottomLeft.X - width
        newLocation = XYZ(rightX, offset, 0)
    viewport.LabelOffset = newLocation
    TransactionManager.Instance.TransactionTaskDone()

    return viewport

# Prepare input from Dynamo to Revit
items = UnwrapElement(IN[0])
offset = IN[1]
location = IN[2]

# Return the results
if isinstance(IN[0], list):
    OUT = [CenterViewTitle(x, offset, location) for x in items]
else:
    OUT = CenterViewTitle(items, offset, location)
```

APPENDIX 4: DYNAMO SCRIPTS FOR ALIGNMENT OF THE VIEW TITLE WITH THE VIEWPORT

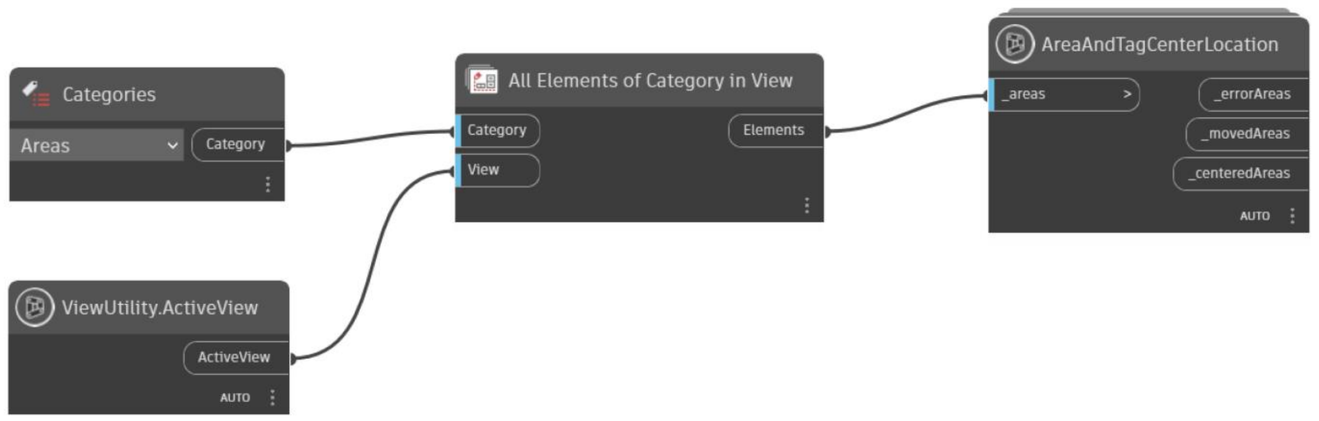

INPUT - VIEW SET
<Double click here to edit group description>
View Sets
viewSet
GET ALL THE VIEWPORTS FROM THE VIEW SET
<Double click here to edit group description>
ViewSets.Views
viewSet
views
View.GetViewport
Sheet
Viewport(s)
List.Flatten
list
amount
list
rhythm | Viewport.AlignViewTitle
viewport
void
AUTO

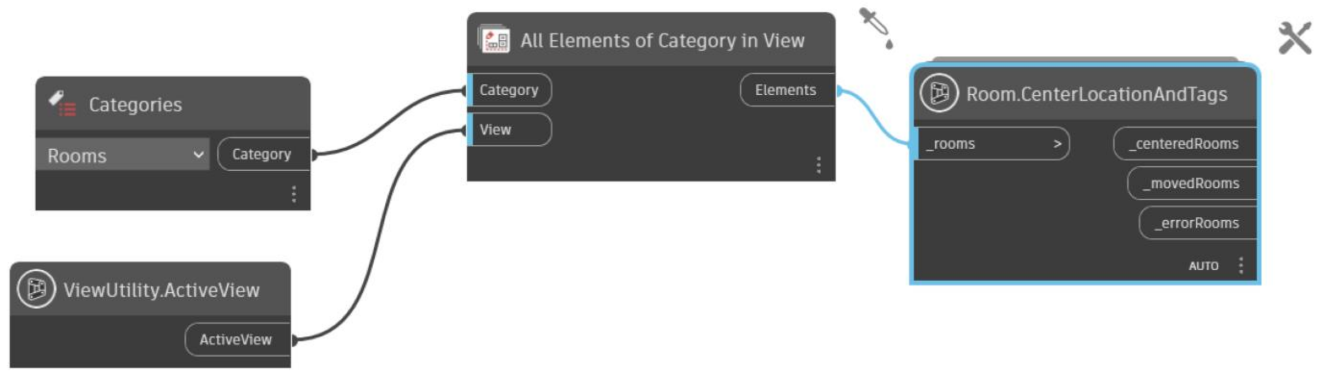


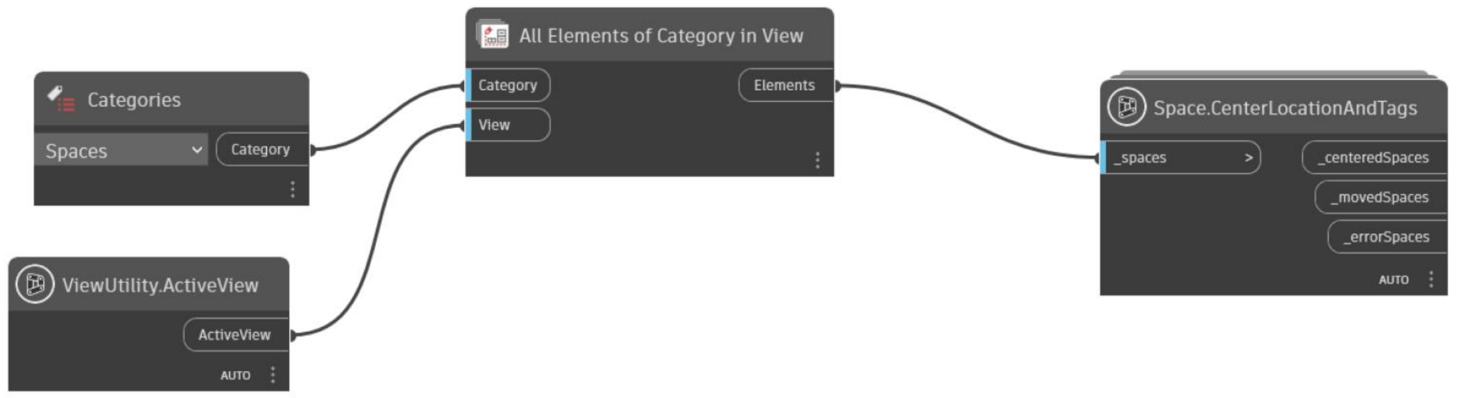
APPENDIX 5: DYNAMO SCRIPT FOR DETECTION OF OVERLAPPING INFORMATION



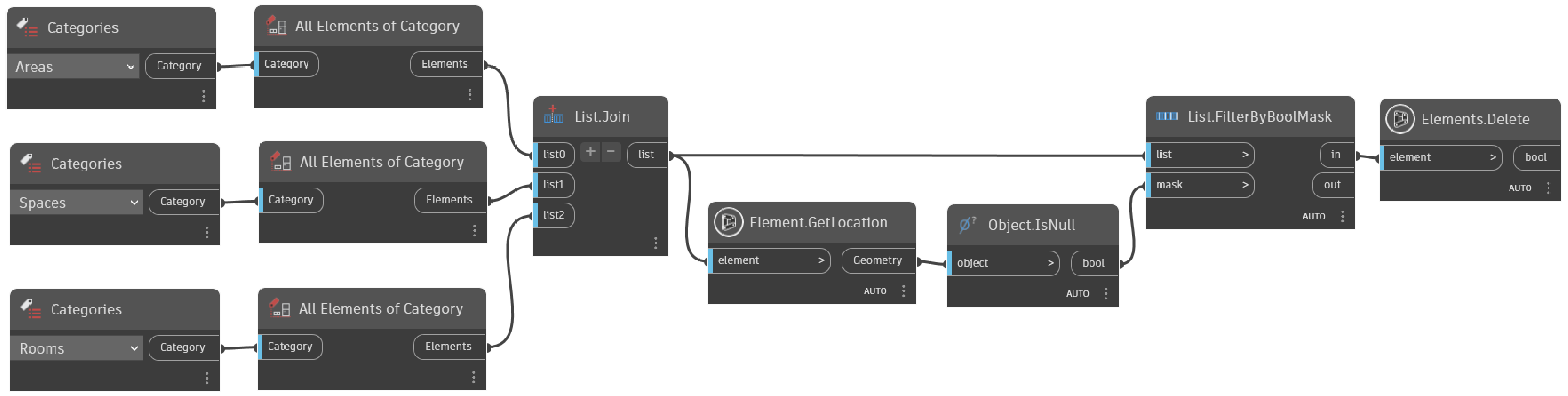
APPENDIX 6: DYNAMO SCRIPT FOR CENTRALIZATION OF ROOMS, AREAS, REVIT SPACES, AND THEIR RESPECTIVE TAGS



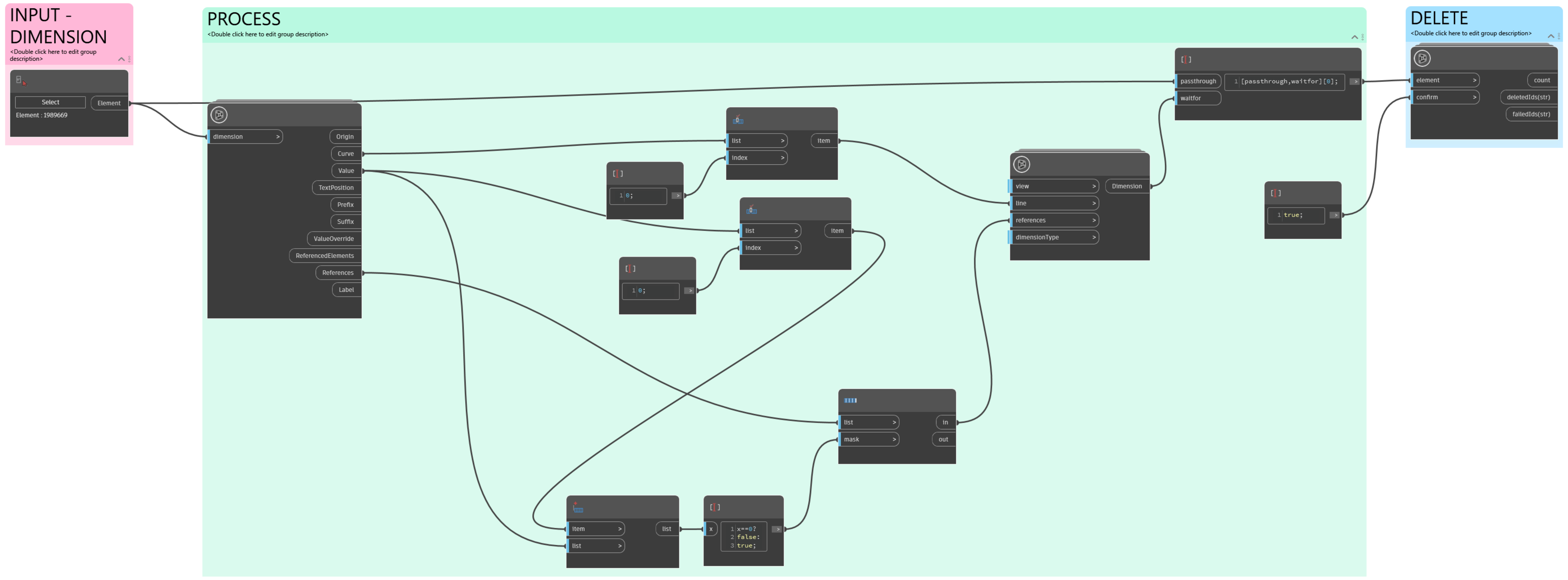




APPENDIX 7: DYNAMO SCRIPT FOR REMOVAL OF NOT PLACED ROOMS, AREAS, AND REVIT SPACES



APPENDIX 8: DYNAMO SCRIPT FOR REMOVAL OF VALUE 0 FROM DIMENSION ANNOTATIONS



```
[passthrough,waitfor][0];
```

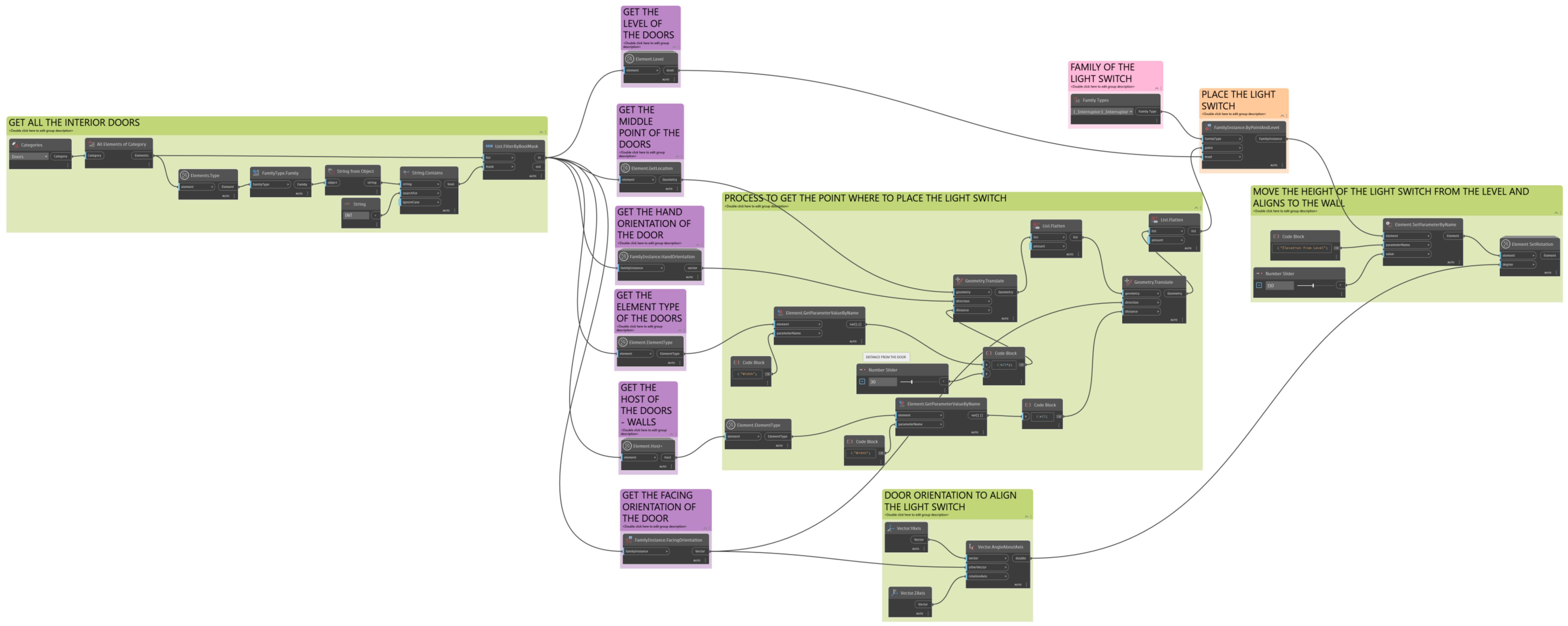
```
0;
```

```
0;
```

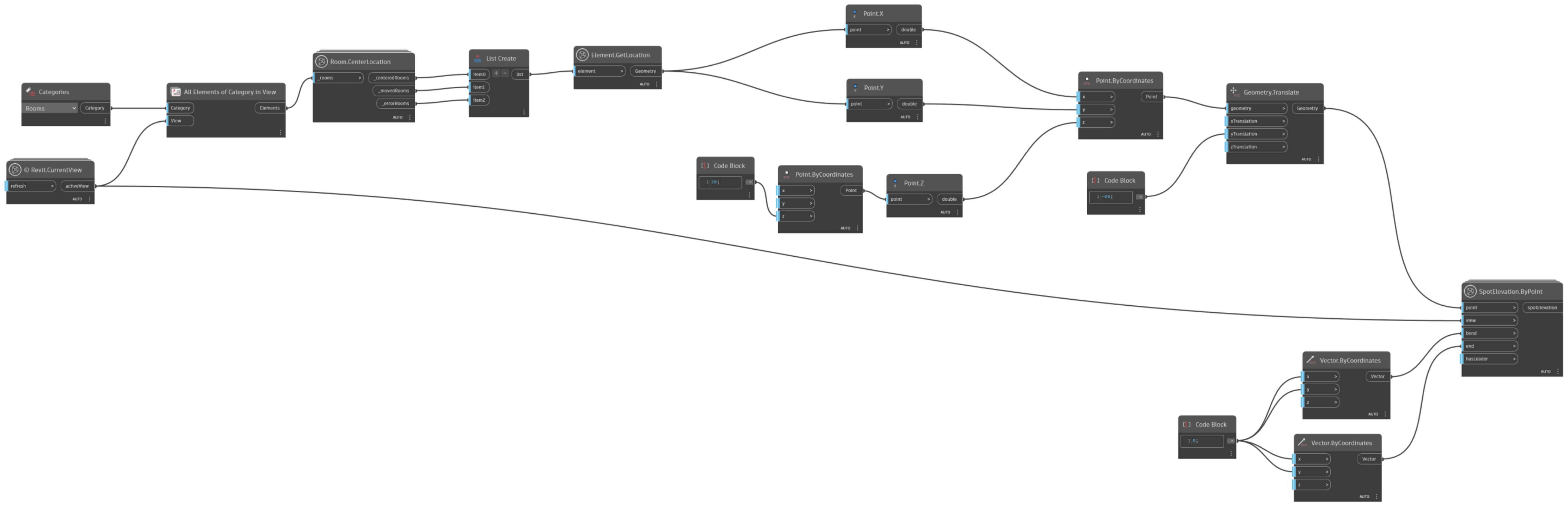
```
x==0?
false:
true;
```

```
true;
```

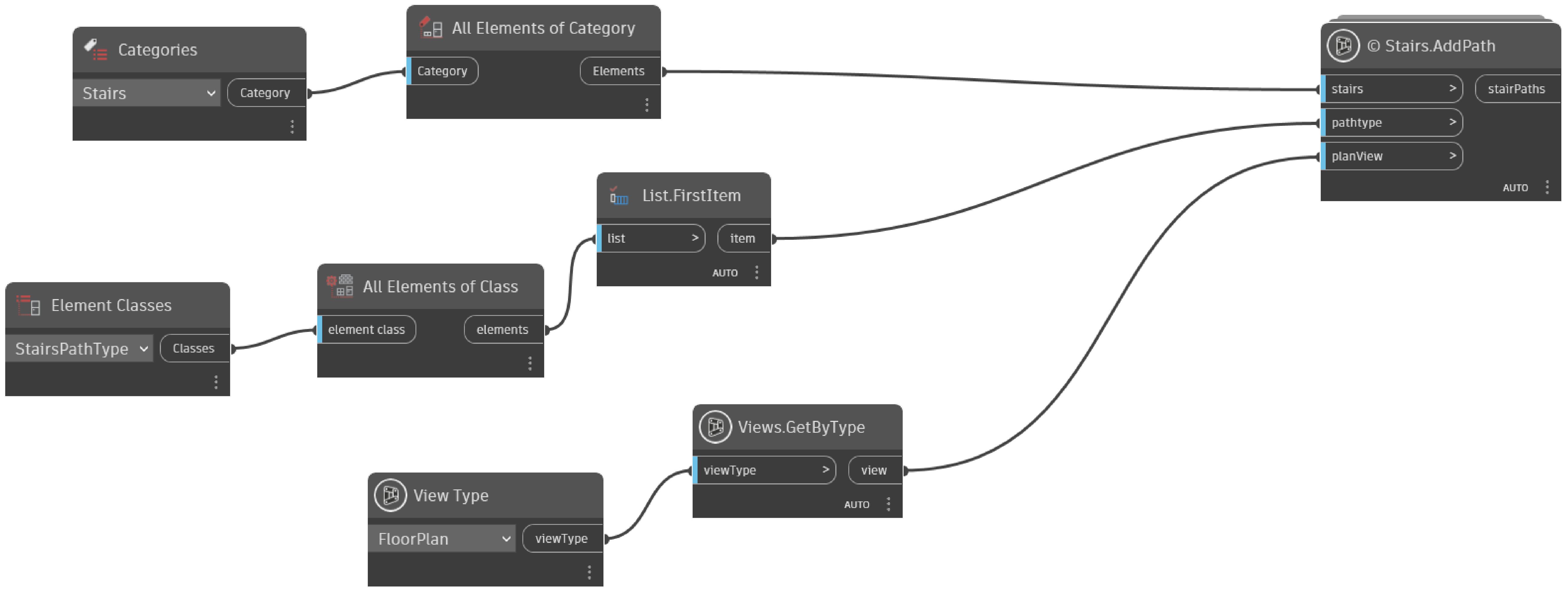
APPENDIX 9: DYNAMO SCRIPT FOR INSERTION OF LIGHT SWITCH REFERENCED TO A DOOR


GET ALL THE INTERIOR DOORS
Categories
Doors
All Elements of Category
Elements.Type
FamilyType.Family
String from Object
String.Contains
String
List.FilterByBoolMask
GET THE LEVEL OF THE DOORS
Element.Level
GET THE MIDDLE POINT OF THE DOORS
Element.GetLocation
GET THE HAND ORIENTATION OF THE DOOR
FamilyInstance.HandOrientation
GET THE ELEMENT TYPE OF THE DOORS
Element.ElementType
GET THE HOST OF THE DOORS - WALLS
Element.Host+
GET THE FACING ORIENTATION OF THE DOOR
FamilyInstance.FacingOrientation
FAMILY OF THE LIGHT SWITCH
Family Types
PLACE THE LIGHT SWITCH
FamilyInstance.ByPointAndLevel
PROCESS TO GET THE POINT WHERE TO PLACE THE LIGHT SWITCH
Element.GetParameterValueByName
Code Block
DISTANCE FROM THE DOOR
Number Slider
Geometry.Translate
List.Flatten
Element.ElementType
DOOR ORIENTATION TO ALIGN THE LIGHT SWITCH
Vector.YAxis
Vector.AngleAboutAxis
Vector.ZAxis
MOVE THE HEIGHT OF THE LIGHT SWITCH FROM THE LEVEL AND ALIGNS TO THE WALL
Code Block
Number Slider
Element.SetParameterByName
Element.SetRotation

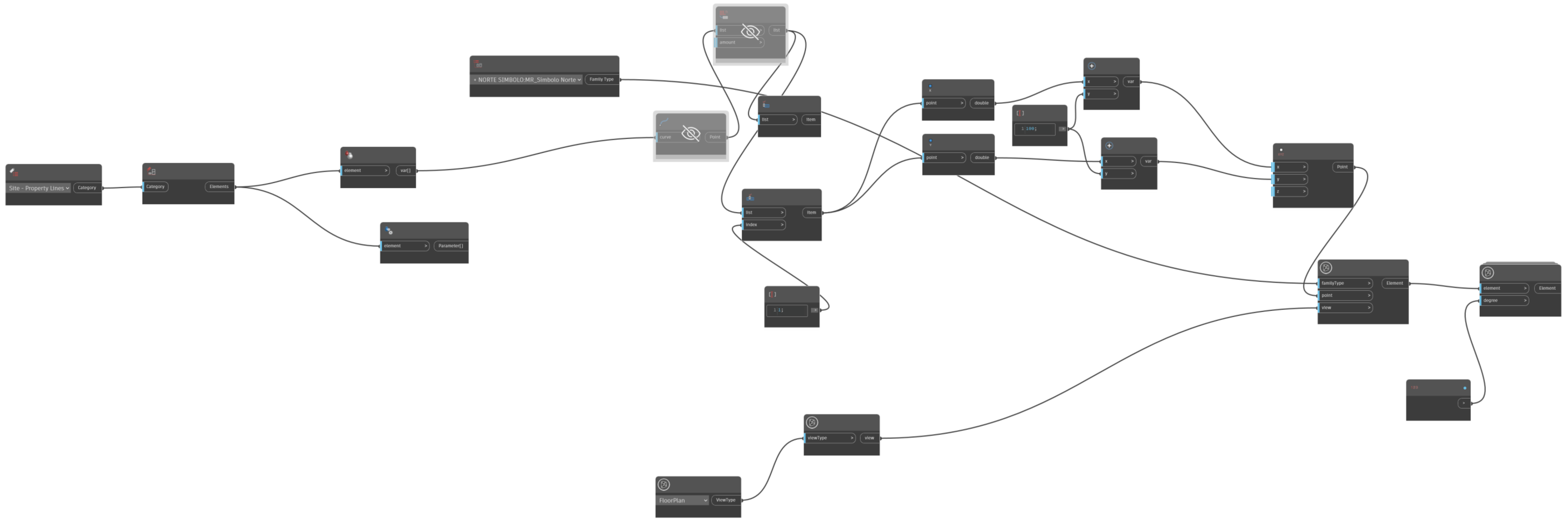
APPENDIX 10: DYNAMO SCRIPT FOR INSERTION OF SPOT ELEVATION IN A VIEW ALIGNED TO THE CENTER OF ALL ROOMS



APPENDIX 11: DYNAMO SCRIPT FOR INSERTION OF STAIR PATH



APPENDIX 12: DYNAMO SCRIPT FOR INSERTION OF NORTH SYMBOL IN ALL FLOOR PLANS


Site - Property Lines
Category
Category
Elements
element
var[]
element
Parameter[]
NORTE SIMBOLO:MR_Simbolo Norte
Family Type
list
amount
list
curve
Point
list
item
list
index
item
1 1;
point
double
point
double
1 100;
x
y
var
x
y
var
x
y
z
Point
familyType
point
view
Element
element
degree
Element
viewType
view
FloorPlan
ViewType