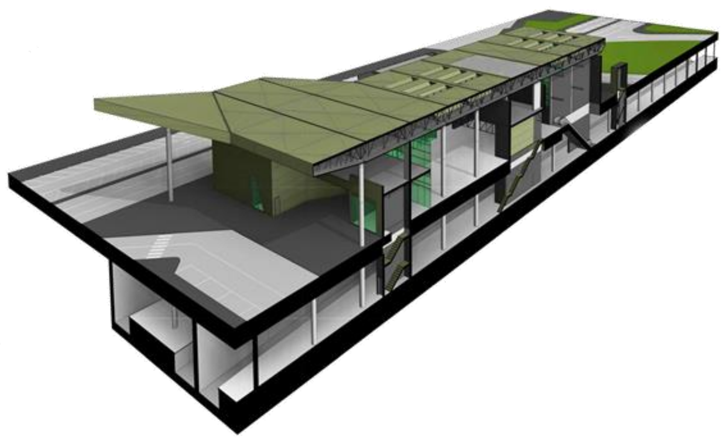
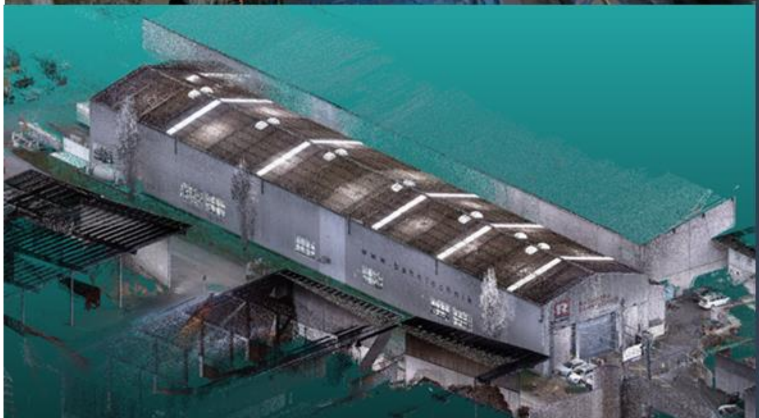




COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu

D5.5 – BIM-based Standard test Methods for Geometric Quality Control v1

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310

D5.5 – BIM-based Standard test Methods for Geometric Quality Control v1

Dissemination Level:	Public
Deliverable Type:	Report
Lead Partner:	UEDIN
Contributing Partners:	UEDIN, CERTH
Due date:	31-05-2022
Actual submission date:	31-05-2022

Authors

Name	Beneficiary	Email
Martín Bueno	UEDIN	martin.bueno@ed.ac.uk
Frédéric Bosché	UEDIN	f.bosche@ed.ac.uk
Thanos Tsakiris	CERTH	atsakir@iti.gr

Reviewers

Name	Beneficiary	Email
Giorgos Giannakis	HYPERTECH	g.giannakis@hypertech.gr
Damiano Falcioni	BOC-AG	damiano.falcioni@boc-group.com

Version History

Version	Editors	Date	Comment
0.1	Martín Bueno	23.03.2022	Table of Content
0.2	Martín Bueno	04.05.2022	Document first draft
0.3	Frédéric Bosché	04.05.2022	Document second draft
0.4	Martín Bueno	09.05.2022	Document third draft
0.5	Martín Bueno	18.05.2022	First reviewed draft
0.9	Frédéric Bosché	18.05.2022	Final review
1.0	Martín Bueno, Giorgos Giannakis	31.05.2022	Submission to the EC portal

Disclaimer

Executive Summary

The COGITO Deliverable D5.5 “BIM-based Standard test Methods for Geometric Quality Control v1” documents the COGITO Geometric Quality Control solution and presents the first iteration of the development activities in T5.3 “BIM-based Standard Test Methods for Geometric Quality Control”. Overall, the Geometric Quality Control solution aims to perform detailed geometric and dimensional tolerance checking automatically by analysing the as-built data (laser scanned point clouds) and as-planned data (project BIM model). The output results contain detail information of each rule used, its tolerances and its as-built scalar results obtained from all the instances of their applications on the given project.

The Geometric Quality Control solution, called GeometricQC tool, is a set of modular open-source components and sub-components. It is delivered as a service that can run from the command line on any operating system or as a webservice. The GeometricQC tool works in two different phases. The first one is the offline pre-processing phase (in charge of the GeometricQC offline processing component) where it extracts all the relevant information from the as-designed data to define the set of geometric QC tolerance checks that are to be conducted during the project’s construction phase. The second phase is the active geometric tolerance verification (performed by the GeometricQC active geometric quality control analysis component) where all the as-built data are processed and analysed to obtain detailed results for each of the geometric quality control instances that were detected in the pre-processing phase.

The present documentation of the Geometric Quality Control solution, along with its components, is oriented towards the functionalities they broadly deliver, the technology stack they build upon, the inputs, outputs and APIs they expose, the installation instructions, the assumptions and restrictions, the applications examples, the development and integration status, and the requirements coverage. In this first release, the COGITO GeometricQC tool implements a set of the envisaged functionalities (i.e. IFC file interpreter, automatic generation of QC instances, active as-built dimensional analysis) and its usage is illustrated and evaluated based on artificial case examples, obtained during the development of the components.

Table of contents

List of Figures

List of Tables

List of Acronyms

Term	Description
API	Application Programming Interface
ARC	Automated Rule Checking
BIM	Building Information Modelling
B-Rep	Boundary Representation
CAD	Computer-Aided Design
COGITO	Construction Phase diGItal Twin mOdel
CSV	Comma-Separated Value
DCC	Digital Command Center
DTP	Digital Twin Platform
ID	Identifier
IFC	Industry Foundation Classes
OBJ	Object file
QC	Quality control
STL	Standard Triangle/Tessellation Language
TLS	Terrestrial Laser Scanning
UID	Unique Identifier
WP	Work Package

1 Introduction

1.1 Scope and Objectives of the Deliverable

This deliverable reports on the work conducted between M12 and M19 on the Geometric Quality Control (QC) tool being developed as part of T5.3. BIM-based Standard Test Methods for Geometric Quality Control. The GeometricQC tool can be seen as the main contribution outcome of the combination of T5.1 and T5.3 for the Geometric QC process proposed by the COGITO consortium. The scope of the GeometricQC tool is to perform automatic QC compliance, and report its results back to the relevant stakeholders, such as the quality managers and the project managers. Having the QC results at hand (and visualising them using the DigitAR and Digital Command Centre (DCC) tools being developed in Tasks T5.4 and T7.3-4 respectively), these stakeholders can decide on whether the already built elements require any remedial works.

More specifically, this deliverable reports on the development of the first release of the GeometricQC tool solution, focusing on its two main components listed below:

- **GeometricQC tool offline processing component:** this component performs a set of fully automated pre-processing steps to extract all the relevant project information in order to obtain the list of geometric QC instances that need to be conducted throughout the project. The processing steps mainly involve obtaining the list of elements in the original BIM file, compute their geometric relationships and cross-reference them with the QC specifications from standards.
- **GeometricQC tool active geometric quality control analysis component:** this component is in charge of the main computation work of the GeometricQC tool by integrating most of the other sub-components and performing the QC tasks for each of the structural elements in the BIM model when the project is being constructed.

These two components provide the core functionalities of the GeometricQC tool, and also integrate the sub-components introduced in "*D5.1: Innovative Scan-vs-BIM- based Geometric QC component v1*".

1.2 Relation to other Tasks and Deliverables

This deliverable reports the first version (v1) of the GeometricQC tool. A second version (v2) of the GeometricQC tool will be reported at the end of M24 in *"D5.6: BIM-based Standard test Methods for Geometric Quality Control v2"* including new functionalities and improvements to the components presented in this deliverable.

1.2.1 Relation to other COGITO tools and components

The GeometricQC tool serves as a data consumer of the **Scan-vs-BIM solution** and as both a consumer and provider of the **Digital Twin Platform**. In alignment with the COGITO system architecture, it does not interact with other tools and components within the COGITO solution.

1.2.2 Relation to other Deliverables

The GeometricQC tool presented in this deliverable, and all the mentioned components, is being prototyped based on the User Requirements defined in *"D2.1: Stakeholder requirements for the COGITO system"*, as well as the functional and non-functional requirements introduced in *"D2.4: COGITO system architecture v1"* and *"D2.5: COGITO system architecture v2"*, the ontologies and data models from *"D3.2: COGITO Data Model and Ontology Definition and Interoperability Design"* and *"D3.3: COGITO Data Model and Ontology Definition and Interoperability Design v2"*, and the components, sub-components and tools described in *"D5.1: Innovative Scan-vs-BIM-based Geometric QC component v1"* and *"D7.3: Extraction, Transformation & Loading Tools and Model Checking v2"*.

The GeometricQC tool is principally used to support the Use Case 2.1 (UC2.1).

1.3 Structure of the Deliverable

The deliverable is organised as follows: Section 2 presents an overview of the GeometricQC tool, placing the main components and sub-components in the overall context of the solution and the processing pipeline. Sections 3 and 4 describe the *GeometricQC tool offline processing* and *GeometricQC tool active geometric quality control analysis* components, including the technology stack, implementation tools, the input/output data, and API documentation, application examples, licensing, installation instructions, development and integration status, requirements coverage, and assumptions and restrictions of each component. Section 5 concludes the deliverable.

2 Overview of the Geometric Quality Control tool

The aim of the proposed geometric quality control tool (GeometricQC tool from now on) is to perform highly automated geometric QC during the execution phase by following standard test methods that can be found in standards and regulations or be project specific. The GeometricQC tool is designed to conduct automatic geometric QC by comparing the as-designed and as-built geometry. The as-designed information is obtained from the original BIM model, while the as-built data is captured by terrestrial laser scanning (TLS) and stored in the form of point clouds.

The GeometricQC tool was defined in "*D2.1: Stakeholder requirements for the COGITO system*", "*D2.4: COGITO system architecture v1*" and "*D2.5: COGITO system architecture v2*", and as such, it was designed to be a back-end service application composed of a set of object oriented classes, like the ones previously defined in the Scan-vs-BIM solution in "*D5.1: Innovative Scan-vs-BIM-based Geometric QC component v1*". As already stablished in the COGITO system architecture, the GeometricQC tool aims to use open formats (such as IFC, OBJ, STL, etc. for the as-designed data, and PLY, E57, PTS, etc. for the as-built data), targeting to increased interoperability, accessibility, and data sharing among third parties.

The core idea of the GeometricQC tool is akin to follow an Automated Rule Checking (ARC) approach [1] [2]. The GeometricQC tool is divided in two (2) main sub-components:

- **GeometricQC tool offline processing:** this component is in charge of extracting all the structural elements from the as-designed data, obtain the geometric relationships between them, and cross reference these with the QC dictionary to obtain the full set of QC tasks that need to be conducted throughout the project. This step is only executed the first time the project is created since all this information should be immutable along the duration of the project.
- **GeometricQC tool active geometric quality control analysis:** this component can be seen as the active QC tool, since it is the one that performs the geometric quality control and generates the QC results that are stored in the DTP for sharing with the relevant stakeholders. For its execution, it requires that the elements involved in a given QC instance are already built and scanned by the surveyor, and the as-built data is geo-referenced (or, more specifically, the point cloud data is referenced in the same coordinate system as the project BIM model). At this point, the component automatically matches the as-built data to the 3D geometry of the elements in the BIM model (as described in "*D5.1: Innovative Scan-vs-BIM-based Geometric QC component v1*") and performs the scheduled geometric control check defined by the offline processing component.

Figure 1 presents a high-level diagram of the tool, highlighting the abovementioned components (and phases) that are thoroughly presented in Sections 3 and 4, respectively.

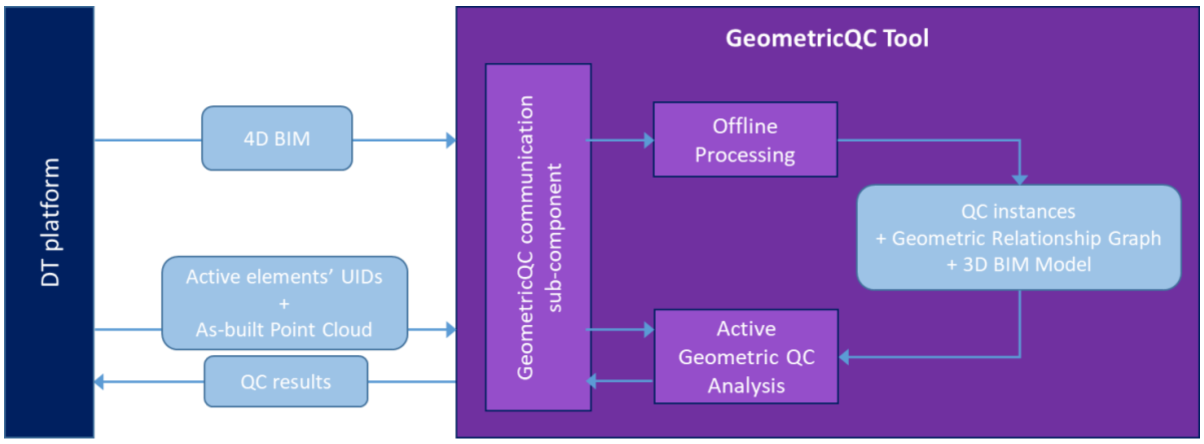

DT platform
4D BIM
Active elements' UIDs + As-built Point Cloud
QC results
GeometricQC Tool
GeometricQC communication sub-component
Offline Processing
QC instances + Geometric Relationship Graph + 3D BIM Model
Active Geometric QC Analysis

Figure 1: GeometricQC Tool workflow

3 GeometricQC tool offline processing component

In this section, we describe the *GeometricQC tool offline processing* component. This component is the one in charge of doing all the pre-processing of the project as-designed data during the planning phase (hence it is done offline, i.e. during the planning phase, only once per project). At the start of every project, several things need to be defined. Within the GeometricQC tool, all the geometric QC specifications and tolerances have to be identified and translated to digital rules that will be verified throughout the project. For this, we require to process the original BIM file and its schedule (i.e. 4D model) to extract the different structural elements, compute their geometric relationships according to the ones identified in standards ([3] in case of concrete elements, and [4] for steel ones), and obtain the list of all the dimensions (or geometry) and tolerance that needs to be checked for each of them, over time throughout the construction.

The end result of the offline processing during the planning phase is the detailed list of these geometric QC instances, the specific elements they involve, and the date when they shall be conducted. Since it is an offline planning step based on the agreed design, the list is immutable, and can be stored. The following sub-sections describe in more detail the different sub-components involved in the offline processing and its technical aspects and requirements.

3.1 Prototype Overview

The objective of the *GeometricQC offline processing* component is to enable the GeometricQC tool to load, interpret and manage the construction's project as-planned data, and generate the list of geometric QC instances that need to be performed during the construction phase.

The *GeometricQC offline processing* component is quite simple to use and include in any software project. The *GeometricQC offline processing* component only requires the file path of the IFC file, the schedule activity relating to each element, and the output file where all the generated information is stored, to avoid repetition of its execution in future instalments. As it can be seen in Figure 2, the component is composed of different classes acting as sub-components. Each of them is in charge of obtaining/processing some kind of information, and interconnected, generate the list of geometric QC instances. The different sub-components and their functionalities are analysed below.

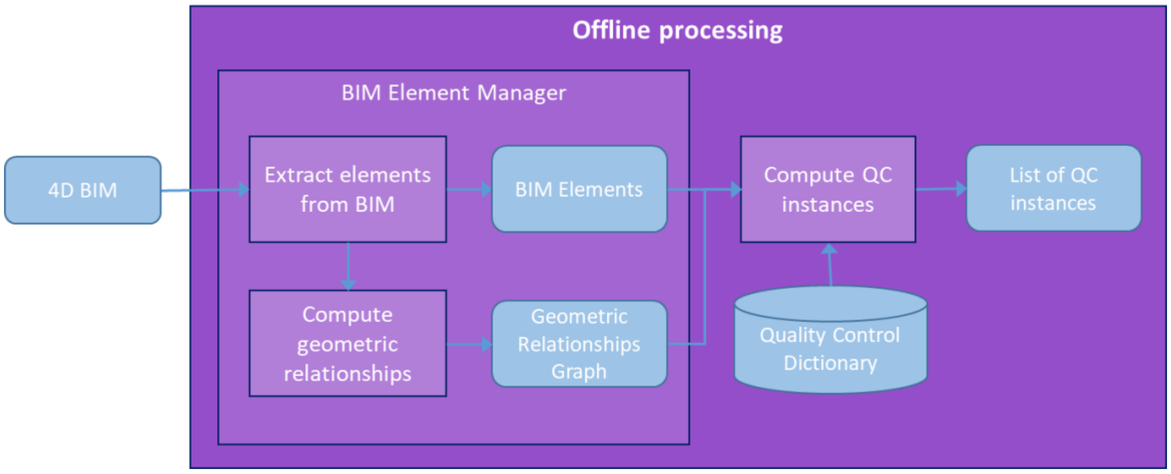

Offline processing
BIM Element Manager
4D BIM
Extract elements from BIM
BIM Elements
Compute geometric relationships
Geometric Relationships Graph
Compute QC instances
List of QC instances
Quality Control Dictionary

Figure 2: GeometricQC offline processing detailed workflow

3.1.1 BIM Element Manager sub-component

As already explained in D5.1, the main objective of the *BIM Element Manager* sub-component is to allow the GeometricQC tool to load, interpret, and manage the construction project as-planned data. To avoid replication of information that has already been provided about this sub-component, we infer the interested reader in D5.1. Herein, we focus on reporting updates to the BIM Element Manager since D5.1 submission. Thus the additions to the current version of the *BIM Element Manager* sub-component concern the following additional features:

- **Obtain the structural element material from the IFC file:** the **IFCManager** class now includes a method to extract the material information of each IFC structural element of interest and assign it to the **BIMElement** object. This material information is important for obtaining the different geometric QC tolerances checks corresponding to each material type.
- **Obtain and store the geometric relationship graph:** thanks to the **RelGraph** class (see 3.1.3 for more detail), the **BIMElementManager** is capable of storing each structural element as a node and the geometric relationship between them as the edge in a graph structure. This graph structure is useful to perform queries of all the related elements and cross-reference with the *Quality Control dictionary* (Section 3.1.2).

Figure 3 illustrates the updated *BIM Element Manager* sub-component and the different interconnected classes that guarantee its proper operation.

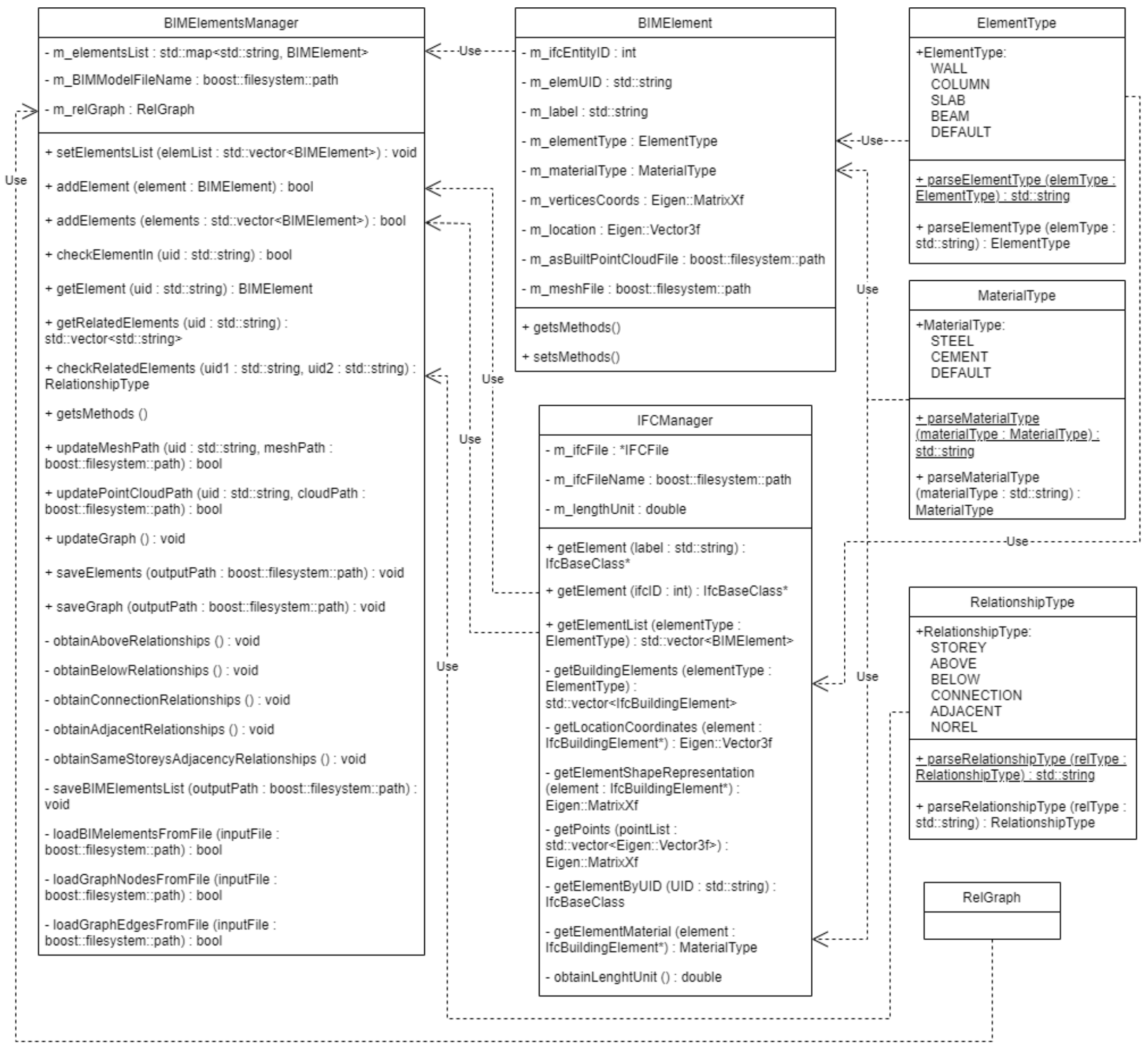

BIMElementsManager
- m_elementsList : std::map<std::string, BIMElement>
- m_BIMModelFileName : boost::filesystem::path
- m_relGraph : RelGraph
+ setElementsList (elemList : std::vector<BIMElement>) : void
+ addElement (element : BIMElement) : bool
+ addElements (elements : std::vector<BIMElement>) : bool
+ checkElementIn (uid : std::string) : bool
+ getElement (uid : std::string) : BIMElement
+ getRelatedElements (uid : std::string) : std::vector<std::string>
+ checkRelatedElements (uid1 : std::string, uid2 : std::string) : RelationshipType
+ getsMethods ()
+ updateMeshPath (uid : std::string, meshPath : boost::filesystem::path) : bool
+ updatePointCloudPath (uid : std::string, cloudPath : boost::filesystem::path) : bool
+ updateGraph () : void
+ saveElements (outputPath : boost::filesystem::path) : void
+ saveGraph (outputPath : boost::filesystem::path) : void
- obtainAboveRelationships () : void
- obtainBelowRelationships () : void
- obtainConnectionRelationships () : void
- obtainAdjacentRelationships () : void
- obtainSameStoreysAdjacencyRelationships () : void
- saveBIMElementsList (outputPath : boost::filesystem::path) : void
- loadBIMelementsFromFile (inputFile : boost::filesystem::path) : bool
- loadGraphNodesFromFile (inputFile : boost::filesystem::path) : bool
- loadGraphEdgesFromFile (inputFile : boost::filesystem::path) : bool
BIMElement
- m_ifcEntityID : int
- m_elemUID : std::string
- m_label : std::string
- m_elementType : ElementType
- m_materialType : MaterialType
- m_verticesCoords : Eigen::MatrixXf
- m_location : Eigen::Vector3f
- m_asBuiltPointCloudFile : boost::filesystem::path
- m_meshFile : boost::filesystem::path
+ getsMethods()
+ setsMethods()
ElementType
+ElementType:
WALL
COLUMN
SLAB
BEAM
DEFAULT
+ parseElementType (elemType : ElementType) : std::string
+ parseElementType (elemType : std::string) : ElementType
MaterialType
+MaterialType:
STEEL
CEMENT
DEFAULT
+ parseMaterialType (materialType : MaterialType) : std::string
+ parseMaterialType (materialType : std::string) : MaterialType
IFCManager
- m_ifcFile : *IFCFile
- m_ifcFileName : boost::filesystem::path
- m_lengthUnit : double
+ getElement (label : std::string) : IfcBaseClass*
+ getElement (ifcID : int) : IfcBaseClass*
+ getElementList (elementType : ElementType) : std::vector<BIMElement>
- getBuildingElements (elementType : ElementType) : std::vector<IfcBuildingElement>
- getLocationCoordinates (element : IfcBuildingElement*) : Eigen::Vector3f
- getElementShapeRepresentation (element : IfcBuildingElement*) : Eigen::MatrixXf
- getPoints (pointList : std::vector<Eigen::Vector3f>) : Eigen::MatrixXf
- getElementByUID (UID : std::string) : IfcBaseClass
- getElementMaterial (element : IfcBuildingElement*) : MaterialType
- obtainLenghtUnit () : double
RelationshipType
+RelationshipType:
STOREY
ABOVE
BELOW
CONNECTION
ADJACENT
NOREL
+ parseRelationshipType (relType : RelationshipType) : std::string
+ parseRelationshipType (relType : std::string) : RelationshipType
RelGraph
Use

Figure 3: BIM Element Manager sub-component classes and its relationships

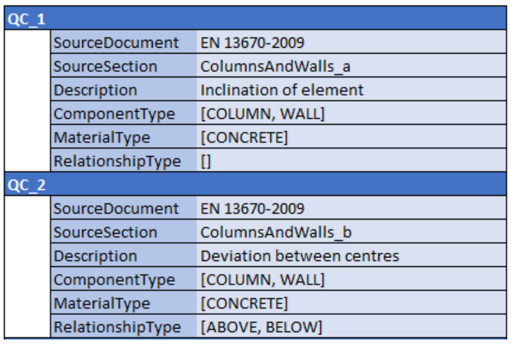
3.1.2 Quality Control Dictionary

The *Quality Control Dictionary* is the sub-component that allows the user to establish and define the different QC instances and tolerances that are required for the different structural elements. In other words, the QC dictionary encompasses all the geometric QC "rules" that need to be applied during the construction phase. Each rule is designed to define the generic context where it should be applied. For example, the structures are defined by the type of elements involved (column, beam, slab, etc), their material type (steel or concrete), and their geometric relationship (parallel, connected, physical connection, etc). Every time the the defined context is encountered in a given input 3D BIM model, the corresponding rule must be applied, i.e. an instance of that rule is created involving the specific element(s).

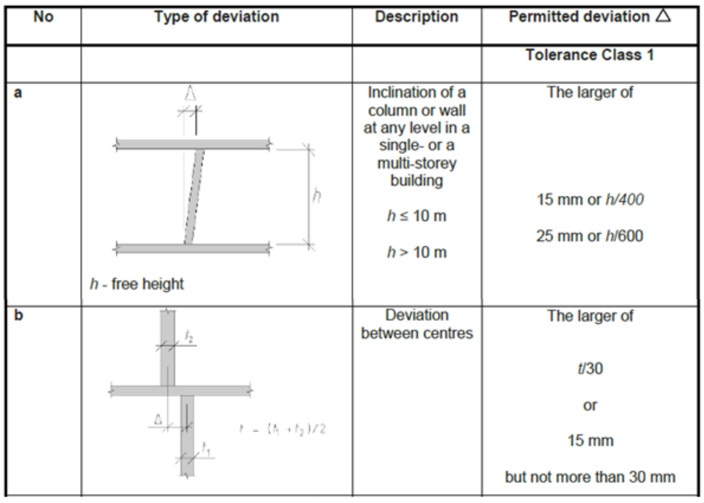
In order for the QC Dictionary to be more descriptive and user friendly, each implemented rule contains additional information, fall under two main groups:

- **Rule description:**
  - *Source document:* regulation or standard number, or specific ID to identify the document where the rule is found.
  - *Source section:* an ID to identify the rule within the source document.
  - *Description:* a brief description of the rule, extracted from examples of the original document.
- **Rule context:**
  - *Element type:* the type of element that the rule is applied to (i.e. columns, beams, etc).
  - *Material type:* the material type of the structural element that the rule is applied to (i.e. steel, concrete).
  - *Relationship type:* the geometric relationship the involved structural elements need to have for this rule (i.e. below, above, etc). In case the rule only involves a single element, this field is left empty.

Figure 4 shows an example of two different dictionary entries with their respective standard specifications that can be found in EN 13670:2009 [3].



QC_1		
	SourceDocument	EN 13670-2009
	SourceSection	ColumnsAndWalls_a
	Description	Inclination of element
	ComponentType	[COLUMN, WALL]
	MaterialType	[CONCRETE]
	RelationshipType	[]
QC_2		
	SourceDocument	EN 13670-2009
	SourceSection	ColumnsAndWalls_b
	Description	Deviation between centres
	ComponentType	[COLUMN, WALL]
	MaterialType	[CONCRETE]
	RelationshipType	[ABOVE, BELOW]



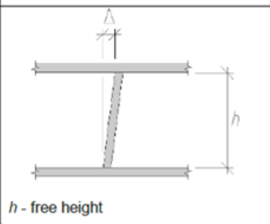
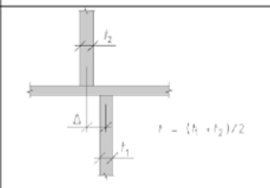
No	Type of deviation	Description	Permitted deviation Δ
			Tolerance Class 1
a		Inclination of a column or wall at any level in a single- or a multi-storey building $h \leq 10$ m $h > 10$ m	The larger of 15 mm or $h/400$ 25 mm or $h/600$
b		Deviation between centres	The larger of $t/30$ or 15 mm but not more than 30 mm

Figure 4: (left) *Quality Control dictionary* entries example. (right) EN 13670:2009 references for the dictionary entries

Currently, 15 rules have been identified and extracted from EN 13670-2009 [3] and encoded in the Quality Control Dictionary:

- **QC_1: Inclination of a column/wall** *(10.4 Columns and Walls, No a)*
- **QC_2: Deviation between centres** *(10.4 Columns and Walls, No b)*
- **QC_3: Curvature of a column/wall between adjacent storey levels** *(10.4 Columns and Walls, No c)*

- **QC_4: Location of a column/wall at any storey level with respect to base level** *(10.4 Columns and Walls, No d)*
- **QC_5: Location of a beam-to-column connection measured relative to the column** *(10.5 Beams and Slabs, No a)*
- **QC_6: Position of bearing axis of support** *(10.5 Beams and Slabs, No b)*
- **QC_7: Cross-sectional dimensions** *(10.6 Sections, No a)*
- **QC_8: Lap-joints** *(10.6 Sections, No c)*
- **QC_9: Free space between adjacent columns/walls** *(Annex G – G.10.4 Columns and Walls, No c)*
- **QC_10: Horizontal straightness of beams** *(Annex G – G.10.5 Beams and Slabs, No a)*
- **QC_11: Distance between adjacent beams** *(Annex G – G.10.5 Beams and Slabs, No b)*
- **QC_12: Inclination of a beam/slab** *(Annex G – G.10.5 Beams and Slabs, No c)*
- **QC_13: Level of adjacent beams** *(Annex G – G.10.5 Beams and Slabs, No d)*
- **QC_14: Level of adjacent floors at supports** *(Annex G – G.10.5 Beams and Slabs, No e)*
- **QC_15: Orthogonality of a cross-section** *(Annex G – G.10.6 Sections, No a)*

The QC Dictionary is stored internally in the GeometricQC tool as a file using a JSON format. As can be seen in Figure 5, the Geometric QC Dictionary sub-component includes two main object classes that are interconnected:

- **GeomQC class:** this class is used to describe the quality control instances that are obtained as described in Section 3.1.4. The class contains all the information stored in the dictionary's entries and extra information relative to the execution of the QC verification, such as the schedule timestamp, the timestamp when the rule was verified, the result, used tolerance values and scalar result values, as optional output file paths. For the QC Dictionary purposes only the relevant information stored in the dictionary is populated in the fields and stored in the dictionary.
- **GeomQCDictionary class:** this class contains the different QC Dictionary entries stored in a map structure to facilitate the access and queries to and from it. Along the most relevant methods in the class, we can find the get() methods that return the relevant information (material types, element types, relationship types) of the queried dictionary entry. In addition, the check methods are useful to verify if the elements satisfy the keywords for the dictionary entry.

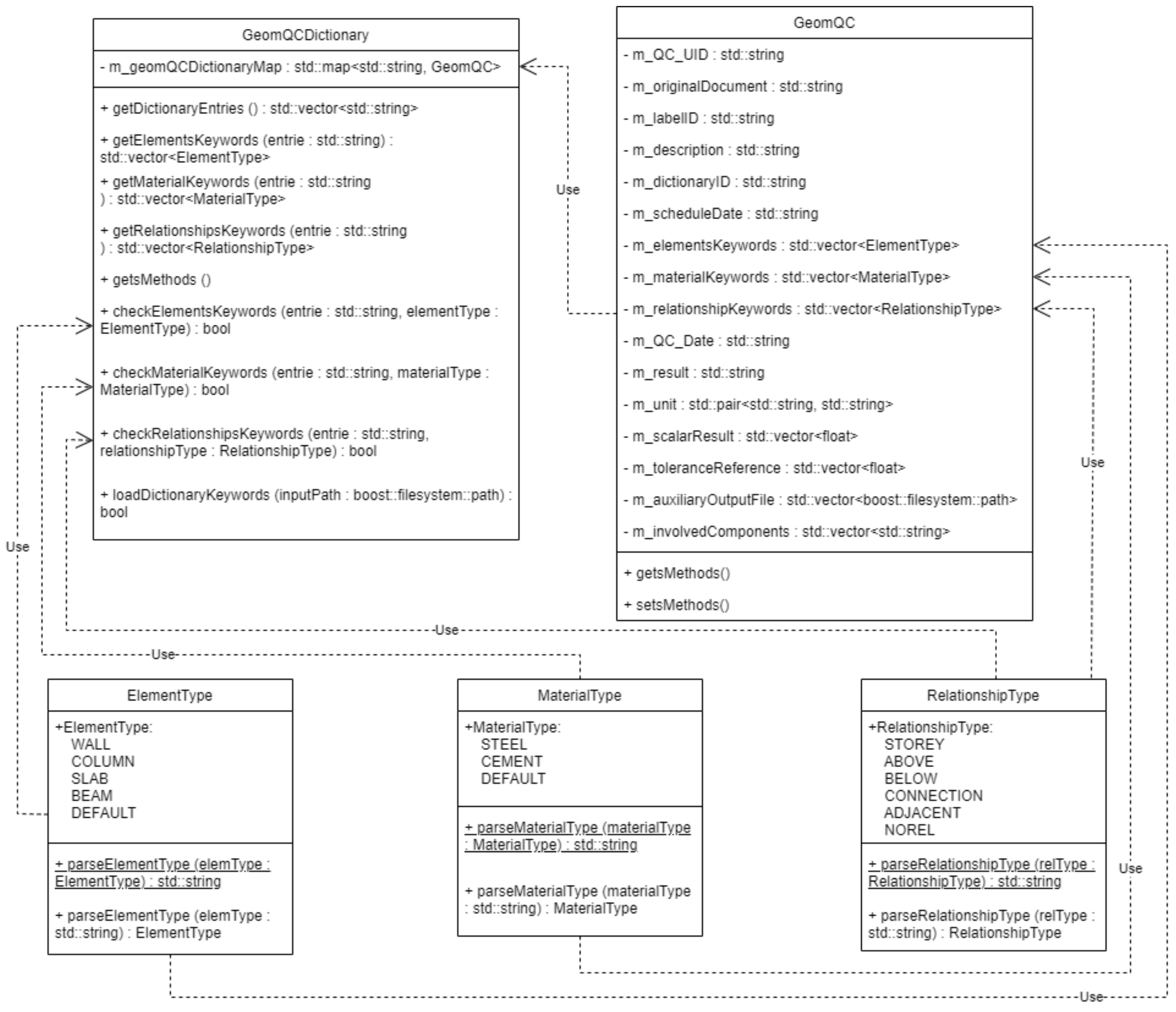

GeomQCDictionary
- m_geomQCDictionaryMap : std::map<std::string, GeomQC>
+ getDictionaryEntries () : std::vector<std::string>
+ getElementsKeywords (entrie : std::string) : std::vector<ElementType>
+ getMaterialKeywords (entrie : std::string) : std::vector<MaterialType>
+ getRelationshipsKeywords (entrie : std::string) : std::vector<RelationshipType>
+ getsMethods ()
+ checkElementsKeywords (entrie : std::string, elementType : ElementType) : bool
+ checkMaterialKeywords (entrie : std::string, materialType : MaterialType) : bool
+ checkRelationshipsKeywords (entrie : std::string, relationshipType : RelationshipType) : bool
+ loadDictionaryKeywords (inputPath : boost::filesystem::path) : bool
GeomQC
- m_QC_UID : std::string
- m_originalDocument : std::string
- m_labelID : std::string
- m_description : std::string
- m_dictionaryID : std::string
- m_scheduleDate : std::string
- m_elementsKeywords : std::vector<ElementType>
- m_materialKeywords : std::vector<MaterialType>
- m_relationshipKeywords : std::vector<RelationshipType>
- m_QC_Date : std::string
- m_result : std::string
- m_unit : std::pair<std::string, std::string>
- m_scalarResult : std::vector<float>
- m_toleranceReference : std::vector<float>
- m_auxiliaryOutputFile : std::vector<boost::filesystem::path>
- m_involvedComponents : std::vector<std::string>
+ getsMethods()
+ setsMethods()
Use
Use
Use
Use
Use
Use
Use
Use
ElementType
+ElementType:
WALL
COLUMN
SLAB
BEAM
DEFAULT
+ parseElementType (elemType : ElementType) : std::string
+ parseElementType (elemType : std::string) : ElementType
MaterialType
+MaterialType:
STEEL
CEMENT
DEFAULT
+ parseMaterialType (materialType : MaterialType) : std::string
+ parseMaterialType (materialType : std::string) : MaterialType
RelationshipType
+RelationshipType:
STOREY
ABOVE
BELOW
CONNECTION
ADJACENT
NOREL
+ parseRelationshipType (relType : RelationshipType) : std::string
+ parseRelationshipType (relType : std::string) : RelationshipType

Figure 5: Geometric Quality Control Dictionary sub-component classes and its relationships

3.1.3 Geometric Relationships Graph

The *Geometric Relationships Graph* is the way the GeometricQC tool interprets the BIM model for geometric and dimensional quality control. In Section 3.1.1 we saw how the different structural elements can be interpreted from the BIM model by using an IFC file. Despite the fact that these files contain all the element types and materials, they rarely contain the geometric relationships between them. Therefore, the *Geometric Relationships Graph* sub-component is designed to detect such relationships and store them in a data structure to facilitate their interpretation. A network graph structure is used for this, where each of the nodes represents a structural element from the BIM model, and each edge represents the geometric relationship between pairs of them.

A network graph is used because it is easy to read and interpret, it can be analysed to provide useful statistical information, e.g., identify key critical elements from the analysis of connectivity, it can be stored and loaded, hence it only needs to be generated once, and it is easily extensible to include other elements, materials, and relationships types, and to include different properties and connections.

The different geometric relationships that have been considered so far, driven by the rules selected from EN 13670:2009 [3], include:

- **Above:** when one structural element (column/wall) is exactly on top of another.
- **Below:** when one structural element (column/wall) is exactly below of another.
- **Adjacent storey level:** when a slab is vertically adjacent to another
- **Same storey adjacency:** when an element is parallel to another of the same type in the same storey.

- **Physical connection:** when an element is physically touching another one (typically beams to columns).

Figure 6 shows an example of the network graph structure using four wall type elements and the geometric relationships between them used so far in the GeometricQC tool.

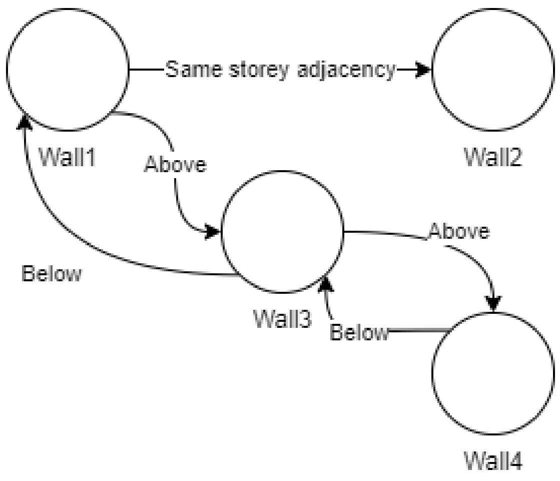


Figure 6: Geometric Relationships Graph example

The *Geometric Relationships Graph* comprises a set of interconnected classes (Figure 7):

- **ElementNode class:** this class represents the graph node and all its information. As each node represents a structural element from the BIM model, it stores the relevant information about it, including general information, such as the UID and label, and information relevant for the GeometricQC tool, such as the element and material types.
- **RelEdge class:** this class represents the graph edges and all their information. As two different nodes (structural elements) are related, they have an edge that connects them. The information required to describe a connection involves the from and to node's IDs, their labels (to make it easier to identify the elements), and the type of relationship that the edge represents.
- **RelationshipType class:** this class contains a list of possible geometric relationships between the elements that can be found in the BIM model and are used in the standards and regulations, as detailed above. This class is likely to be extended to include more geometric relationship types as the development of the tools progresses.
- **RelGraph class:** this class is the container of the network graph data structure which represents the *Geometric Relationships Graph* sub-component. The graph is constructed using the ElementNode and RelEdge classes, representing the nodes and edges respectively. The main methods include add nodes and add edges, get methods, export the graph as CSV file, and methods to check if a structural element is part of the graph.

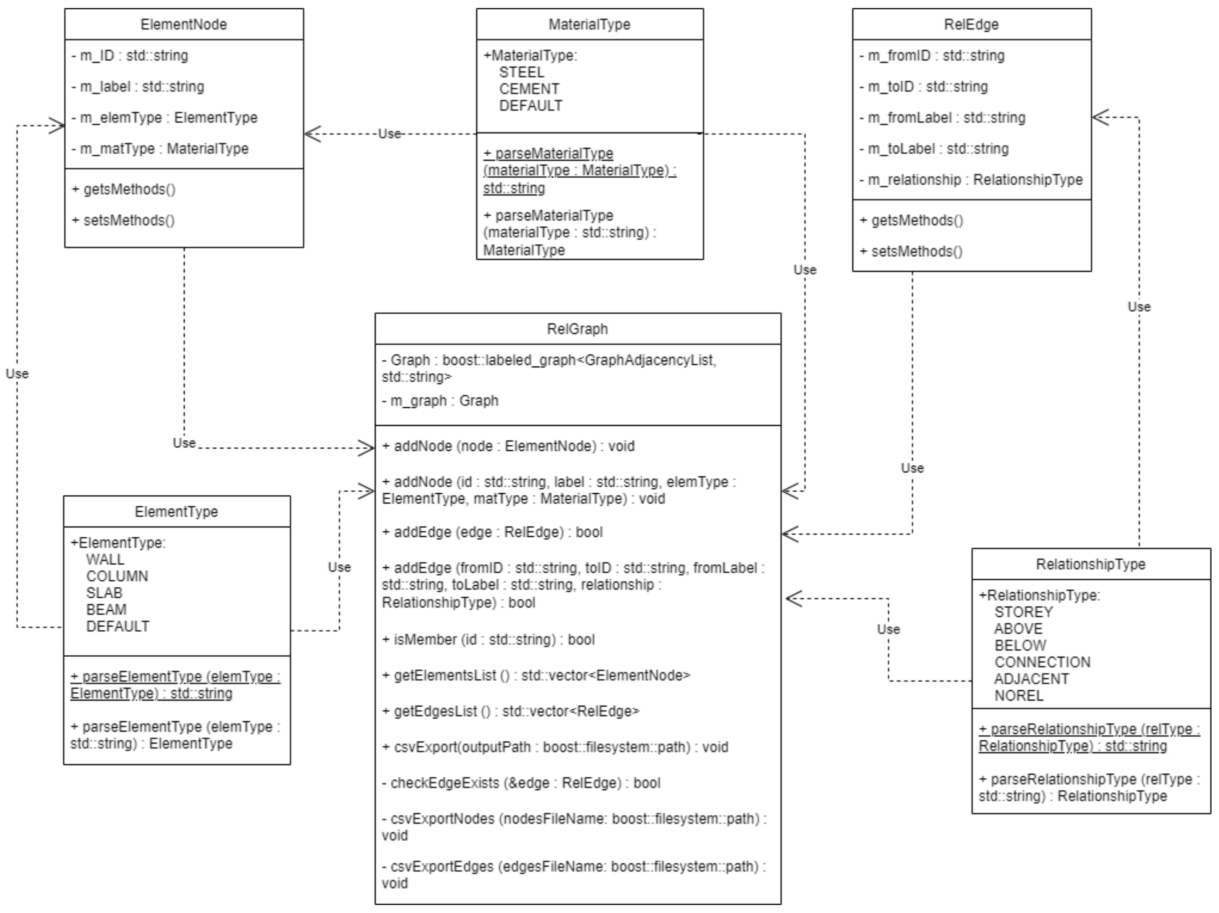

ElementNode
- m_ID : std::string
- m_label : std::string
- m_elemType : ElementType
- m_matType : MaterialType
+ getsMethods()
+ setsMethods()
MaterialType
+MaterialType:
STEEL
CEMENT
DEFAULT
+ parseMaterialType (materialType : MaterialType) : std::string
+ parseMaterialType (materialType : std::string) : MaterialType
RelEdge
- m_fromID : std::string
- m_toID : std::string
- m_fromLabel : std::string
- m_toLabel : std::string
- m_relationship : RelationshipType
+ getsMethods()
+ setsMethods()
RelGraph
- Graph : boost::labeled_graph<GraphAdjacencyList, std::string>
- m_graph : Graph
+ addNode (node : ElementNode) : void
+ addNode (id : std::string, label : std::string, elemType : ElementType, matType : MaterialType) : void
+ addEdge (edge : RelEdge) : bool
+ addEdge (fromID : std::string, toID : std::string, fromLabel : std::string, toLabel : std::string, relationship : RelationshipType) : bool
+ isMember (id : std::string) : bool
+ getElementsList () : std::vector<ElementNode>
+ getEdgesList () : std::vector<RelEdge>
+ csvExport(outputPath : boost::filesystem::path) : void
- checkEdgeExists (&edge : RelEdge) : bool
- csvExportNodes (nodesFileName: boost::filesystem::path) : void
- csvExportEdges (edgesFileName: boost::filesystem::path) : void
ElementType
+ElementType:
WALL
COLUMN
SLAB
BEAM
DEFAULT
+ parseElementType (elemType : ElementType) : std::string
+ parseElementType (elemType : std::string) : ElementType
RelationshipType
+RelationshipType:
STOREY
ABOVE
BELOW
CONNECTION
ADJACENT
NOREL
+ parseRelationshipType (relType : RelationshipType) : std::string
+ parseRelationshipType (relType : std::string) : RelationshipType
Use

Figure 7: Geometric Relationships Graph sub-component classes and its relationships

3.1.4 Quality Control Manager

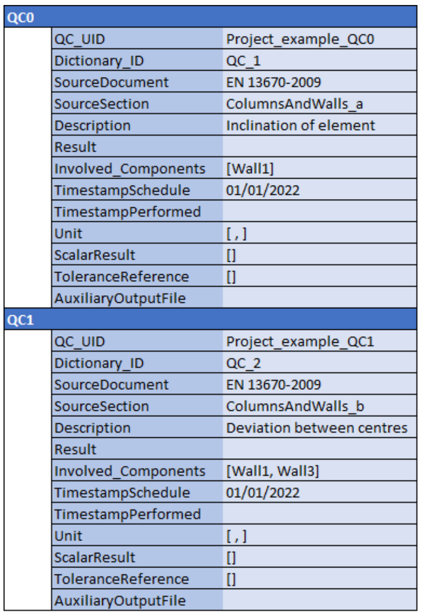
Up to this point, the GeometricQC offline processing tool sub-components are set to (1) extract all the relevant information from the BIM model and the QC Dictionary, and (2) prepare the data structures containing all that information. The *Geometric QC Manager* is presented here to cross-reference all that information and obtain the list of final QC instances that are involved in the construction project.

Given the *Geometric QC Dictionary* from 3.1.2, the *Geometric Relationships Graph* (3.1.3) is queried to provide all the instances of the rule "contexts" in the BIM model, stored by the *BIM Element Manager* (Section 3.1.1), i.e. involving the same *Element and Material Types and Relationship Type*. Figure 8 shows an example of two different QC instances using the same QC Dictionary entries from Figure 4 and the relationship graph from Figure 6. We can see that the list contains similar fields as the QC Dictionary entries, enriched with fields related to the QC execution, such as the QC result and the date it was performed. This was intentional from an implementation point of view, so the same data structure can be reused and it is easier to identify which QC Dictionary rule is referred to in the QC instances. A second version of this sub-component will include a step that checks the construction schedule to automatically populate the *TimestampSchedule* field of each QC rule instance.

As can be seen in Figure 9, the *Geometric QC Manager* includes two main interconnected classes with other data type dependencies:

- **GeomQC class:** this class is used to describe the QC instances for the given construction project. The class contains all the information stored in the dictionary's entries and extra information relative to the execution of the QC verification, such as the schedule timestamp, the timestamp when the rule was verified, the result, used tolerance values and scalar result values, as well as optional output file paths.

- **GeomQCManager class:** this class is in charge of storing all the QC instances and trigger the execution of all the relevant checks for the structural elements of interest. It also contains the list of QC instances performed during the session to export its results to the DTP and update the results from the project's list of QC instances.



QC0		
	QC_UID	Project_example_QC0
	Dictionary_ID	QC_1
	SourceDocument	EN 13670-2009
	SourceSection	ColumnsAndWalls_a
	Description	Inclination of element
	Result	
	Involved_Components	[Wall1]
	TimestampSchedule	01/01/2022
	TimestampPerformed	
	Unit	[,]
	ScalarResult	[]
	ToleranceReference	[]
	AuxiliaryOutputFile	
QC1		
	QC_UID	Project_example_QC1
	Dictionary_ID	QC_2
	SourceDocument	EN 13670-2009
	SourceSection	ColumnsAndWalls_b
	Description	Deviation between centres
	Result	
	Involved_Components	[Wall1, Wall3]
	TimestampSchedule	01/01/2022
	TimestampPerformed	
	Unit	[,]
	ScalarResult	[]
	ToleranceReference	[]
	AuxiliaryOutputFile	

Figure 8: Geometric QC instances list example

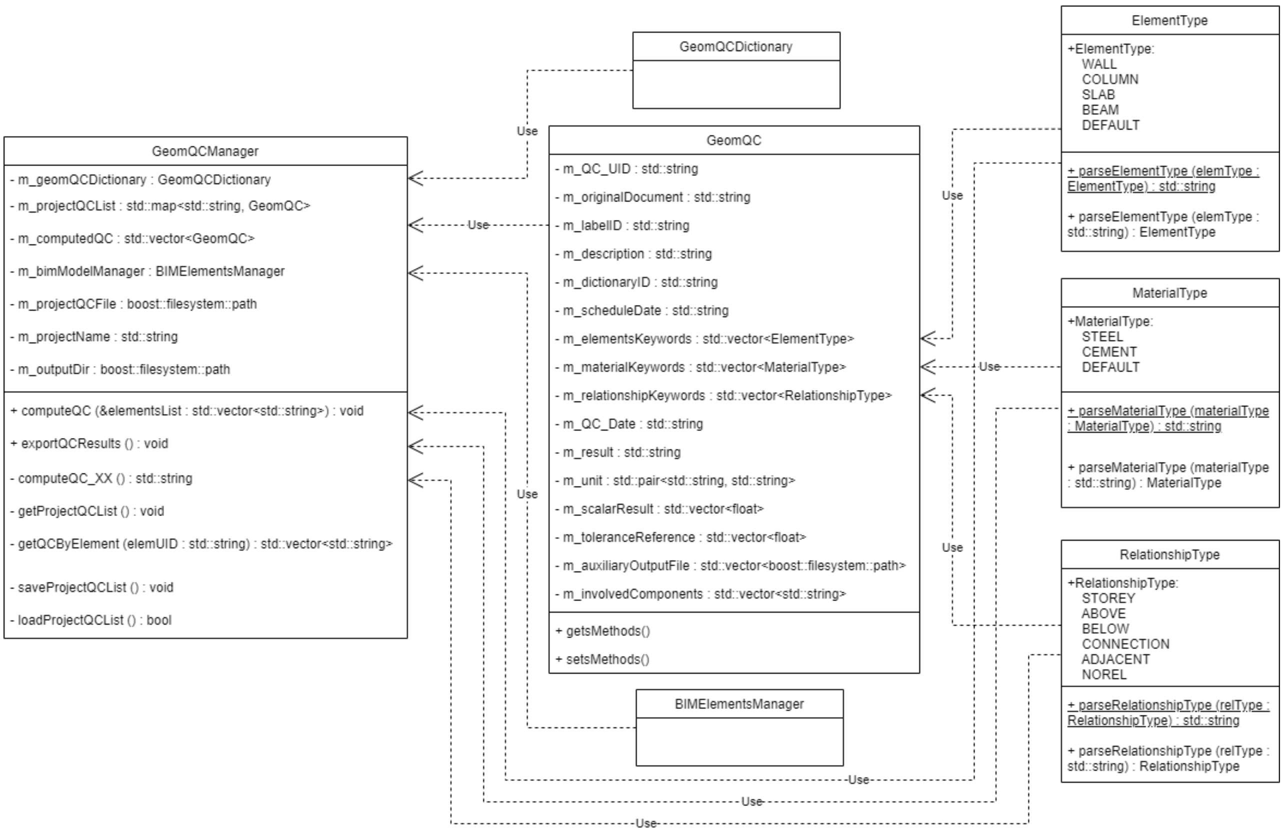

GeomQCDictionary
GeomQCManager
- m_geomQCDictionary : GeomQCDictionary
- m_projectQCList : std::map<std::string, GeomQC>
- m_computedQC : std::vector<GeomQC>
- m_bimModelManager : BIMElementsManager
- m_projectQCFile : boost::filesystem::path
- m_projectName : std::string
- m_outputDir : boost::filesystem::path
+ computeQC (&elementsList : std::vector<std::string>) : void
+ exportQCResults () : void
- computeQC_XX () : std::string
- getProjectQCList () : void
- getQCByElement (elemUID : std::string) : std::vector<std::string>
- saveProjectQCList () : void
- loadProjectQCList () : bool
GeomQC
- m_QC_UID : std::string
- m_originalDocument : std::string
- m_labelID : std::string
- m_description : std::string
- m_dictionaryID : std::string
- m_scheduleDate : std::string
- m_elementsKeywords : std::vector<ElementType>
- m_materialKeywords : std::vector<MaterialType>
- m_relationshipKeywords : std::vector<RelationshipType>
- m_QC_Date : std::string
- m_result : std::string
- m_unit : std::pair<std::string, std::string>
- m_scalarResult : std::vector<float>
- m_toleranceReference : std::vector<float>
- m_auxiliaryOutputFile : std::vector<boost::filesystem::path>
- m_involvedComponents : std::vector<std::string>
+ getsMethods()
+ setsMethods()
BIMElementsManager
ElementType
+ElementType:
WALL
COLUMN
SLAB
BEAM
DEFAULT
+ parseElementType (elemType : ElementType) : std::string
+ parseElementType (elemType : std::string) : ElementType
MaterialType
+MaterialType:
STEEL
CEMENT
DEFAULT
+ parseMaterialType (materialType : MaterialType) : std::string
+ parseMaterialType (materialType : std::string) : MaterialType
RelationshipType
+RelationshipType:
STOREY
ABOVE
BELOW
CONNECTION
ADJACENT
NOREL
+ parseRelationshipType (relType : RelationshipType) : std::string
+ parseRelationshipType (relType : std::string) : RelationshipType
Use
Use
Use
Use
Use
Use
Use
Use
Use

Figure 9: *Geometric QC Manager* sub-component classes and its relationships

3.2 Technology Stack and Implementation Tools

The *GeometricQC tool offline processing* component is a set of headers only libraries that can be included in any software project like any other libraries. The classes are written in C++ v14 for compatibility purposes. In order to be able to facilitate its development, the GeometricQC tool offline processing sub-component uses the following open source C++ libraries:

- **Eigen**: high-level C++ library of template headers for linear algebra, matrix and vector operations, geometrical transformations, numerical solvers and related algorithms. It is used to handle the matrices and vectors of point coordinates, and as an algebra toolbox.
- **Boost**: set of libraries for the C++ programming language that provides support for tasks and structures such as multithreading, regular expressions, unit testing, etc., and improves the capabilities of the C++ language. Boost contains over 150 individual libraries. Boost is used in a number of places within the GeometricQC tool, mainly to deal with JSON formats, system files and directories management and speed up processes.
- **IfcOpenShell**: open-source library designed to enable software developers to work with the IFC file format. It is the main library to read and pre-process BIM IFC files and extract all the relevant information of each element.
- **OpenCascade**: open-source library designed to help software developers to work with 3D CAD, CAM, CAE, B-Rep, etc. It is used by IfcOpenShell to extract and work with the geometry of each BIM element from the IFC file.
- **Open3D**: open-source library designed to help the software developers and users to work with three-dimensional data, such as point clouds and meshes. It is the main library to read and process the point cloud and mesh data.

Table 1: Libraries and Technologies used in the *GeometricQC offline processing* sub-component

Library/Technology Name	Version	License
Eigen	3.3.9	MPL2
Boost	1.76.0	Boost Software License 1.0
IfcOpenShell	0.6.0	LGPL-3.0 License
OpenCascade	7.5.0	LGPL-2.1 License
Open3D	0.13	MPL2

3.3 Input, Output and API Documentation

The *GeometricQC offline processing* component can be used in two different ways: (1) as standalone object classes that can be included in any software project, or (2) as part of the GeometricQC tool. In both cases, it requires the same input parameters:

- **IFC file path**: BIM file in IFC file format of the project. The GeometricQC tool will create a *BIM Element Manager* object using the IFC file path as input parameter, or as in the standalone case, it has to be used as input parameter at object creation. The sub-component will be able to load the IFC file and extract all the BIM elements from it and store them in the manager.
- **Output folder path**: folder path where all the outputs that are generated by the tool will be stored. In the case of *the GeometricQC offiline processing* component, it is the output path where the pre-processing data, such as the *BIM Element Manager* file, the geometric relationships graph and the QC instances list are stored.

This component is designed to be executed once before the construction phase starts and produce the different internal files for the follow up execution instances. The GeometricQC tool can run from the command prompt without requiring a specific API. In its second release, to be consistent with the rest of the COGITO solution, the GeometricQC tool will expose an execution interaction with the Digital Twin Platform. For this, an interface will be designed, in conjunction with WP7, to execute the service and deal with the different input parameters formats and outputs.

3.4 Application Example

The *GeometricQC offline processing* component is essentially used to extract and prepare the geometric information from the BIM file and to produce the list of QC instances that need to be executed and verified during the construction project execution. For such data extraction, it only requires as input the IFC file path to function properly. In this section, we will show the outputs generated using an extensive BIM structural model with hundreds of structural elements.

3.4.1 Revit Technical School model

The Revit Sample Project Technical School is a structural sample model provided by Autodesk [5]. It contains hundreds of different structural elements of concrete (557) and steel (22) material, and it is easy to export it from their proprietary format into the IFC open format used by the COGITO solution. At this stage of development, this model has been considered as an ideal sample for testing and validating the initial version of the developed tools. Figure 10 shows the 3D model representation of the file.

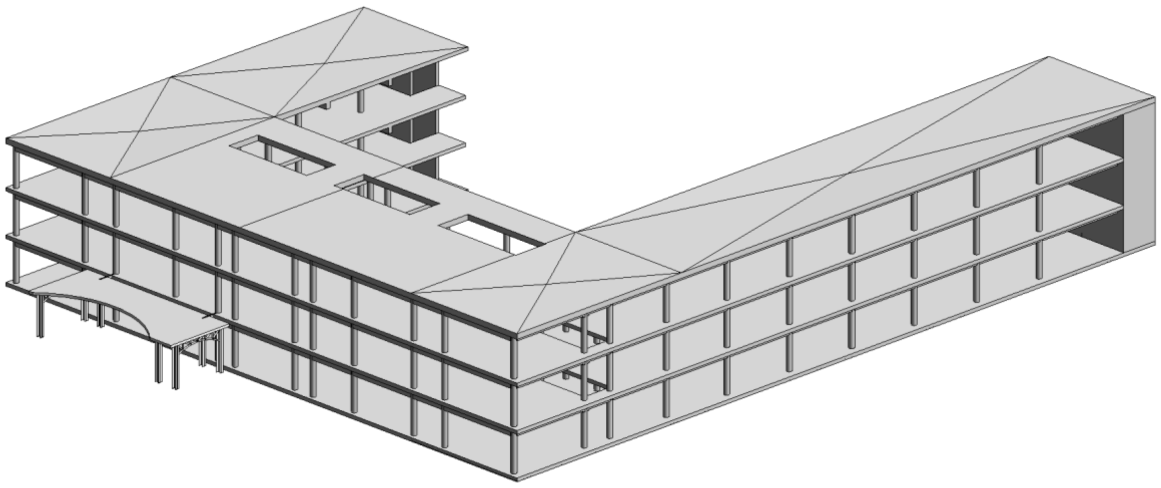


Figure 10: Revit Technical School sample model

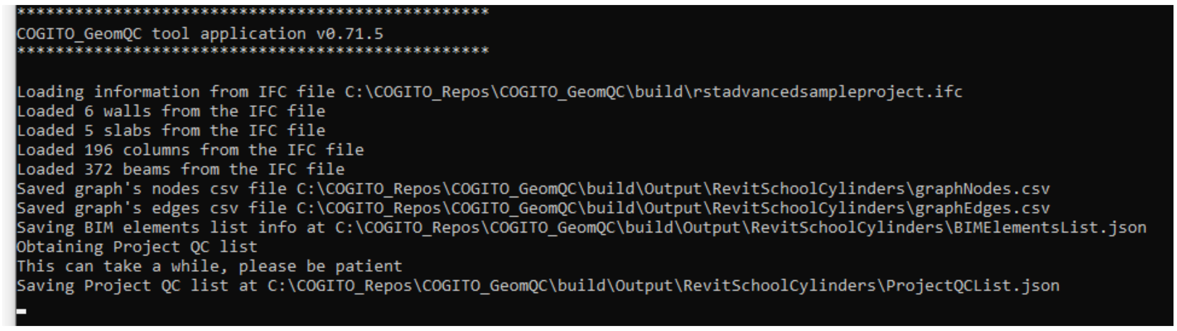
As mentioned above, the current (preliminary) version of the dictionary contains 15 rules, described in 3.1.2, extracted from EN 13670-2009 [3]. The rules involve four types of common structural elements (columns, walls, beams, slabs) and five types of geometrical relationships detailed in 3.1.3 (adjacent storey level, same storey adjacency, above, below, and physical connection).

Once the input IFC file and the output folder have been set, the GeometricQC tool invokes the *GeometricQC offline processing* component to extract all the relevant information and generate the list of QC instances, following a four steps process as detailed below:

- The IFC file is processed and all the structural elements are extracted and stored in the BIM Element Manager.
- The *Geometric Relationships Graph* is constructed by analysing each of the elements stored in the *BIM Element Manager* and computing their relationships using the geometry from the IFC file.
- The *Geometric QC Dictionary* is loaded.
- The *Geometric QC Manager* cross-references each dictionary entry with the graph network object and creates a QC instance once the "context" is satisfactory.

As depicted in Figure 11, the *GeometricQC offline processing* component extracted the 579 structural elements, and identified 6,677 relevant geometrical relationships stored in the *Geometrical Relationships Graph* from the Revit Technical School sample model. The nodes and geometrical relationships can be detailed as follows:

- Structural elements (nodes):
  - 6 walls
  - 5 slabs
  - 196 columns
  - 372 beams
- Geometric relationships (edges):
  - 126 Above
  - 126 Below
  - 6,013 Same storey adjacency
  - 407 Physical connection
  - 5 Adjacent storey level



```
************************************************
COGITO_GeomQC tool application v0.71.5
************************************************

Loading information from IFC file C:\COGITO_Repos\COGITO_GeomQC\build\rstadvancedsampleproject.ifc
Loaded 6 walls from the IFC file
Loaded 5 slabs from the IFC file
Loaded 196 columns from the IFC file
Loaded 372 beams from the IFC file
Saved graph's nodes csv file C:\COGITO_Repos\COGITO_GeomQC\build\Output\RevitSchoolCylinders\graphNodes.csv
Saved graph's edges csv file C:\COGITO_Repos\COGITO_GeomQC\build\Output\RevitSchoolCylinders\graphEdges.csv
Saving BIM elements list info at C:\COGITO_Repos\COGITO_GeomQC\build\Output\RevitSchoolCylinders\BIMElementsList.json
Obtaining Project QC list
This can take a while, please be patient
Saving Project QC list at C:\COGITO_Repos\COGITO_GeomQC\build\Output\RevitSchoolCylinders\ProjectQCList.json
_
```

Figure 11: GeometricQC tool offline processing output example

Figure 12 illustrates the graph network where the nodes represent a structural element, and the edges the geometric relationship between them. The image contains a legend with the colour code meaning.

After cross-referencing all the dictionary entries and graph, the *GeometricQC offline processing* component obtained 9,750 geometric QC instances with the following breakdown:

- **QC_1 Inclination of a column/wall** *(10.4 Columns and Walls, No a)*: **195**
- **QC_2: Deviation between centres** *(10.4 Columns and Walls, No b)*: **126**
- **QC_3: Curvature of a column/wall between adjacent storey levels** *(10.4 Columns and Walls, No c)*: **195**
- **QC_4: Location of a column/wall at any storey level with respect to base level** *(10.4 Columns and Walls, No d)*: **126**
- **QC_5: Location of a beam-to-column connection measured relative to the column** *(10.5 Beams and Slabs, No a)*: **396**
- **QC_6: Position of bearing axis of support** *(10.5 Beams and Slabs, No b)*: **357**
- **QC_7: Cross-sectional dimensions** *(10.6 Sections, No a)*: **551**
- **QC_8: Lap-joints** *(10.6 Sections, No c)*: **11**
- **QC_9: Free space between adjacent columns/walls** *(Annex G – G.10.4 Columns and Walls, No c)*: **3039**
- **QC_10: Horizontal straightness of beams** *(Annex G – G.10.5 Beams and Slabs, No a)*: **357**
- **QC_11: Distance between adjacent beams** *(Annex G – G.10.5 Beams and Slabs, No b)*: **1737**
- **QC_12: Inclination of a beam/slab** *(Annex G – G.10.5 Beams and Slabs, No c)*: **362**
- **QC_13: Level of adjacent beams** *(Annex G – G.10.5 Beams and Slabs, No d)*: **1737**
- **QC_14: Level of adjacent floors at supports** *(Annex G – G.10.5 Beams and Slabs, No e)*: **5**
- **QC_15: Orthogonality of a cross-section** *(Annex G – G.10.6 Sections, No a)*: **557**

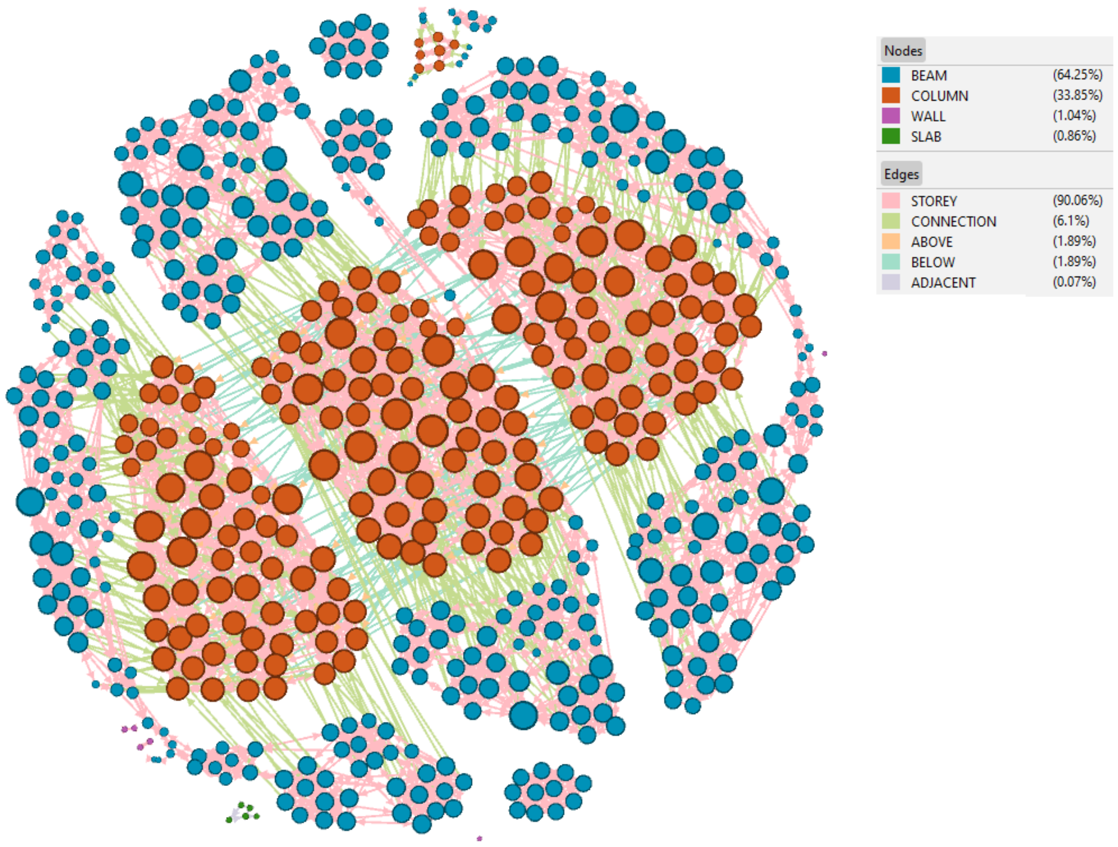
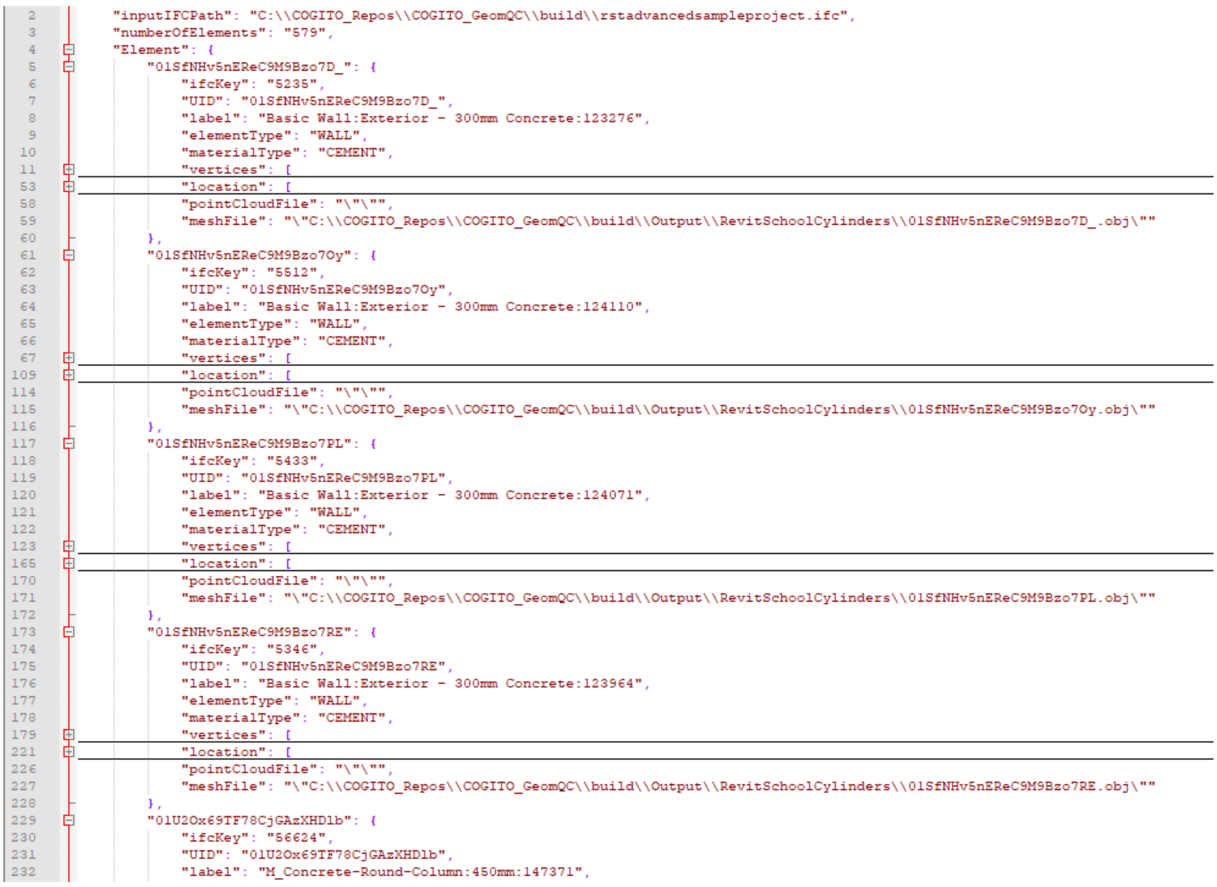


Figure 12: Graph representation of the Revit Technical School sample model after GeometricQC offline processing

Taking into account that we are using a limited preliminary QC dictionary with only 15 rules, and that we are only considering the concrete structures, we can see the significant amount of detailed geometrical and dimensional tolerances verifications that need to happen during the construction phase, some of them also requiring high precision measurements and meticulous labour work. The GeometricQC tool would be able to facilitate the execution of all those QC instances and generate a detailed and accurate result for each of them, allowing the involved stakeholders to focus on only the remedial work that needs to be carried over in case a requirement is not fulfilled.

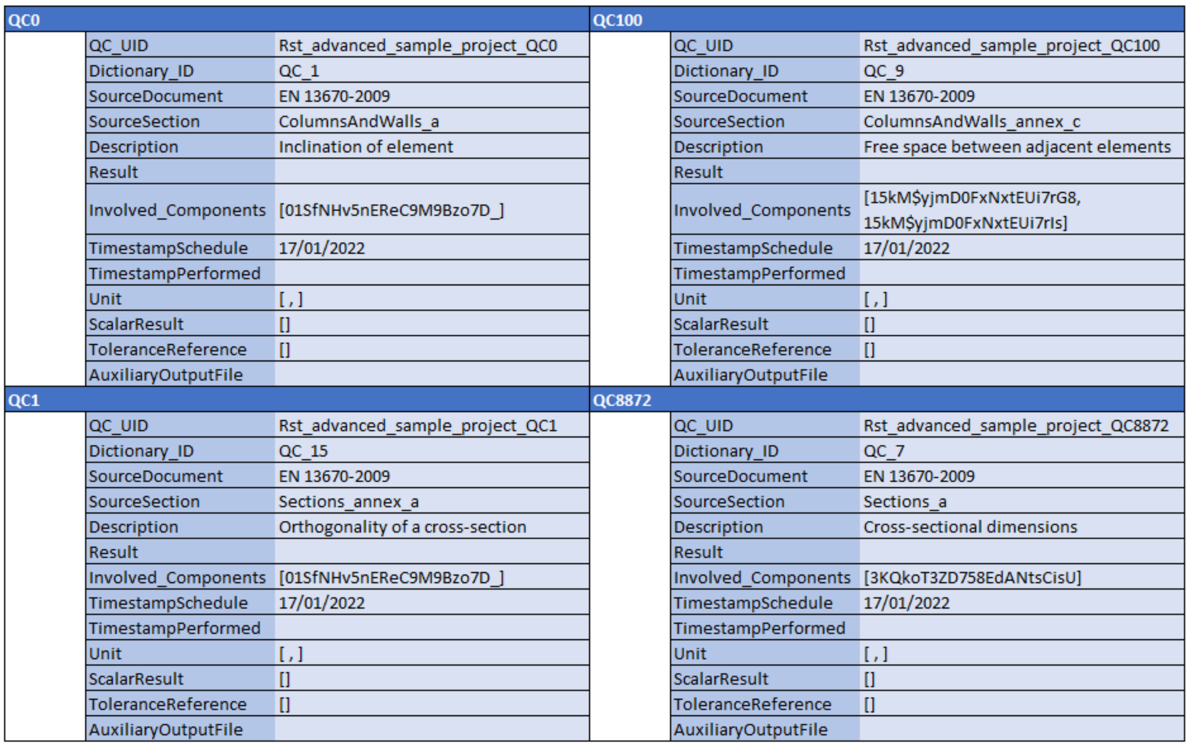
Finally, after all the offline processing is done, the GeometricQC tool generates four (4) different files, which are then internally available for the rest of the workflow and subsequent execution instances of the tool:

- **BIM Element list file (*BIMElementList.json*):** contains all the structural elements extracted from the IFC file, and the necessary information for the GeometricQC tool to use them without having to read the IFC file every time some information is needed (Figure 14).
- **Geometric Relationships Graph node list file (*graphNodes.csv*):** file with the same information as the *ElementNode* class in a comma-separated format. This file format is easy to read and interpreted and can be opened by graph interpreter tools like Gephi [6].
- **Geometric Relationships Graph edges list file (*graphEdges.csv*):** file with the same information as the *RelEdge* class in a comma-separated format. As the nodes file, it can be read and interpreted with a graph interpreter tool like Gephi [6].
- **Project QC instances list (*ProjectQCList.json*):** this file contains the list of all the QC instances that belong to the project. This list is definitive, and will be updated every time one of them is executed. Figure 13 shows an example of four of those QC instances that were obtained from the Revit Technical School model.



```
    "inputIFCPath": "C:\\COGITO_Repos\\COGITO_GeomQC\\build\\rstadvancedsampleproject.ifc",
    "numberOfElements": "579",
    "Element": {
        "01SfNHv5nEReC9M9Bzo7D_": {
            "ifcKey": "5235",
            "UID": "01SfNHv5nEReC9M9Bzo7D_",
            "label": "Basic Wall:Exterior - 300mm Concrete:123276",
            "elementType": "WALL",
            "materialType": "CEMENT",
            "vertices": [
            "location": [
            "pointCloudFile": "\"\"",
            "meshFile": "\"C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolCylinders\\01SfNHv5nEReC9M9Bzo7D_.obj\""
        },
        "01SfNHv5nEReC9M9Bzo7Oy": {
            "ifcKey": "5512",
            "UID": "01SfNHv5nEReC9M9Bzo7Oy",
            "label": "Basic Wall:Exterior - 300mm Concrete:124110",
            "elementType": "WALL",
            "materialType": "CEMENT",
            "vertices": [
            "location": [
            "pointCloudFile": "\"\"",
            "meshFile": "\"C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolCylinders\\01SfNHv5nEReC9M9Bzo7Oy.obj\""
        },
        "01SfNHv5nEReC9M9Bzo7PL": {
            "ifcKey": "5433",
            "UID": "01SfNHv5nEReC9M9Bzo7PL",
            "label": "Basic Wall:Exterior - 300mm Concrete:124071",
            "elementType": "WALL",
            "materialType": "CEMENT",
            "vertices": [
            "location": [
            "pointCloudFile": "\"\"",
            "meshFile": "\"C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolCylinders\\01SfNHv5nEReC9M9Bzo7PL.obj\""
        },
        "01SfNHv5nEReC9M9Bzo7RE": {
            "ifcKey": "5346",
            "UID": "01SfNHv5nEReC9M9Bzo7RE",
            "label": "Basic Wall:Exterior - 300mm Concrete:123964",
            "elementType": "WALL",
            "materialType": "CEMENT",
            "vertices": [
            "location": [
            "pointCloudFile": "\"\"",
            "meshFile": "\"C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolCylinders\\01SfNHv5nEReC9M9Bzo7RE.obj\""
        },
        "01U2Ox69TF78CjGAzXHDlb": {
            "ifcKey": "56624",
            "UID": "01U2Ox69TF78CjGAzXHDlb",
            "label": "M_Concrete-Round-Column:450mm:147371",
```

Figure 13: BIM Element Manager JSON file example obtained from the Revit Technical School model using the GeometricQC tool



QC0			QC100		
	QC_UID	Rst_advanced_sample_project_QC0		QC_UID	Rst_advanced_sample_project_QC100
	Dictionary_ID	QC_1		Dictionary_ID	QC_9
	SourceDocument	EN 13670-2009		SourceDocument	EN 13670-2009
	SourceSection	ColumnsAndWalls_a		SourceSection	ColumnsAndWalls_annex_c
	Description	Inclination of element		Description	Free space between adjacent elements
	Result			Result	
	Involved_Components	[01SfNHv5nEReC9M9Bzo7D_]		Involved_Components	[15kM$yjmD0FxNxtEUi7rG8, 15kM$yjmD0FxNxtEUi7rIs]
	TimestampSchedule	17/01/2022		TimestampSchedule	17/01/2022
	TimestampPerformed			TimestampPerformed	
	Unit	[,]		Unit	[,]
	ScalarResult	[]		ScalarResult	[]
	ToleranceReference	[]		ToleranceReference	[]
	AuxiliaryOutputFile			AuxiliaryOutputFile	
QC1			QC8872		
	QC_UID	Rst_advanced_sample_project_QC1		QC_UID	Rst_advanced_sample_project_QC8872
	Dictionary_ID	QC_15		Dictionary_ID	QC_7
	SourceDocument	EN 13670-2009		SourceDocument	EN 13670-2009
	SourceSection	Sections_annex_a		SourceSection	Sections_a
	Description	Orthogonality of a cross-section		Description	Cross-sectional dimensions
	Result			Result	
	Involved_Components	[01SfNHv5nEReC9M9Bzo7D_]		Involved_Components	[3KQkoT3ZD758EdANtsCisU]
	TimestampSchedule	17/01/2022		TimestampSchedule	17/01/2022
	TimestampPerformed			TimestampPerformed	
	Unit	[,]		Unit	[,]
	ScalarResult	[]		ScalarResult	[]
	ToleranceReference	[]		ToleranceReference	[]
	AuxiliaryOutputFile			AuxiliaryOutputFile	

Figure 14: Revit Technical School geometrical tolerances list example

3.5 Licensing

The GeometricQC tool is free software; you can redistribute it and/or modify it under the terms of the **GNU General Public License Version 3** as published by the Free Software Foundation (https://www.gnu.org/licenses/gpl-3.0.en.html).

3.6 Installation Instructions

No complex installation is required. Binaries will be provided in the Cyberbuild Datashare collections. GeometricQC tool is a set of header-only libraries that can be included in any project just by adding the header files to the includes list.

3.7 Development and integration status

The *GeometricQC offline processing* sub-component is currently under development, already achieving all the main basic functionalities which are ready to use. The *GeometricQC offline processing* sub-component does not interact with any external component; all the interactions are within the GeometricQC tool.

It is expected new functionalities will be available in the final version of the GeometricQC tool. A summary road map of the new functionalities that should be covered can be found below:

- Improve generated file loader speed.
- Include new rules and geometric relationships obtained from the steel standards EN 1090-2 [4] and project sites specifics.
- Take into account the planned schedule (4D BIM) of the execution of work for scheduling the QC instances.
- Code refactor and clean up.

3.8 Requirements Coverage

Despite the fact that the *GeometricQC offline processing* component is just a back-end part of the GeometricQC tool, it already covers a couple of the requirements defined in D2.4, D2.5 and D2.1.

The functional and non-functional requirements that were introduced in the COGITO System Architecture and that have already been covered by this version of the *GeometricQC offline processing* are presented in Table 2. Functional requirement Req-1.1 is partially covered using this component, thanks to the IFC file loader and interpreter. However, the schedule part of the 4D BIM will be covered in the second version of the sub-component. In addition, despite storing the matched point cloud data file path into each BIM element, the *GeometricQC offline processing* component does not have the capability to load the point cloud data. That part is covered by the *Point Cloud Matching and Segmentation* sub-component as defined in D5.1. Thanks to the C++ property of using different objects in header only classes, Req-2.1, Req-2.2, and Req-2.3 are well covered with this component. As previously mentioned, this set of classes (as part of the full GeometricQC tool library) can be used autonomously in any other software project, or as part of the full GeometricQC tool. In addition, its scalability is well handled using object-oriented programming, which enables straightforward addition of new functionalities and properties to each of the components.

Table 2: *GeometricQC offline processing* sub-component requirements coverage from D2.4 and D2.5

ID	Description	Type	Status
Req-1.1	Loading as planned 4D BIM and point cloud data	Functional	Partially achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Reusability	Non-Functional	Achieved

Req-2.3	Interoperability	Non-Functional	Achieved

Table 3 presents the stakeholders requirements that have been documented in D2.1 and are relevant to this component. COGI-CS-1 and COGI-CS-4 are covered simply by the programming language that has been selected for the development of the GeometricQC tool. COGI-QC-8 is partially achieved thanks to the inclusion of quality control rules regarding concrete structures. Regarding COGI-QC-11, the *GeometricQC offline processing* component allows to load and interpret the as-designed data from the BIM model and extract the QC instances where the QC-related activities should be conducted, thus allowing to automate their geometric QC control in future QC activities.

Table 3: *GeometricQC offline processing* sub-component requirements coverage from D2.1

ID	Description	Type	Priority	Status
COGI-CS-1	Runs on desktop or laptop PC	• Operational	Must	Achieved
COGI-CS-4	Runs on Windows	• Operational	Must	Achieved
COGI-CS-5	Runs on Mac	• Operational	Could	Achieved
COGI-QC-8	Supports systematic quality control on earthworks, substructures, concrete works	• Performance	Should	Partially achieved
COGI-QC-11	Automates QC-related activities	• Functional • Operational	Must	Achieved

3.9 Assumptions and Restrictions

The first version of the *GeometricQC offline processing* component has been delivered under certain assumptions and restrictions, listed below:

- It has been developed to be part of the GeometricQC tool, so no standalone programme is supposed to be executed for this component.
- It currently supports IFC files of version 4.0.
- It requires the input IFC file to be correct (free of geometric errors), consistent with the IFC 4.0 schema and containing the material for each of the structural elements.
- BIM elements are expected to have the vertical along the +Z axis using the right-hand side coordinate frame.
- The elements' reference coordinate frame within the entire GeometricQC tool uses global coordinates; hence the generated output files contain coordinates information in the same global space.
- The measures are stored using the international metric system, and the distance measurement unit used in the sub-component is the metre [m].

4 GeometricQC tool active geometric quality control analysis

In this section, we describe the *GeometricQC tool active geometric quality control analysis* component. This component, as can be seen in Figure 1, is the one in charge of querying all the necessary data from the other components, trigger the *Point Cloud Matching and Segmentation* component, and execute the QC instances relevant to the structural elements of interest at the current time in the execution of the project. As a result, the project's QC instances obtained from the *GeometricQC offline processing* component are updated with the results, a new list of performed QCs is generated with only the information relevant to each GeometricQC tool execution, and the relevant complementary files (such as result colour coded meshes and point clouds) are generated to be shared with the different stakeholders and other COGITO tools.

The following sub-sections describe in more detail the *GeometricQC tool active geometric quality control analysis* component and its technical aspects and requisites.

4.1 Prototype Overview

The objective of the *GeometricQC tool active geometric quality control analysis* component is to verify all the QC instances in each GeometricQC tool execution given the list of structural elements of interest listed from the QC work order, i.e. the ones that have been just built, scanned, and need to be geometrically checked – hereafter we refer to these elements as the *active* structural elements. This component does not have an object class. Instead, it can be seen as an integration component, where all the other parts of the GeometricQC tool come together. Figure 15: GeometricQC tool active geometric quality control analysis sub-component detailed workflow shows a more detailed workflow for the component than the simplified version presented in Section 2.

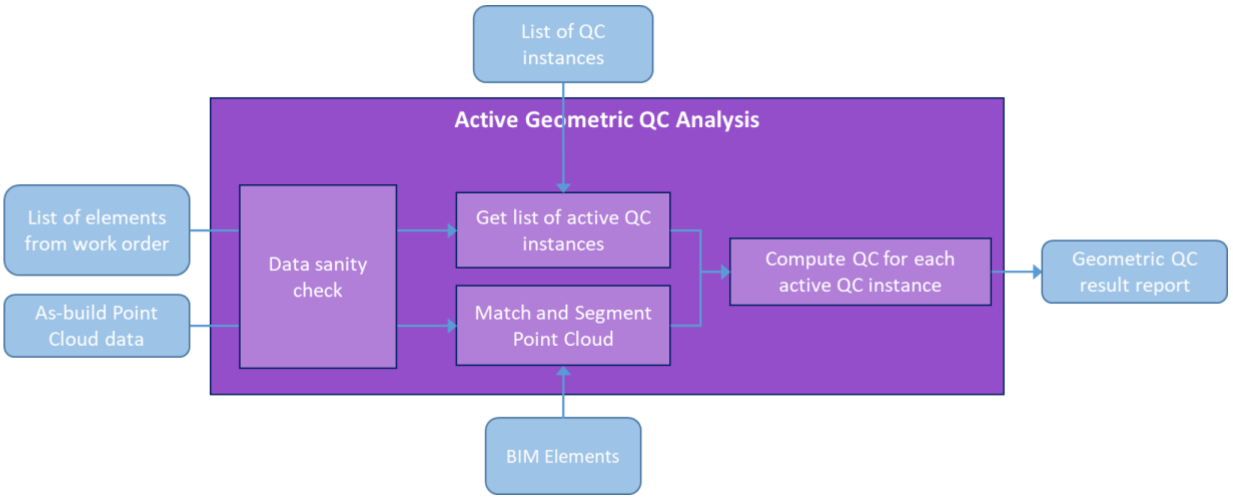

List of QC instances
Active Geometric QC Analysis
List of elements from work order
As-build Point Cloud data
Data sanity check
Get list of active QC instances
Match and Segment Point Cloud
Compute QC for each active QC instance
Geometric QC result report
BIM Elements

Figure 15: GeometricQC tool active geometric quality control analysis sub-component detailed workflow

The *GeometricQC tool active geometric quality control analysis* component is executed by the GeometricQC communication sub-component, which will act as an interface between the DTP and the rest of the GeometricQC tool. In that interface, the relevant input parameters are prepared and the relevant service call is made. The current version of the component only requires:

- The list of active structural elements of interest listed from the QC work order. The list of active structural elements only contains the unique IDs (UIDs) of those elements, and thus can be passed as a text file.
- The (geo-)referenced as-built point cloud data associated to the same QC work order. The as-built point cloud data is required to be in a well-known open format, such as PLY and/or E57.

The next step for the component is to obtain the list of structural elements and conduct some *data sanity check* to verify that all of them are part of the *BIM Element Manager*, and that all the relevant information is available and ready to be used. If an error is found, the sub-component returns a result with a failure message accompanied with an explanation. If everything is satisfactory, the sub-component iterates through the list of structural elements, and for each of them performs the matching and segmentation of the input point clouds (using the Point Cloud Matching and Segmentation sub-component described in D5.1). Besides, the sub-component and obtains a list of the active QC instances, i.e. the QC instance that can be checked at the current time.

At this point, the component is ready to perform the quality control defined in each of the active QC instances. For each QC instance, it calls the relevant QC algorithm with all the required input data.

As explained in Section 3.1.4, all QC types have the same output, making it easily interoperable. Once a QC instance is verified, the overall list of project QC instances is updated with the result (in order to keep track of the QC progress for the entire construction project), the result is exported into a structured output file (JSON) containing the detailed QC tolerance check information, and a set of auxiliary files are generated. These files include: the segmented point clouds and meshes coloured according to the QC instance results, allowing if required, to visually inspect the results and check at first glance where the problem is and which one it is using the DigitAR or the DCC.

The current implementation of the *GeometricQC tool active geometric quality control analysis* component includes the QC algorithms for 6 entries in the Geometric QC Dictionary:

- **QC_1 Inclination of a column/wall** *(10.4 Columns and Walls, No a)*
- **QC_2: Deviation between centres** *(10.4 Columns and Walls, No b)*
- **QC_5: Location of a beam-to-column connection measured relative to the column** *(10.5 Beams and Slabs, No a)*
- **QC_9: Free space between adjacent columns/walls** *(Annex G – G.10.4 Columns and Walls, No c)*
- **QC_12: Inclination of a beam/slab** *(Annex G – G.10.5 Beams and Slabs, No c)*
- **QC_14: Level of adjacent floors at supports** *(Annex G – G.10.5 Beams and Slabs, No e)*

4.2 Technology Stack and Implementation Tools

The *GeometricQC tool active geometric quality control analysis* is a set of headers only libraries that can be included in any software project as any other libraries. The classes are written in C++ v14 for compatibility purposes. In order to be able to facilitate its development, the *GeometricQC tool active geometric quality control analysis* component uses the following open source C++ libraries:

- **Eigen**: high-level C++ library of template headers for linear algebra, matrix and vector operations, geometrical transformations, numerical solvers and related algorithms. It is used to handle the matrices and vectors of point coordinates, and as an algebra toolbox.
- **Boost**: set of libraries for the C++ programming language that provides support for tasks and structures such as multithreading, regular expressions, unit testing, etc., and improves the capabilities of the C++ language. Boost contains over 150 individual libraries. Boost is used in a number of places within the GeometricQC tool, mainly to deal with JSON formats, system files and directories management and speed up processes.
- **Open3D**: open-source library designed to help the software developers and users to work with three-dimensional data, such as point clouds and meshes. It is the main library to read and process the point cloud and mesh data.

Table 4: Libraries and Technologies used in the GeometricQC tool active geometric quality control analysis sub-component

Library/Technology Name	Version	License

Eigen	3.3.9	MPL2
Boost	1.76.0	Boost Software License 1.0
Open3D	0.13	MPL2

4.3 Input, Output and API Documentation

The *GeometricQC tool active geometric quality control analysis* component requires the following input parameters:

- **List of structural elements UIDs:** a text file with the list of the active structural elements UIDs of interest that are contained in the QC work order.
- **As-built point cloud data file names:** a text file with the list of as-built point clouds data file names.
- **Output folder path**: folder path where to store all the outputs that are generated by the tool. In the case of the *GeometricQC tool active geometric quality control analysis* component, this is the output path where to store the colour coded point clouds and meshes, and the QC results output file.

This component generates a series of output files:

- **QC instances' auxiliary files:** these files are point clouds and meshes coloured according to each QC result.
- **QC results output file:** JSON file QC instances containing only the results for the entries related to the active input UIDs.

This component (and consequently the GeometricQC tool) can run from the command prompt without requiring a specific API. In its second release, to be consistent with the rest of the COGITO solution, the GeometricQC tool will expose an API to enable any interaction with the Digital Twin Platform. For this, an interface will be designed, in conjunction with WP7, to execute the service and deal with the different input parameters formats and outputs.

4.4 Application Example

The *GeometricQC tool active geometric quality control analysis* component is used to perform dimensional and geometric QC over the QC instances obtained by the *GeometricQC offline processing* component. For this, it only requires the list of structural elements that need to be verified, and the as-built point cloud data, already registered in the project's coordinate system. In this section, we will show the outputs generated using an extensive BIM structural model with hundreds of structural elements**.**

4.4.1 Revit Technical School model

The Revit Sample Project Technical School is a structural sample model provided by Autodesk [5]. Please see Section 3.4.1 for more details.

In this case we modified the positioning of a couple of elements in the mesh file and generated a synthetic point cloud with some noise to mimic a realistic laser scanning result. We can see the generated point cloud next to the BIM model in Figure 16.

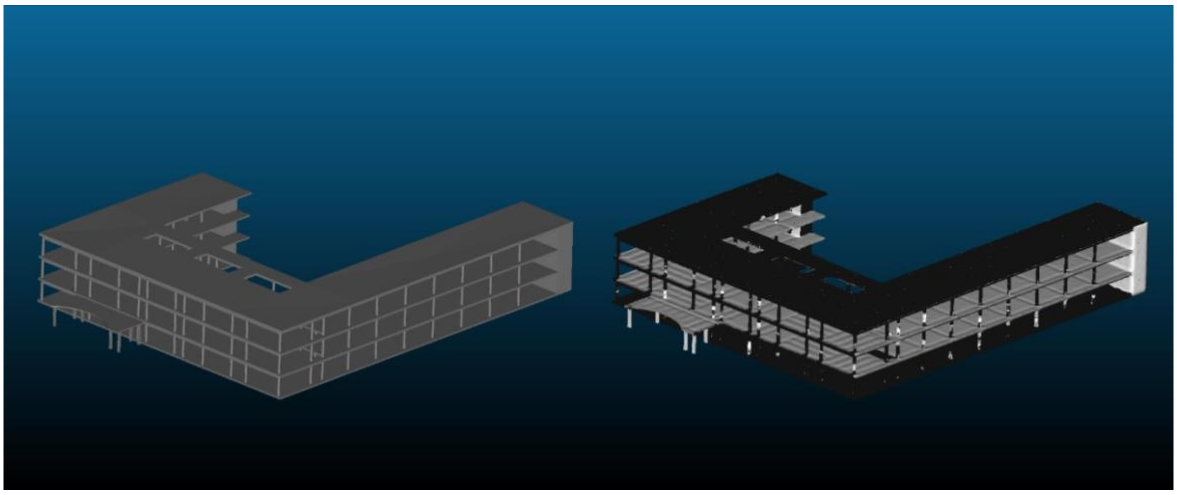


Figure 16: Revit Sample Project Technical School BIM model (left) and synthetic point cloud (right)

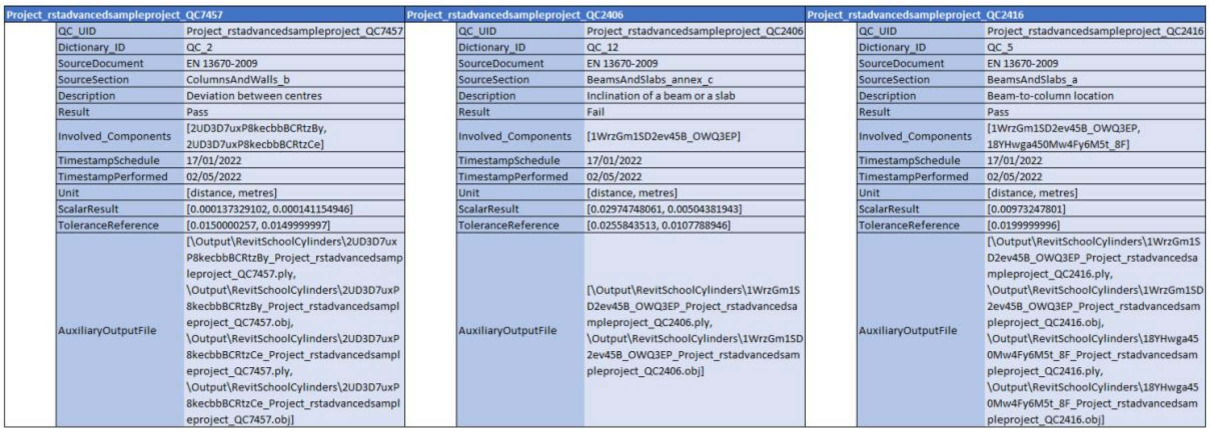
From the *GeometricQC offline processing* component we obtained the list of QC instances (3.4.1), so now we start from a list of structural elements, including at least one of each of the 4 types, and the synthetic point cloud. The list of structural elements includes 13 different UIDs.

After loading all the required information, and performing the point cloud matching and segmentation, the GeometricQC tool analyses a total of 141 QC instances broken down as follows:

- **QC_1 Inclination of a column/wall** *(10.4 Columns and Walls, No a)*: **8**
- **QC_2: Deviation between centres** *(10.4 Columns and Walls, No b)*: **8**
- **QC_5: Location of a beam-to-column connection measured relative to the column** *(10.5 Beams and Slabs, No a)*: **21**
- **QC_9: Free space between adjacent columns/walls** *(Annex G – G.10.4 Columns and Walls, No c)*: **96**
- **QC_12: Inclination of a beam/slab** *(Annex G – G.10.5 Beams and Slabs, No c)*: **5**
- **QC_14: Level of adjacent floors at supports** *(Annex G – G.10.5 Beams and Slabs, No e)*: **3**

For this execution, the 141 QC instances from the project's list were updated. In addition, a new output file related to the active structural elements of interest is also generated containing the results of those 141 QC instances. Figure 17 shows a couple of examples in that output result file. We can see that in the first case, the QC refers to the dictionary entry **"QC_2: Deviation between centres"**, that the results is a ***"Pass",*** which means the geometric tolerances are satisfactory, and what the scalar deviation for the X and Y axis are, as well as what the geometric tolerance is to determine the QC result. We can also analyse the auxiliary files that were generated, corresponding to the meshes and segmented point clouds for the two columns (Figure 18). In this case, since the QC result is satisfactory, the objects are coloured green and we can see from the detailed views that there are not significant deviations.

The other results of Figure 17 show the outcome of performing 2 QCs for a beam that has incorrect pose and position. Both results **"QC_12: Inclination of a beam/slab"** and **"QC_5: Location of a beam-to-column connection measured relative to the column"** contain all the same information as the previous example, but in this case, the result for "**QC_12**" returns a ***"Fail"***, since the beam is compromised, but a *"**Pass**"* for "**QC_5**" because the connection between the beam and column is measured with respect to the centre of the column and a vertical deviation should not affect it. Figure 19 shows the inspection of the auxiliary files, where we can see that both meshes and point clouds are coloured red, allowing the user to visualise the problems by exploring the as-built data. A close-up view in this case allows us to verify that the as-built data indeed presents deviations and these are significant enough for the QCs to fail.



Project_rstadvancedsampleproject_QC7457	
QC_UID	Project_rstadvancedsampleproject_QC7457
Dictionary_ID	QC_2
SourceDocument	EN 13670-2009
SourceSection	ColumnsAndWalls_b
Description	Deviation between centres
Result	Pass
Involved_Components	[2UD3D7uxP8kecbbBCRtzBy, 2UD3D7uxP8kecbbBCRtzCe]
TimestampSchedule	17/01/2022
TimestampPerformed	02/05/2022
Unit	[distance, metres]
ScalarResult	[0.000137329102, 0.000141154946]
ToleranceReference	[0.0150000257, 0.0149999997]
AuxiliaryOutputFile	[\Output\RevitSchoolCylinders\2UD3D7uxP8kecbbBCRtzBy_Project_rstadvancedsampleproject_QC7457.ply, \Output\RevitSchoolCylinders\2UD3D7uxP8kecbbBCRtzBy_Project_rstadvancedsampleproject_QC7457.obj, \Output\RevitSchoolCylinders\2UD3D7uxP8kecbbBCRtzCe_Project_rstadvancedsampleproject_QC7457.ply, \Output\RevitSchoolCylinders\2UD3D7uxP8kecbbBCRtzCe_Project_rstadvancedsampleproject_QC7457.obj]

Project_rstadvancedsampleproject_QC2406	
QC_UID	Project_rstadvancedsampleproject_QC2406
Dictionary_ID	QC_12
SourceDocument	EN 13670-2009
SourceSection	BeamsAndSlabs_annex_c
Description	Inclination of a beam or a slab
Result	Fail
Involved_Components	[1WrzGm1SD2ev45B_OWQ3EP]
TimestampSchedule	17/01/2022
TimestampPerformed	02/05/2022
Unit	[distance, metres]
ScalarResult	[0.02974748061, 0.00504381943]
ToleranceReference	[0.0255843513, 0.0107788946]
AuxiliaryOutputFile	[\Output\RevitSchoolCylinders\1WrzGm1SD2ev45B_OWQ3EP_Project_rstadvancedsampleproject_QC2406.ply, \Output\RevitSchoolCylinders\1WrzGm1SD2ev45B_OWQ3EP_Project_rstadvancedsampleproject_QC2406.obj]

Project_rstadvancedsampleproject_QC2416	
QC_UID	Project_rstadvancedsampleproject_QC2416
Dictionary_ID	QC_5
SourceDocument	EN 13670-2009
SourceSection	BeamsAndSlabs_a
Description	Beam-to-column location
Result	Pass
Involved_Components	[1WrzGm1SD2ev45B_OWQ3EP, 18YHwga450Mw4Fy6M5t_8F]
TimestampSchedule	17/01/2022
TimestampPerformed	02/05/2022
Unit	[distance, metres]
ScalarResult	[0.00973247801]
ToleranceReference	[0.0199999996]
AuxiliaryOutputFile	[\Output\RevitSchoolCylinders\1WrzGm1SD2ev45B_OWQ3EP_Project_rstadvancedsampleproject_QC2416.ply, \Output\RevitSchoolCylinders\1WrzGm1SD2ev45B_OWQ3EP_Project_rstadvancedsampleproject_QC2416.obj, \Output\RevitSchoolCylinders\18YHwga450Mw4Fy6M5t_8F_Project_rstadvancedsampleproject_QC2416.ply, \Output\RevitSchoolCylinders\18YHwga450Mw4Fy6M5t_8F_Project_rstadvancedsampleproject_QC2416.obj]

Figure 17: Example results obtained using the GeometricQC tool for Revit Sample Project Technical School model

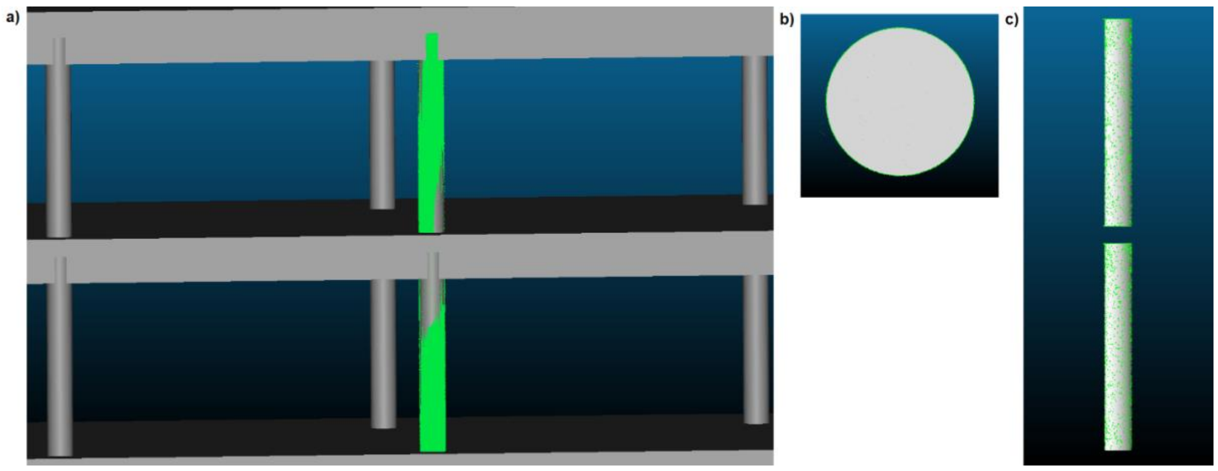


Figure 18: Example of a satisfactory geometric QC result

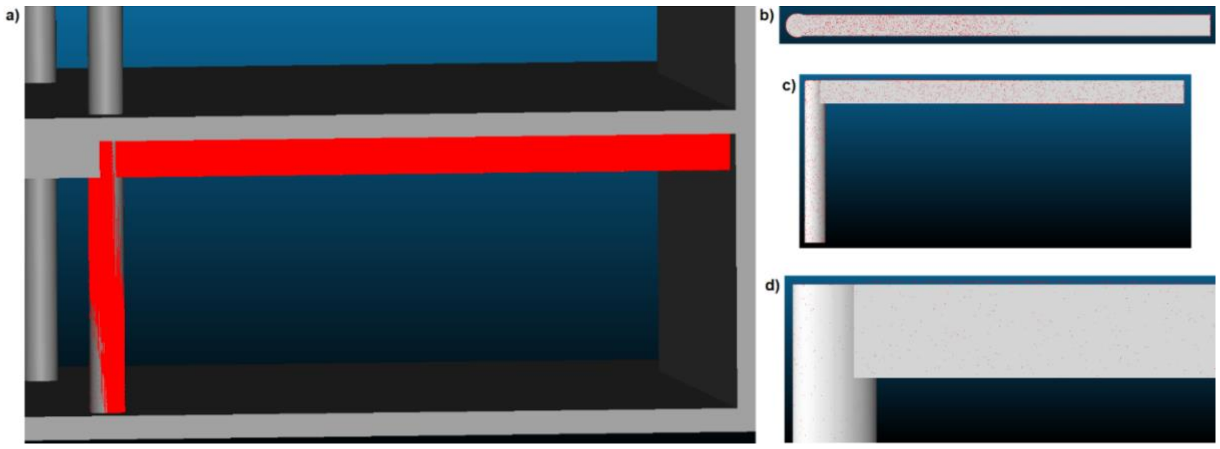


Figure 19: Example of an unsatisfactory geometric QC result

4.5 Licensing

The GeometricQC tool is free software; you can redistribute it and/or modify it under the terms of the **GNU General Public License Version 3** as published by the Free Software Foundation (https://www.gnu.org/licenses/gpl-3.0.en.html).

4.6 Installation Instructions

No complex installation is required. Binaries will be provided in the Cyberbuild Datashare collections. GeometricQC tool is a set of header-only libraries that can be included in any project just by adding the header files to the includes list.

4.7 Development and integration status

The *GeometricQC tool active geometric quality control analysis* component is currently under development, having the main basic functionalities already in place and ready to use. The *GeometricQC tool active geometric quality control analysis* component does not interact with any external component; all the interactions are within the GeometricQC tool.

It is expected new functionalities will be available in the final version of the GeometricQC tool. A summary of the new functionalities that should be covered can be found below:

- Include new geometric and dimensional tolerances verification methods for the rules and geometric relationships obtained from the concrete and steel standards.
- Code refactor and clean up.
- Auxiliary files generation and management refactor.
- Load/save as-built point cloud data in E57 format.
- Middleware interface to work with the DTP and the GeometricQC tool.

4.8 Requirements Coverage

Although the *GeometricQC tool active geometric quality control analysis* component is delivered as a back-end part of the GeometricQC tool, it already covers a couple of the requirements defined in D2.4, D2.5 and D2.1.

The functional and non-functional requirements from the COGITO System Architecture that are already covered are presented in Table 5. Functional requirement Req-1.1 is partially covered by the integration of the *BIM Element Manager,* the *Scan-vs-BIM solution*, and the Open3D library. However, the schedule part of the 4D BIM is still pending. Req-1.2 is partially achieved thanks to the *Point Cloud Matching and Segmentation* sub-component by associating the 3D points to the different structural elements. Regarding Req-1.3, the GeometricQC tool already has the capability of performing the automatic geometric QC verifications of six different tolerances defined in the concrete standard BS EN 13670:2009 [3], so this requirement can be considered achieved, despite the fact that we still require to include additional QC tolerances.

Thanks to the C++ property of using different objects in header only classes, Req-2.1, Req-2.2, and Req-2.3 are well covered within this sub-component. As previously mentioned, this set of classes can be used autonomously in any other software project, or as part of the full GeometricQC tool. In addition, its scalability is well handled using object-oriented programming, which enables straightforward addition of new functionalities and properties to each of the components.

Table 5: *GeometricQC tool active geometric quality control analysis* sub-component requirements coverage from D2.4 and D2.5

ID	Description	Type	Status
Req-1.1	Loading as planned 4D BIM and point cloud data	Functional	Partially achieved
Req-1.2	Object detection in point cloud	Functional	Partially achieved
Req-1.3	Defects' detection based on digitalised dimensional QC specifications	Functional	Achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Reusability	Non-Functional	Achieved
Req-2.3	Interoperability	Non-Functional	Achieved

Table 6 presents the stakeholders requirements that have been documented in D2.1 and are relevant to this component. COGI-CS-1 and COGI-CS-4 are covered simply by the programming language that has been selected for the development of the GeometricQC tool. COGI-QC-8 is partially achieved thanks to the inclusion of quality control rules regarding concrete structures. COGI-QC-10 is partially achieved with the QC results output file that is generated by the sub-component. This file is stored in the DTP and the results can then be visualised by the PM or QM through DigitAR and DCC. Regarding COGI-QC-11, the *GeometricQC tool active geometric quality control analysis* component allows to perform the geometric and dimensional tolerance verifications, using the list of QC instances generated by the *GeometricQC offline processing* component. COGI-QC-16 is partially achieved by using the Open3D library, which is able to handle the point cloud PLY format. The E57 format will be introduced in the 2nd release of the component. COGI-QC20 is achieved with the QC result output file that is generated, where it contains a detailed information of the results, tolerances, and auxiliary files that will be used to visualise the QC results in the DigitAR and DCC.

Table 6: *GeometricQC tool active geometric quality control analysis* component requirements coverage from D2.1

ID	Description	Type	Priority	Status
COGI-CS-1	Runs on desktop or laptop PC	• Operational	Must	Achieved
COGI-CS-4	Runs on Windows	• Operational	Must	Achieved
COGI-CS-5	Runs on Mac	• Operational	Could	Achieved
COGI-QC-8	Supports systematic quality control on earthworks, substructures, concrete works	• Performance	Should	Partially achieved
COGI-QC-10	Notifies PM of QC results at least on a weekly basis, and ideally on a daily basis	• Functional • Performance	Must	Partially achieved
COGI-QC-11	Automates QC-related activities	• Functional • Operational	Must	Achieved
COGI-QC-16	Handles point clouds standard formats (E57, PLY)	• Functional	Must	Partially achieved
COGI-QC-20	Issues QC results reports that include: links to the BIM model with annotations of the survey result	• Functional • Design constraint	Should	Achieved

4.9 Assumptions and Restrictions

The *GeometricQC tool active geometric quality control analysis* component has a number of assumptions and restrictions, which are presented in the following:

- BIM elements and point clouds are expected to have the vertical along the +Z axis using the right-hand side coordinate frame.
- The coordinate reference frame within the entire GeometricQC tool uses global coordinates, hence, the generated output files contain coordinates information in the same global space.
- The measures are stored using the international metric system, and the distance measurement unit used in the sub-component is the metre [m].
- The input as-built data has to be pre-registered in the same global coordinate system as the BIM file, i.e. the project's coordinate system, minimising the registration error to ensure an accurate point cloud matching and segmentation.
- It currently only supports *.ply* files. Support for the E57 file format shall be provided in the 2nd version.

- It currently only handles one input point cloud file to perform the matching. In the 2nd version, it will include support for multiple (pre-registered) point clouds as input.
- It has been developed to be part of the GeometricQC tool, so no standalone programme is supposed to be executed for this component.
- The *GeometricQC offline preprocessing* component has already been executed and all the internal output files have been generated and are available as input for the *GeometricQC tool active geometric quality control analysis* component.

5 Conclusions

The GeometricQC tool is designed to be used in two different phases. During the planning phase, all the offline pre-processing of the as-planned data is performed, whereas in the construction phase, the active geometrical analysis is iteratively running. The tool uses as input the as-planned data in the form of the as-planned 4D BIM model (digital semantically rich 4D model of the building/asset created by the design and planning team), and the as-built point cloud data obtained through laser scanning surveys in different steps of the construction phase. The BIM model data are provided by and extract from an IFC format file, while the point cloud data must be pre-registered in the coordinate system of the project (i.e. the same coordinate system as the BIM model).

As documented in this deliverable (D5.5), the core functionalities of the GeometricQC tool are delivered by two components (reflecting the two different processing steps). The first component, namely the *GeometricQC offline processing* component, performs all the as-designed data analysis, by obtaining the 3D geometry from the IFC file, and generating a geometric relationship graph structure containing all the required relationships and used to generate the list of QC instances that must be verified during the project's construction phase. The second component, the *GeometricQC tool active geometric quality control analysis* component performs the active QC analysis of the as-built data during the construction phase. The component uses the *Scan-vs-BIM solution* (detailed in D5.1) to match and segment the as-built data of the structural elements of interest and performs geometrical and dimensional QC processing to analyse whether the structural elements satisfy the defined geometric tolerances. The result of the active phase shall be submitted back to the Digital Twin Platform, where it is distributed to the other relevant COGITO components and the relevant stakeholders. These components have also been designed to be modular, and scalable, allowing the creation of a toolbox, which can be used in any other project.

References

[1] J. Zhang and M. El-Gohary, “Automated Extraction of Information from Building Information Models into a Semantic Logic-Based Representation,” in *Computing in Civil Engineering 2015*, ASCE, 2015, pp. 173-180.

[2] J. Zhang and N. M. El-Gohary, “Semantic-based logic representation and reasoning for automated regulatory compliance checking,” *Journal of Computing in Civil Engineering,* vol. 1, no. 31, 2017.

[3] BS EN 13670:2009, “Execution of concrete structures,” 2010.

[4] BS EN 1090-2, “Execution if steel structures and aluminion structures. Technical requirements for steel structure,” 2018.

[5] Autodesk, “Revit Sample Project Files,” Autodesk, [Online]. Available: https://knowledge.autodesk.com/support/revit/getting-started/caas/CloudHelp/cloudhelp/2022/ENU/Revit-GetStarted/files/GUID-61EF2F22-3A1F-4317-B925-1E85F138BE88-htm.html. [Accessed 29 04 2022].

[6] M. Bastian, S. Heymann and M. Jacomy, “Gephi: An open source software for exploring and manipulating networks,” in *Internation AAAI Conference on Weblogs and Social Media*, 2009.

COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958310