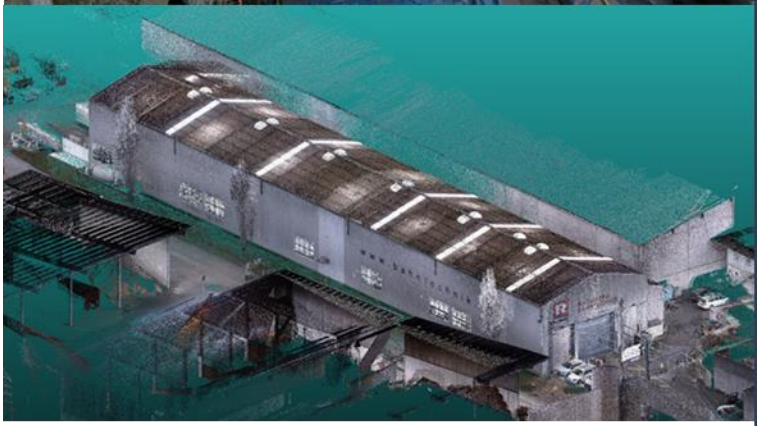


COGITO

CONSTRUCTION PHASE
DIGITAL TWIN MODEL

cogito-project.eu

D5.6 – BIM-based Standard Test Methods for Geometric Quality Control v2

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

D5.6 – BIM-based Standard Test Methods for Geometric Quality Control v2

Dissemination Level:	Public
Deliverable Type:	Report
Lead Partner:	UEDIN
Contributing Partners:	UEDIN, CERTH
Due date:	31-10-2022
Actual submission date:	27-10-2022

Authors

Name	Beneficiary	Email
Martín Bueno	UEDIN	martin.bueno@ed.ac.uk
Frédéric Bosché	UEDIN	f.bosche@ed.ac.uk
Wojciech Boncela	UEDIN	W.Boncela@sms.ed.ac.uk

Reviewers

Name	Beneficiary	Email
Georgios Lilis	UCL	g.lilis@ucl.ac.uk
Raúl García Castro	UPM	rgarcia@fi.ump.es

Version History

Version	Editors	Date	Comment
0.1	M. Bueno	03.08.2022	Table of Content
0.2	M. Bueno	07.10.2022	First draft version of the document
0.3	W. Boncela, M. Bueno	13.10.2022	Added the communication component section
0.4	F. Bosché, M. Bueno	17.10.2022	Refinements to the draft
0.5	G. Lilis, R. G. Castro	21.10.2022	Internal review
0.6	M. Bueno, F. Bosché	24.10.2022	Review comments addressed
0.7	UEDIN	25.10.2022	Final version
1.0	UEDIN, Hypertech	27.10.2022	Submission to the EC portal

Disclaimer

Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use, which may be made, of the information contained therein.

Executive Summary

The COGITO Deliverable D5.6 “BIM-based Standard Test Methods for Geometric Quality Control v2” documents the COGITO Geometric Quality Control solution and presents the second and final 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 detailed information of each rule used, its tolerances and the results obtained from all rule instances applied 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) when it extracts all the relevant information from the as-planned 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) when 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 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 second and final release, the COGITO GeometricQC tool implements all 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
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IFC	Industry Foundation Classes
JSON	JavaScript Object Notation
OBJ	Object file
QC	Quality control
STL	Standard Triangle/Tessellation Language
TLS	Terrestrial Laser Scanning
UID	Unique Identifier
XML	Extensible Markup Language
WP	Work Package

1 Introduction

1.1 Scope and Objectives of the Deliverable

This deliverable reports on the work conducted between M12 and M24 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 checking, 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 second release of the GeometricQC tool solution, focusing on its three 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, computing their geometric relationships and cross-referencing those 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 as the project progresses.
- **GeometricQC tool communication component:** this component is in charge of maintaining the communication with the Digital Twin Platform (DTP), monitoring the incoming of new processing requests, querying all the necessary input information, and returning the geometric quality control results to it, for further use by the different COGITO tools (e.g. DigitAR and DCC).

These three components provide the core functionalities of the GeometricQC tool. They integrate the sub-components introduced in "*D5.2: Innovative Scan-vs-BIM- based Geometric QC component v2*".

1.2 Relation to other Tasks and Deliverables

Table 1 depicts the relations of this document to other deliverables within the COGITO project; the latter should be considered along with this document for further understanding of its contents. This deliverable reports the second and final version (v2) of the GeometricQC tool. The GeometricQC tool is principally used to support the Use Case 2.1 (UC2.1).

Table 1: Relation to other Tasks and Deliverables

Del. Number	Deliverable Title	Relations and Contribution
D2.1	Stakeholder requirements for the COGITO system	Analysis of the end-user requirements in order to create the necessary inputs for defining the COGITO GeometricQC tool and its interactions with other components, along with a thorough description of the business scenarios, use cases and system requirements tailored to the project's goals and therefore setting the skeleton for the geometricQC tool development.
D2.5	COGITO system architecture v2	The second version of the COGITO architecture report that includes updates on the specifications of the GeometricQC tool (hardware/software requirements, programming language(s), development status, functional/non-functional requirements,

		inputs/outputs, along with the format, method, endpoint and protocol for each data type and interface).
D3.3	COGITO Data Model and Ontology Definition and Interoperability Design 2	Documentation of the second version of the COGITO ontologies (available online in the COGITO ontology portal), particularly the resources relevant to the GeometricQC tool.
D5.2	Innovative Scan-vs-BIM-based Geometric QC component v2	Second version of the Scan-vs-BIM solution. This deliverable reports the solution developed to perform automatic as-built matching with the as-planned geometry. The deliverable reports the sub-components, including the technology stack, input, outputs, integration status and usage examples. The Scan-vs-BIM solution presented in this deliverable is strongly connected with the GeometricQC tool as all of its sub-components were developed and are used by the later.
D5.5	BIM-based Standard Test Methods for Geometric Quality Control v1	First version of the GeometricQC tool. For a detailed list of updates from this deliverable, please see section 1.4.
D7.2	Digital Twin Platform Design & Interface Specification v2	First version of the Digital Twin Platform's detailed architecture, providing details on the suppoerted communication protocols for interfacng with the GeometricQC tool and its timeseries database for storing geometric QC results and relevant data.

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.

1.3 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, 4 and 5 describe the *GeometricQC tool offline processing*, the *GeometricQC tool active geometric quality control analysis* and *GeometricQC tool communication* components. For each component, we provide 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. Section 6 concludes the deliverable.

1.4 Updates from the first version of the Deliverable

The second and final version of the GeometricQC tool presents a few new features in comparison with the first version of the solution presented in the first version of the deliverable (D5.5).

First of all, the GeometricQC tool now includes a total of 17 new QC dictionary rules (QC_16 to QC_30), 15 of them corresponding to steel elements and obtained from the steel standard EN 1090-2 [1], and 2 of them corresponding to concrete elements and obtained from the concrete standard EN 13670-2009 [2], making a total of 30 QC dictionary rules combining concrete and steel materials. These rules relate to building elements such as columns and beams, but also rails. From these 30 QC dictionary rules, 18 are included for the automatic QC tolerance verification. Some additional QC tolerances from the QC dictionary rules will be implemented based on the demands arising during the pre-validation and validation activities.

Parallel to the inclusion of the new steel rules, the computation of the corresponding geometric relationships for 18 QC rules are also included in the new version.

As an additional feature, the GeometricQC tool now includes the full integration of the 4D BIM as input, by extracting the schedule information for the different elements. This information is necessary to identify which QC rule instances can be checked each time the GeometricQC tool is informed of the completion of the construction of elements.

As part of the integration of the GeometricQC tool and the rest of the COGITO tools, the new version of the deliverable contains the detailed information of the different input and output exchanges between the DTP and the GeometricQC tool in JavaScript Object Notation (JSON) file format.

Finally, the new GeometricQC communication component is detailed, that manages the communication between the GeometricQC tool and the DTP.

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 support the Use Case 2.1 (UC2.1). The GeometricQC tool aims to conduct automatic geometric QC by comparing the as-planned and as-built geometry. The as-planned 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 defined in the Scan-vs-BIM solution in "*D5.2: Innovative Scan-vs-BIM-based Geometric QC component v2*". As already stablished in the COGITO system architecture, the GeometricQC tool aims to use open formats (such as IFC for the as-planned data, and PLY, E57, etc. for the as-built data), targeting increased interoperability to support effective collaborations among the range of parties involved in project delivery.

The core idea of the GeometricQC tool is akin to follow an Automated Rule Checking (ARC) approach [3] [4]. The GeometricQC tool is divided in two (3) main components:

- **GeometricQC tool offline processing component:** this component is in charge of extracting all the structural elements from the as-planned data, obtain the geometric relationships between them, and cross reference these with the QC dictionary to obtain the full set of QC instances 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 component:** 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 (using the tool/functionality described in "*D5.2: Innovative Scan-vs-BIM-based Geometric QC component v2*") and performs the scheduled geometric control check defined by the offline processing component.
- **GeometricQC tool communication component:** this component is the nexus between the GeometricQC tool and the DTP (and through it the rest of the COGITO components and tools and the different stakeholders). The GeometricQC tool communication component must be in constant connection with the DTP, listening to when a new geometric QC job is required by the project, it must collect all the necessary information from the DTP, trigger the relevant components (active and/or offline processing), and format and return the generated output back to the DTP.

Figure 1 presents a high-level functional diagram of the tool, highlighting the abovementioned components that are presented in more detail in Sections 3, 4 and 5.

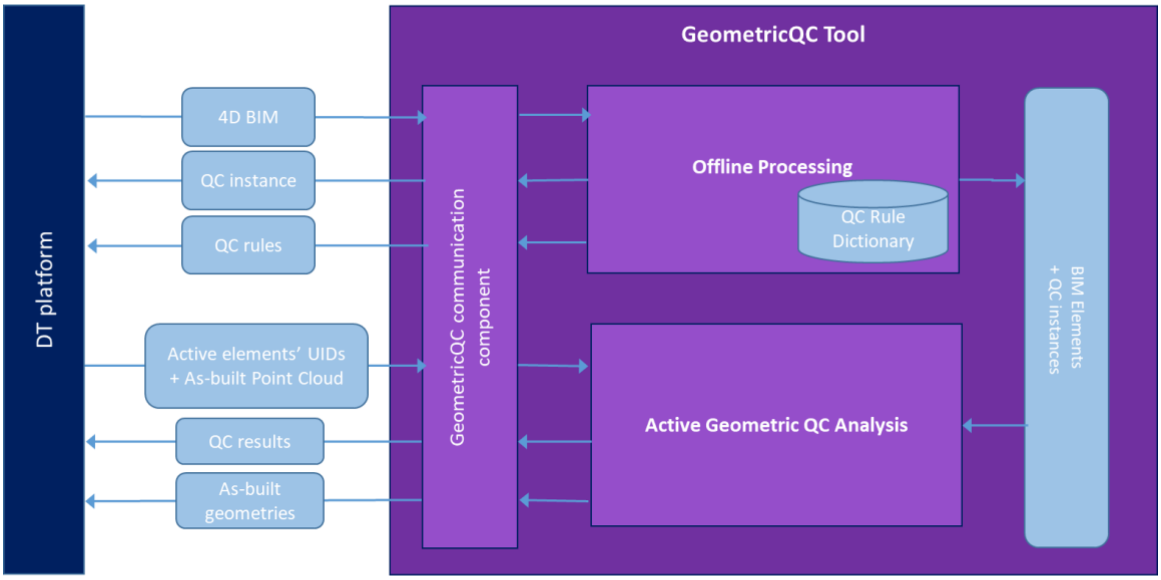

DT platform
4D BIM
QC instance
QC rules
Active elements' UIDs + As-built Point Cloud
QC results
As-built geometries
GeometricQC Tool
GeometricQC communication component
Offline Processing
QC Rule Dictionary
Active Geometric QC Analysis
BIM Elements + QC instances

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-planned 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 exist in the form of digital rules that will be verified throughout the project. Selected specifications for concrete works [2] and steel works [1] have been identified and digitally translated. These rules must define the context within which the specification applies (e.g. what type of object it applies to), the quantity to be measured (e.g. distance), and the tolerance that applies to that measured quantity. Given those rules, the GeometricQC tool processes the as-planned BIM model and the linked project schedule (i.e. 4D model) to identify where each rule applies (i.e. where a specification applies) and when the corresponding QC instance task should be conducted. A rule context is defined by the type of object(s), material(s) and, if the context involves more than one object, the relationship that these objects must have (e.g. being connected). The GeometricQC tool thus identifies where each rule applies by extracting all elements from the BIM model and by computing their relationships. The result of this process is a network graph of nodes (the elements) and their connections (the relationships). The GeometricQC then defines a geometric QC instance at each location in the graph where the context of the QC rule is found.

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 considered immutable, and is 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 *GeometricQC offline processing* component is quite simple to use and include in any software project. It only requires the file path of the BIM model (IFC file), the schedule activity relating to each element, and the output folder where all the generated information must be stored. As it can be seen in Figure 2, the component is composed of different sub-components that altogether enable the generation of the list of geometric QC instances. The different sub-components and their functionalities are detailed below.

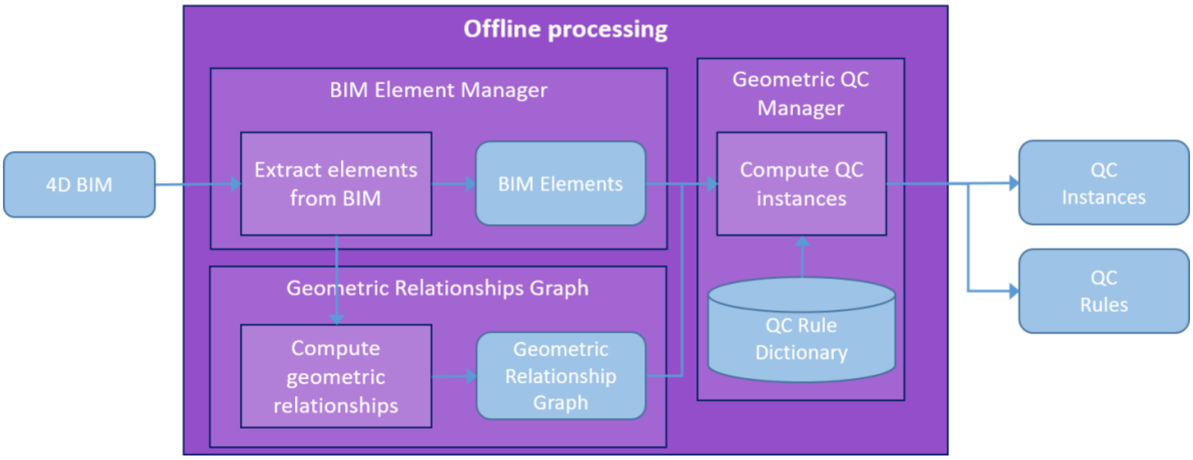


Figure 2: GeometricQC offline processing detailed workflow

3.1.1 BIM Element Manager sub-component

As explained in D5.2, 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 to D5.2 “*Innovative Scan-vs-BIM- based Geometric QC component v2*”. Herein, we focus on reporting additional functionalities to the BIM Element Manager not covered in D5.2, including:

- **Obtain the structural element material from the IFC file:** the **IFCManager** class 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 because the contexts of geometric QC rules often refer to the type of material to which they apply.
- **Obtain and store the geometric relationship graph:** with the **RelGraph** class (see 3.1.3 for more detail), the **BIMElementManager** is capable of storing all BIM model elements of interest (structural elements in the context of COGITO) as nodes and their geometric relationships as edges in a graph structure. This graph is useful to perform queries to identify instances of the contexts defined in the rules contained in the *Quality Control rule dictionary* (Section 3.1.2).
- **Obtain and store the as-planned completion timestamp:** the **IFCManager** class includes the method to retrieve the activity from the construction schedule when each **BIMElement** is to be built, extract its scheduled completion date, and assign it to the **BIMElement** object.

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

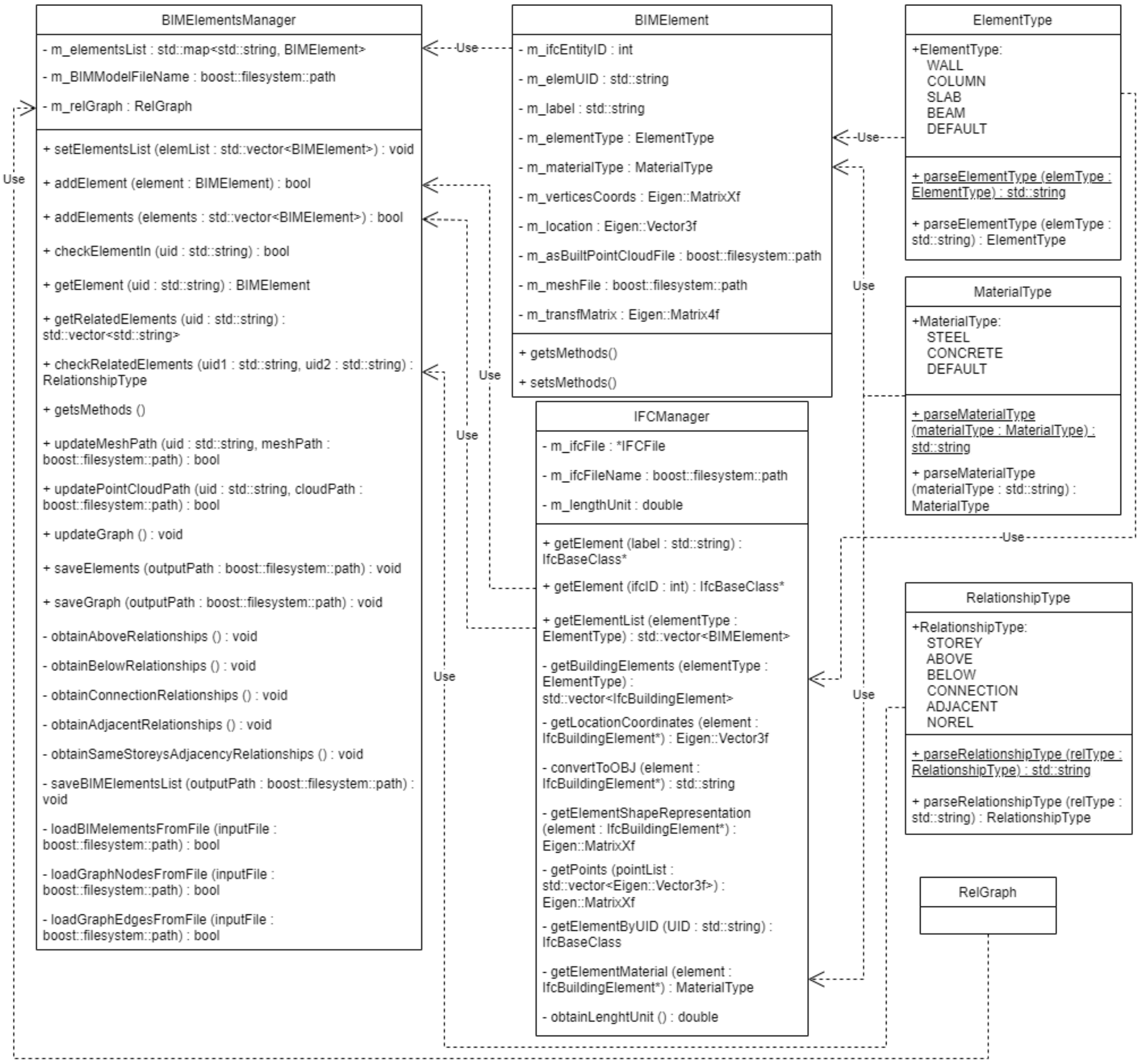

BIMElementsManager
- m_elementsList : std::map<std::string, BIMElement>
- m_BIMModelFileName : boost::filesystem::path
- m_relGraph : RelGraph
+ setElementsList (elemList : std::vector<BIMElement>) : void
+ addElement (element : BIMElement) : bool
+ addElements (elements : std::vector<BIMElement>) : 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
- obtainAboveRelations () : void
- obtainBelowRelations () : void
- obtainConnectionRelations () : 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
- m_transfMatrix : Eigen::Matrix4f
+ getsMethods()
+ setsMethods()
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
- convertToOBJ (element : IfcBuildingElement*) : std::string
- getElementShapeRepresentation (element : IfcBuildingElement*) : Eigen::MatrixXf
- getPoints (pointList : std::vector<Eigen::Vector3f>) : Eigen::MatrixXf
- getElementByUID (UID : std::string) : IfcBaseClass
- getElementMaterial (element : IfcBuildingElement*) : MaterialType
- obtainLenghtUnit () : double
ElementType
+ElementType:
WALL
COLUMN
SLAB
BEAM
DEFAULT
+ parseElementType (elemType : ElementType) : std::string
+ parseElementType (elemType : std::string) : ElementType
MaterialType
+MaterialType:
STEEL
CONCRETE
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
RelGraph
Use
Use
Use
Use
Use
Use
Use
Use
Use

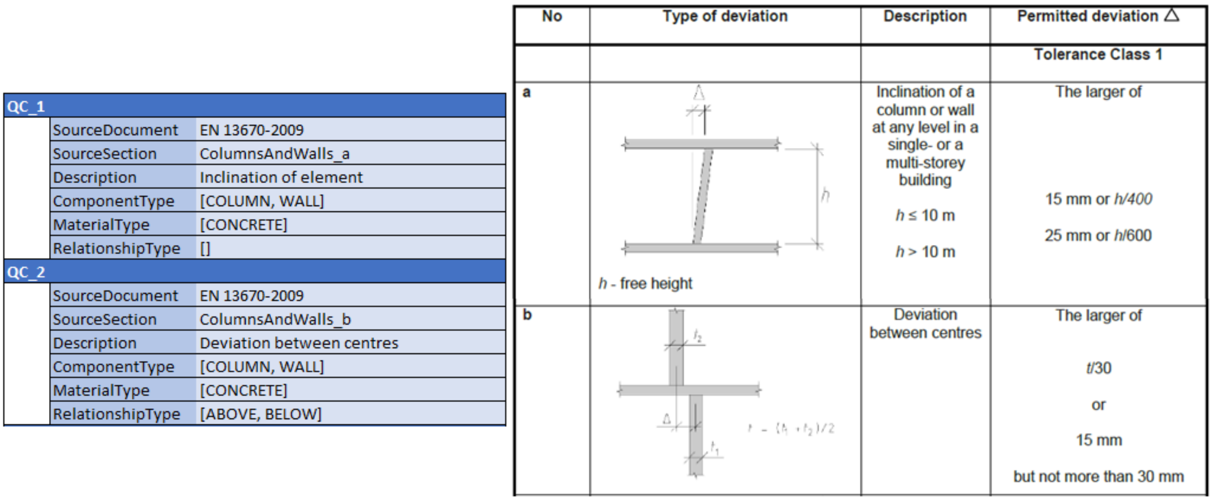
Figure 3: BIM Element Manager sub-component classes and its relationships

3.1.2 Quality Control Rule Dictionary

The *Quality Control Rule Dictionary* is the sub-component that allows the user to establish and define the different QC rules that are set for the given project. Each rule defines the *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 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 also contains a *description* that describes the source of the rule (e.g. established standard) and a text description to ease understanding. The following summarise the content of each rule

- **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 [2].



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	*h* - free height	Inclination of a column or wall at any level in a single- or a multi-storey building *h* ≤ 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, 17 rules have been digitally translated from EN 13670-2009 [2] and 13 rules have been digitally translated from EN 1090-2:2018 [1] and added to the Quality Control Dictionary:

- **QC_1: Inclination of a column/wall** *(EN 13670-2009 10.4 Columns and Walls, No a)*
- **QC_2: Deviation between centres** *(EN 13670-2009 10.4 Columns and Walls, No b)*
- **QC_3: Curvature of a column/wall between adjacent storey levels** *(EN 13670-2009 10.4 Columns and Walls, No c)*

- **QC 4: Location of a column/wall at any storey level with respect to base level** *(EN 13670-2009 10.4 Columns and Walls, No d)*
- **QC 5: Location of a beam-to-column connection measured relative to the column** *(EN 13670-2009 10.5 Beams and Slabs, No a)*
- **QC 6: Position of bearing axis of support** *(EN 13670-2009 10.5 Beams and Slabs, No b)*
- **QC 7: Cross-sectional dimensions** *(EN 13670-2009 10.6 Sections, No a)*
- **QC 8: Lap-joints** *(EN 13670-2009 10.6 Sections, No c)*
- **QC 9: Free space between adjacent columns/walls** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No c)*
- **QC 10: Horizontal straightness of beams** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No a)*
- **QC 11: Distance between adjacent beams** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No b)*
- **QC 12: Inclination of a beam/slab** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No c)*
- **QC 13: Level of adjacent beams** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No d)*
- **QC 14: Level of adjacent floors at supports** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No e)*
- **QC 15: Orthogonality of a cross-section** *(EN 13670-2009 Annex G – G.10.6 Sections, No a)*
- **QC 16: Column position in plane** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No a)*
- **QC 17: Wall position in plane** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No b)*
- **QC 18: Column height relative to adjacent levels** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B15 - 2)*
- **QC 19: Beam slope** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B15 - 3)*
- **QC 20: Levels of adjacent beams** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B15 -6)*
- **QC 21: Spacing between beam centrelines** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B16 - 1)*
- **QC 22: Location at columns** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B16 - 2)*
- **QC 23: Inclination of columns** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B17 – 1 and Table B17-2)*
- **QC 24: Column location** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B20 - 1)*
- **QC 25: Column spacing** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B20 - 3)*
- **QC 26: Location of rail in plan** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B22 - 1)*
- **QC 27: Local alignment of rail** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B22 - 2)*
- **QC 28: Level of rail** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B22 - 3)*
- **QC 29: Relative levels of rails on the two sides of a runway** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B22 - 6)*
- **QC 30: Spacing over span S between centres of crane rails** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B22 - 7)*

The QC Rule 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 Rule Dictionary sub-component includes two main object classes that are interconnected:

- **GeomQC class:** this class is used to describe the QC rules. The class contains all the information stored in the dictionary's entries: the rule *description* (i.e. description, source document and source section) and rule *context* (i.e. elements' materials, types, and relationships).
- **GeomQCDictionary class:** this class contains the different QC Rule Dictionary entries stored in a map structure to facilitate the access and queries to and from it.

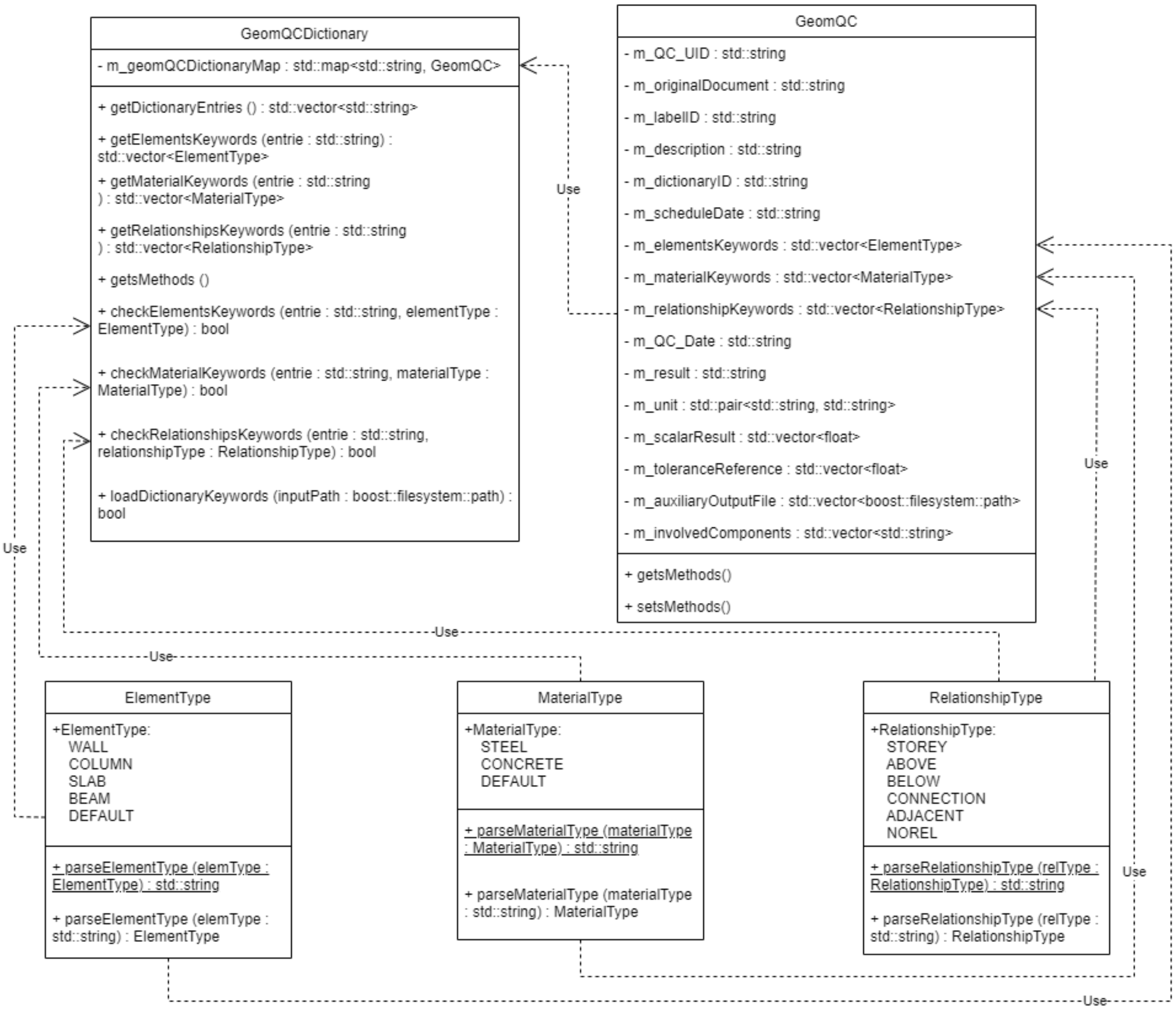

GeomQCDictionary
- m_geomQCDictionaryMap : std::map<std::string, GeomQC>
+ getDictionaryEntries () : std::vector<std::string>
+ getElementsKeywords (entrie : std::string) : std::vector<ElementType>
+ getMaterialKeywords (entrie : std::string) : std::vector<MaterialType>
+ getRelationshipsKeywords (entrie : 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
ElementType
+ElementType:
WALL
COLUMN
SLAB
BEAM
DEFAULT
+ parseElementType (elemType : ElementType) : std::string
+ parseElementType (elemType : std::string) : ElementType
MaterialType
+MaterialType:
STEEL
CONCRETE
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* sub-component creates and manages a geometric network graph structure that is created by the GeometricQC tool from its interpretation of the BIM model. In Section 3.1.1 we saw how the different elements are extracted from the BIM model in IFC format. Despite the fact that IFC files contain all the element types and materials, they do not normally contain the geometric relationships between them. Therefore, the *Geometric Relationships Graph* sub-component is designed to detect geometric relationships of interest (those defined in the context of each QC Rule) and store them in a data structure to facilitate their interpretation. A network graph structure is used for this, where each node 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 relationship types, as well as other or different properties and connections.

The different geometric relationships that have been considered so far are defined by the contexts of the rules selected from EN 13670:2009 [2] and EN 1090-2:2018 [1], and 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/level.
- **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 elements of type "wall" and the geometric relationships between them.

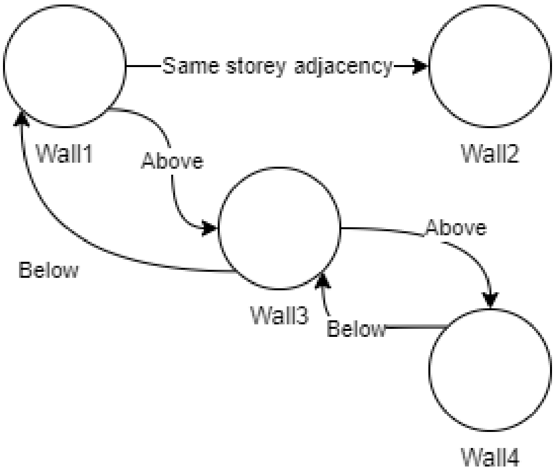


Figure 6: Geometric Relationships Graph example

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 an 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. The information required to describe a connection involves the IDs of the from and to nodes, 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 needs arise during the pre-validation and validation activities of COGITO (WP8).
- **RelGraph class:** this class is the container of the network graph data structure. 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.

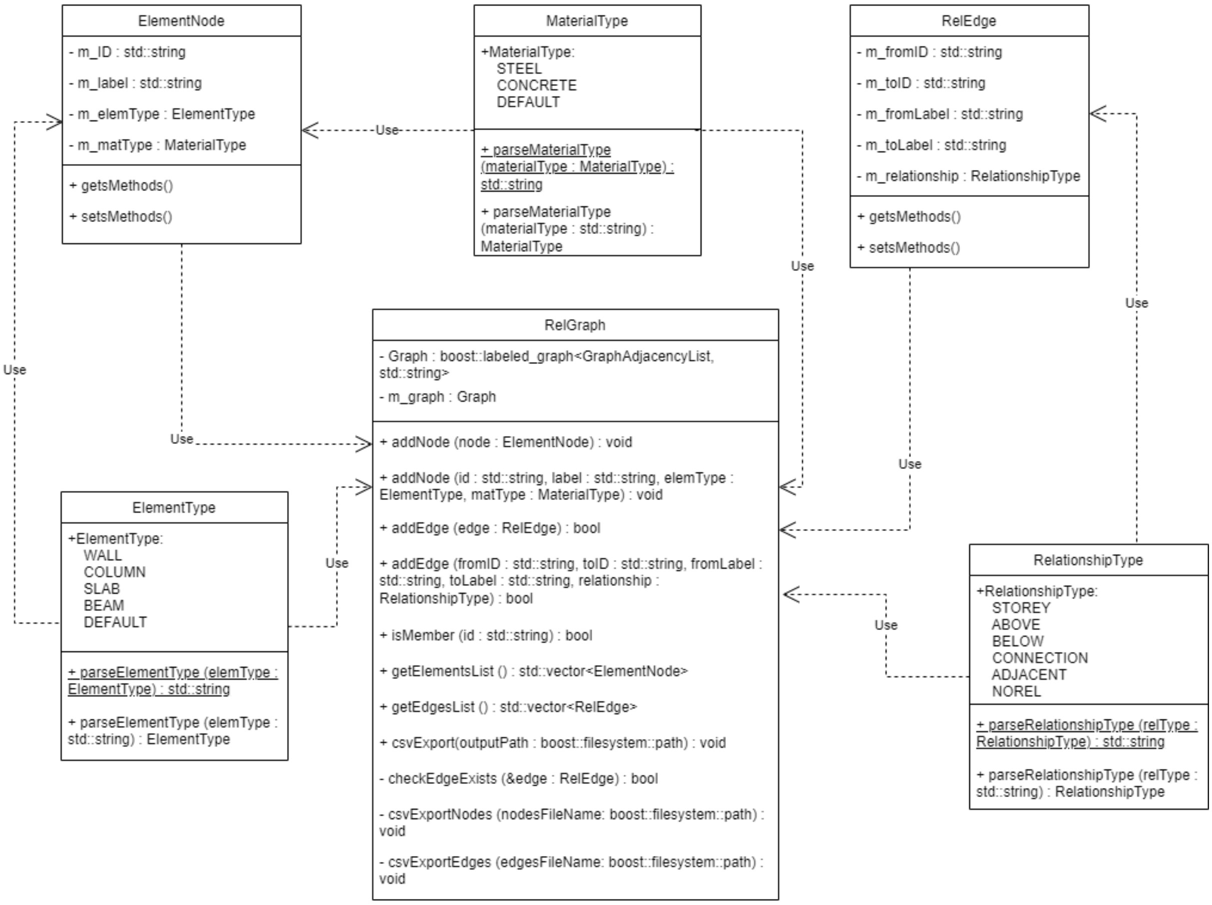

ElementNode
- m_ID : std::string
- m_label : std::string
- m_elemType : ElementType
- m_matType : MaterialType
+ getsMethods()
+ setsMethods()
MaterialType
+MaterialType:
STEEL
CONCRETE
DEFAULT
+ parseMaterialType (materialType : MaterialType) : std::string
+ parseMaterialType (materialType : std::string) : 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
Use
Use
Use
Use
Use
Use
Use

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

3.1.4 Geometric Quality Control Manager

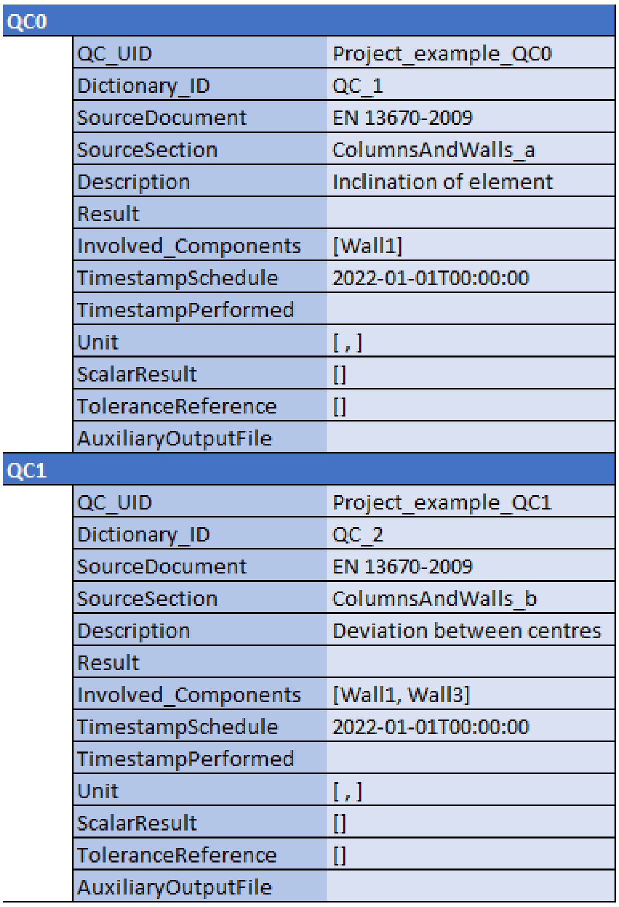
Up to this point, the GeometricQC offline processing tool sub-components are set to (1) extract all the relevant information from the BIM model and the QC Dictionary, and (2) prepare the data structures containing all that information. The *Geometric QC Manager* is the sub-component that cross-references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

The Geometric QC Manager cross references the *BIM Element Manager* (Section 3.1.1) and the *Geometric Relationships Graph* (3.1.3) to verify the contexts of each rule (i.e. involving the same *Elements,* and *Material Types* and *Relationship Types*) in the *Geometric QC Dictionary* (3.1.2) and in case of a satisfactory one, it creates the relevant QC instance. Figure 8 shows an example of two different QC instances obtained for the same QC Dictionary entries from Figure 4 and the relationship graph from Figure 6. For ease of understanding by the reader, the QC instances include the *description* fields of the QC rules related to them, to which are then added the results of the QC execution, such as the QC result and the date at which it was performed. The *Geometric QC Manager* also checks the construction schedule timestamp stored in the *BIM Element Manager* for each of the elements, to automatically populate the scheduled *TimestampSchedule* field of each QC rule instance. This field is relevant for the active geometric QC analysis component (Section 4), where it only considers and executes a QC instance tolerance verification if the elements defined in its context have all been constructed.

Additionally, the Geometric QC Manager contains the necessary methods to export the output JSON files to be used by the GeometricQC Communication component to be sent to the DTP.

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. In addition to what it is presented in 3.1.2, the class also contains information relative to the execution of the QC verification, such as: the schedule timestamp, the timestamp when the rule was verified, the tolerance values, and the scalar result values.
- **GeomQCManager class:** this class is in charge of storing all the QC instances and triggers 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	2022-01-01T00:00:00
	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	2022-01-01T00:00:00
	TimestampPerformed	
	Unit	[,]
	ScalarResult	[]
	ToleranceReference	[]
	AuxiliaryOutputFile	

Figure 8: Geometric QC instances list example

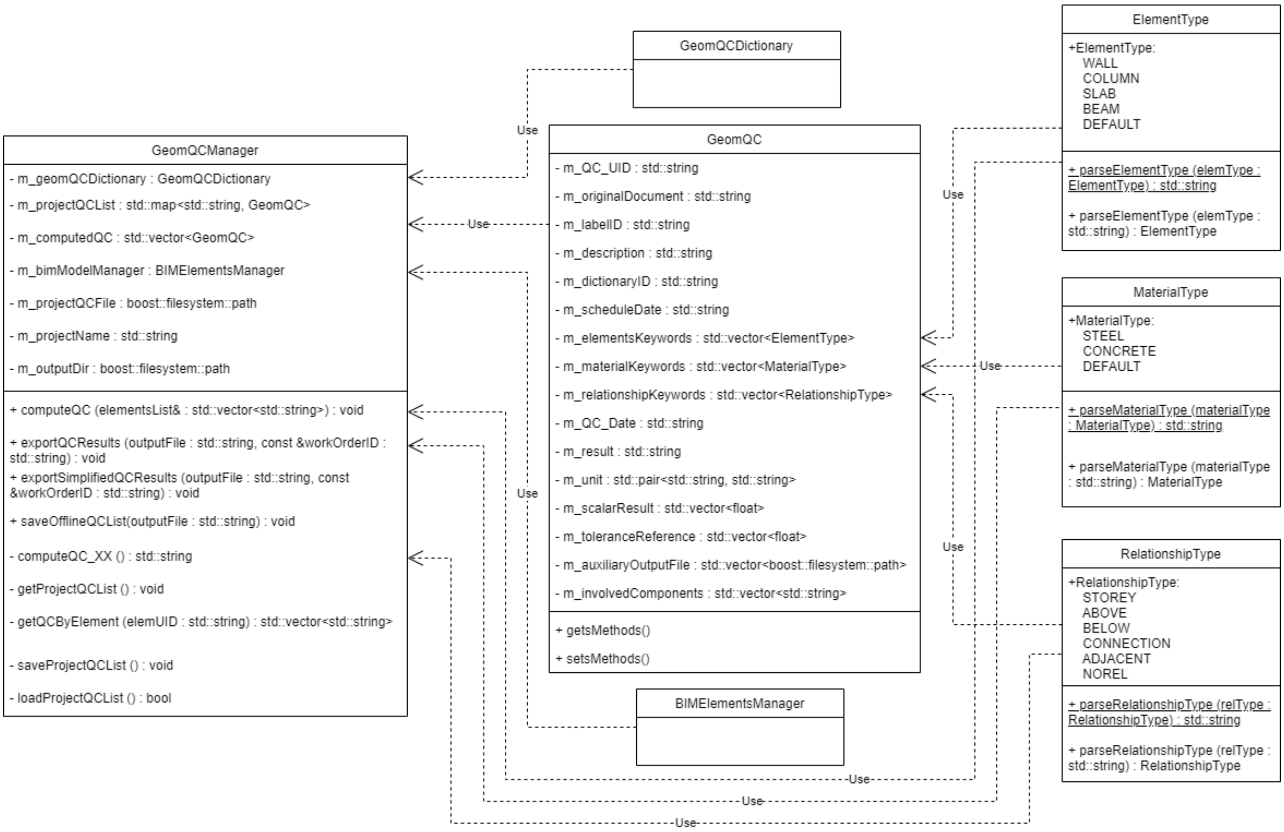

GeomQCDictionary
ElementType
+ElementType:
WALL
COLUMN
SLAB
BEAM
DEFAULT
+ parseElementType (elemType : ElementType) : std::string
+ parseElementType (elemType : std::string) : ElementType
GeomQCManager
- m_geomQCDictionary : GeomQCDictionary
- m_projectQCList : std::map<std::string, GeomQC>
- m_computedQC : 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 (outputFile : std::string, const &workOrderID : std::string) : void
+ exportSimplifiedQCResults (outputFile : std::string, const &workOrderID : std::string) : void
+ saveOfflineQCList(outputFile : std::string) : 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()
MaterialType
+MaterialType:
STEEL
CONCRETE
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
BIMElementsManager
Use
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 2: 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

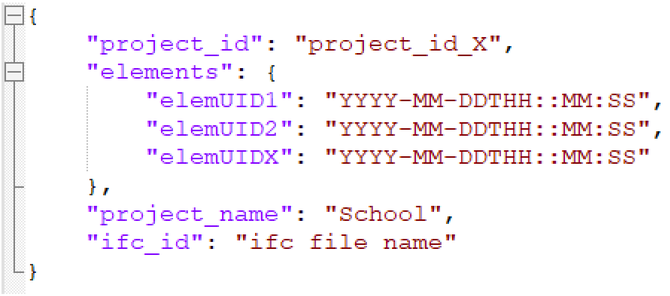
3.3 Input, Output and API Documentation

3.3.1 Inputs

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:

- **Project's Name/ID:** the construction project name or ID given by the user or the DTP. This project identification is used to create a corresponding internal output folder where to store all the generated data by the GeometricQC tool.
- **IFC file path**: file containing the BIM model of the project in IFC format. The GeometricQC tool will create a *BIM Element Manager* object using the IFC file path as input parameter. Alternatively, in the standalone case, it has to be used as input parameter at object creation. The sub-component loads the IFC file and extracts all the BIM elements from it and stores them in the manager. The IFC file should be from version 4 or over.
- **As-planned element completion dates file path:** path to the JSON file with the original as-planned completion dates of all BIM elements.

When the *GeometricQC tool offline processing* component is triggered by the *GeometricQC communication* component, are encoded in a JSON file as can be seen in Figure 10. The element UIDs must be the same UIDS as in the IFC file. In all cases the input information must follow the API described in 3.3.3.



```
{
    "project_id": "project_id_X",
    "elements": {
        "elemUID1": "YYYY-MM-DDTHH::MM:SS",
        "elemUID2": "YYYY-MM-DDTHH::MM:SS",
        "elemUIDX": "YYYY-MM-DDTHH::MM:SS"
    },
    "project_name": "School",
    "ifc_id": "ifc file name"
}
```

Figure 10: 4D BIM input template

3.3.2 Outputs

The GeometricQC offline processing component produces a couple of outputs, both for internal use and to be shared with the DTP:

- **Project's QC dictionary rules:** the list of rules that appear in the dictionary, are also stored in a JSON file to be sent to the Digital Twin Platform (DTP).
- **Project's QC instances:** the full list of QC instances obtained during the offline pre-processing phase. The QC instances are sent to the DTP. This JSON file only contains the relevant information that is available at the pre-processing phase, i.e. it doesn't contain the results and tolerance values, since the QC instances have not been verified yet, and that information is irrelevant at this stage.

The next sub-section shows the different file templates that are used for the correct execution of the *GeometricQC tool offline processing* component.

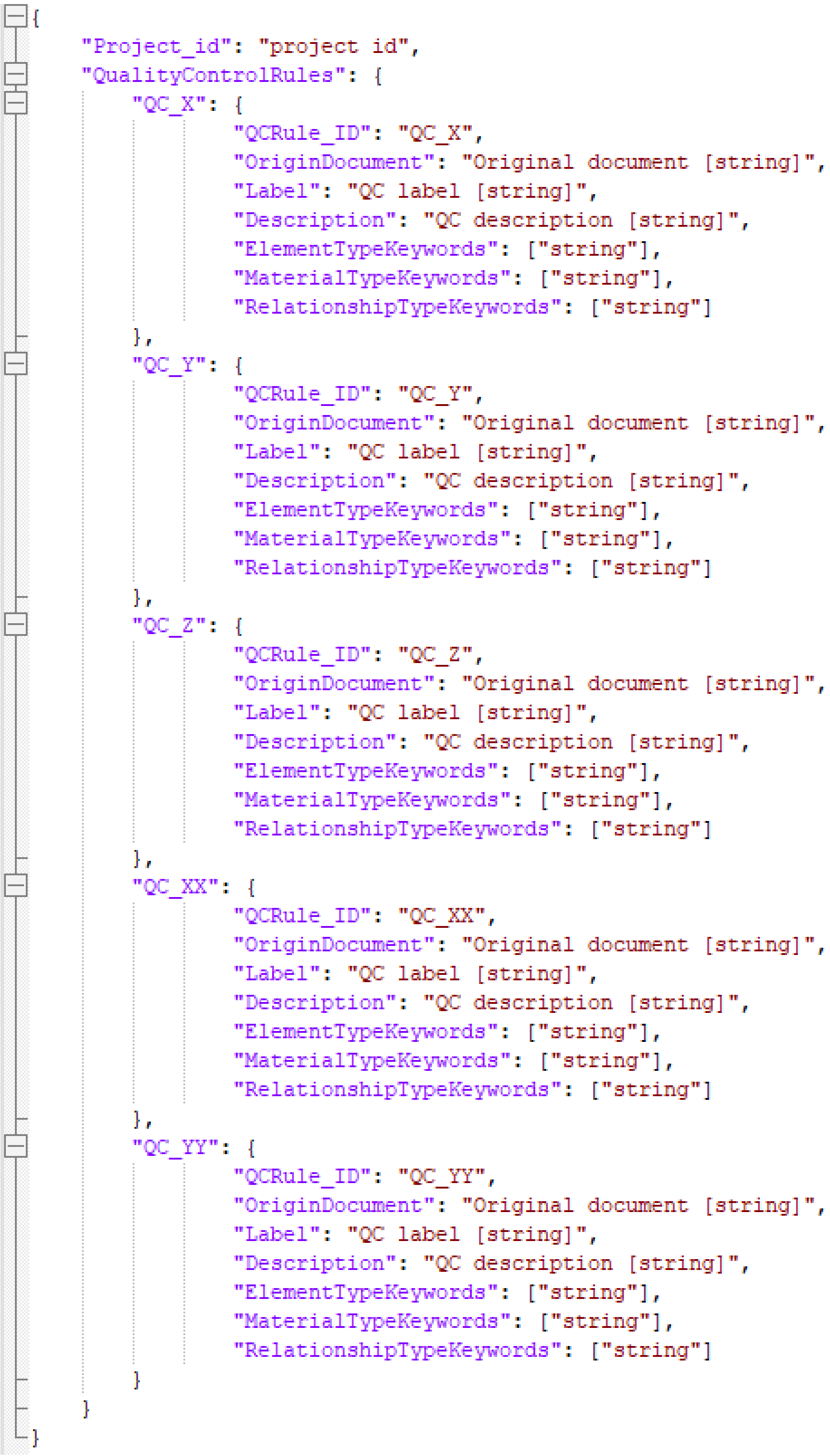
3.3.3 API Documentation

The *GeometricQC tool offline processing* component is executed via the GeometricQC tool communication component using basic command lines executions, with following format:

```
"COGITO_GeomQC.exe -ToolMode offline -projectID ProjectID -IFCfile IFCfile.ifc –scheduleFile ProjectIDschedule.json"
```

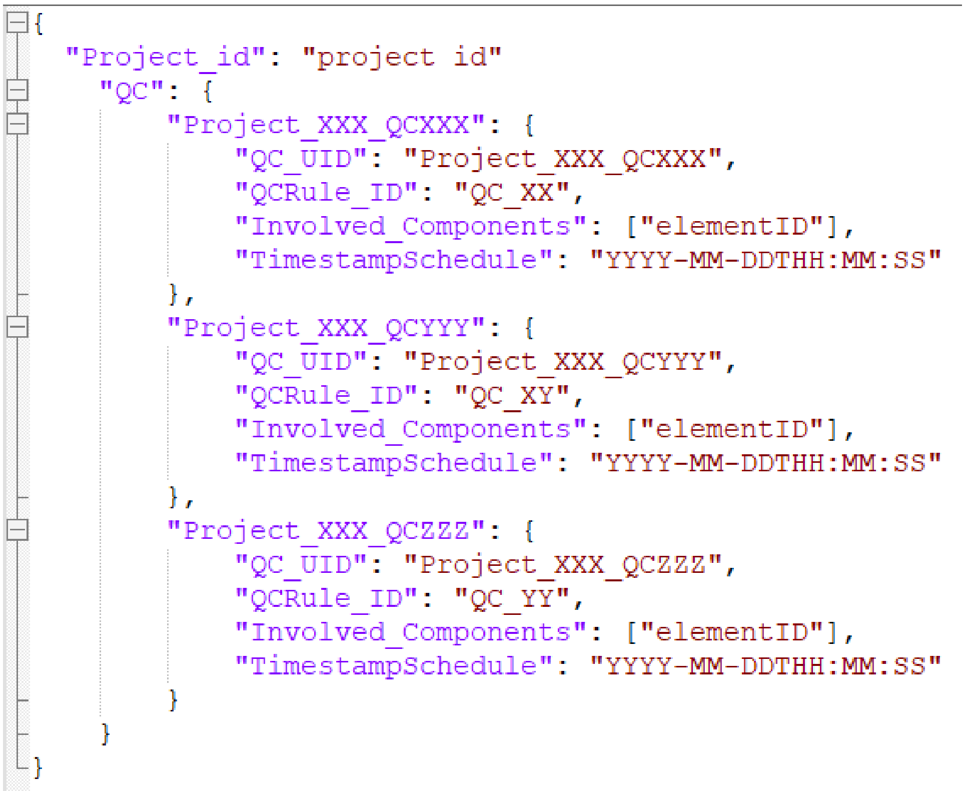
The first parameter is self-explanatory, it triggers the executable of the offline processing. The other parameters are obtained by the *GeometricQC tool communication componen*t through its communication with DTP. All the information is contained in the template in Figure 10. The schedule file is also the same Figure 10 JSON file that is provided by the DTP to the *GeometricQC tool communication sub-component*.

As for the outputs, the *GeometricQC tool offline processing* component generates 2 different JSON files, the first one containing the list of QC dictionary rules, and the second one the list of QC instances generated for the given project. Figure 11 and Figure 12 show the templates used as an output for the QC rules and the QC instances respectively.



```
{
    "Project_id": "project id",
    "QualityControlRules": {
        "QC_X": {
                "QCRule_ID": "QC_X",
                "OriginDocument": "Original document [string]",
                "Label": "QC label [string]",
                "Description": "QC description [string]",
                "ElementTypeKeywords": ["string"],
                "MaterialTypeKeywords": ["string"],
                "RelationshipTypeKeywords": ["string"]
        },
        "QC_Y": {
                "QCRule_ID": "QC_Y",
                "OriginDocument": "Original document [string]",
                "Label": "QC label [string]",
                "Description": "QC description [string]",
                "ElementTypeKeywords": ["string"],
                "MaterialTypeKeywords": ["string"],
                "RelationshipTypeKeywords": ["string"]
        },
        "QC_Z": {
                "QCRule_ID": "QC_Z",
                "OriginDocument": "Original document [string]",
                "Label": "QC label [string]",
                "Description": "QC description [string]",
                "ElementTypeKeywords": ["string"],
                "MaterialTypeKeywords": ["string"],
                "RelationshipTypeKeywords": ["string"]
        },
        "QC_XX": {
                "QCRule_ID": "QC_XX",
                "OriginDocument": "Original document [string]",
                "Label": "QC label [string]",
                "Description": "QC description [string]",
                "ElementTypeKeywords": ["string"],
                "MaterialTypeKeywords": ["string"],
                "RelationshipTypeKeywords": ["string"]
        },
        "QC_YY": {
                "QCRule_ID": "QC_YY",
                "OriginDocument": "Original document [string]",
                "Label": "QC label [string]",
                "Description": "QC description [string]",
                "ElementTypeKeywords": ["string"],
                "MaterialTypeKeywords": ["string"],
                "RelationshipTypeKeywords": ["string"]
        }
    }
}
```

Figure 11: QC rules output template



```
{
    "Project_id": "project id"
      "QC": {
          "Project_XXX_QCXXX": {
              "QC_UID": "Project_XXX_QCXXX",
              "QCRule_ID": "QC_XX",
              "Involved_Components": ["elementID"],
              "TimestampSchedule": "YYYY-MM-DDTHH:MM:SS"
          },
          "Project_XXX_QCYYY": {
              "QC_UID": "Project_XXX_QCYYY",
              "QCRule_ID": "QC_XY",
              "Involved_Components": ["elementID"],
              "TimestampSchedule": "YYYY-MM-DDTHH:MM:SS"
          },
          "Project_XXX_QCZZZ": {
              "QC_UID": "Project_XXX_QCZZZ",
              "QCRule_ID": "QC_YY",
              "Involved_Components": ["elementID"],
              "TimestampSchedule": "YYYY-MM-DDTHH:MM:SS"
          }
      }
}
```

Figure 12: QC instances output template

3.4 Application Example

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 13 shows the 3D model representation of the file.

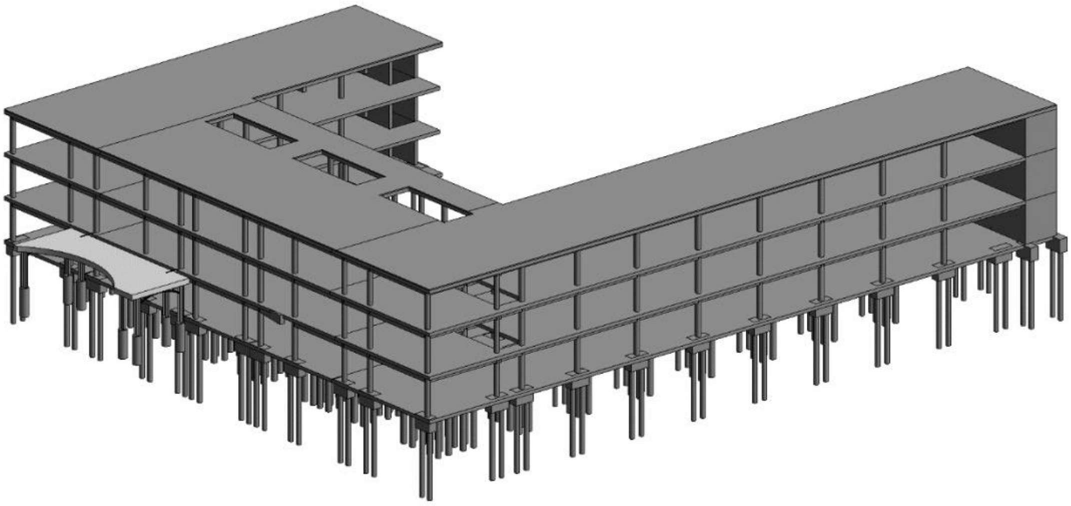


Figure 13: Revit Technical School sample model

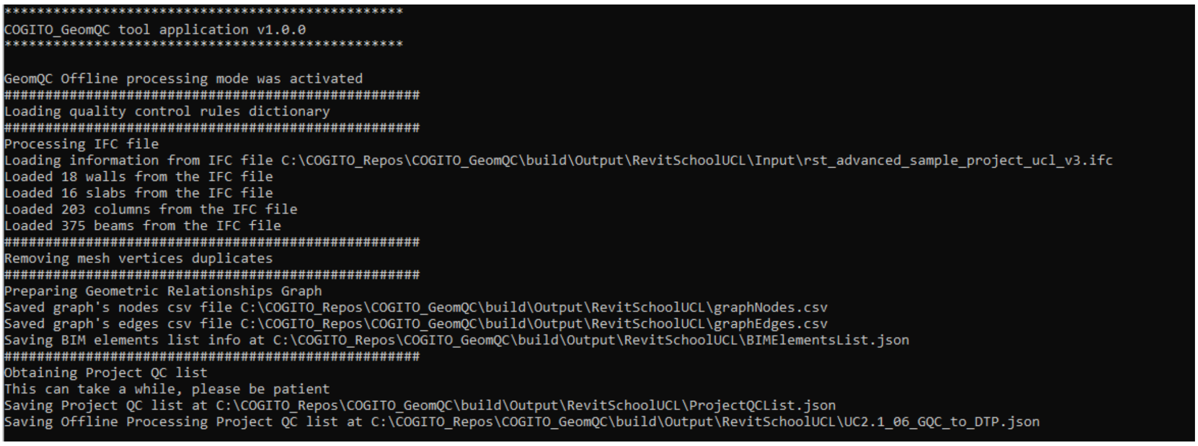
As mentioned above, the current version of the dictionary contains 30 rules, described in 3.1.2, extracted from EN 13670-2009 [2] and EN 1090-2:2018 [1]. The rules involve four types of common structural elements (columns, walls, beams, slabs) and five types of geometrical relationships detailed in 3.1.3 (adjacent storey level, same storey adjacency, above, below, and physical connection).

Once the input IFC file, the schedule 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:

1. The IFC file is processed and all the structural elements are extracted and stored in the BIM Element Manager, including the type, material and finish planned timestamp.
2. 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.
3. The *Geometric QC Dictionary* is loaded.
4. 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 14, the *GeometricQC offline processing* component extracted the 612 structural elements (between steel and concrete elements) and identified 4569 relevant geometrical relationships stored in the *Geometrical Relationships Graph*. The nodes and geometric relationships can be summarised as follows:

- Structural elements (nodes):
  - 18 walls
  - 16 slabs
  - 203 columns
  - 375 beams
- Geometric relationships (edges):
  - 126 Above
  - 126 Below
  - 1126 Same storey adjacency
  - 547 Physical connection
  - 33 Adjacent storey level



```
*************************************************
COGITO_GeomQC tool application v1.0.0
*************************************************

GeomQC Offline processing mode was activated
###################################################
Loading quality control rules dictionary
###################################################
Processing IFC file
Loading information from IFC file C:\COGITO_Repos\COGITO_GeomQC\build\Output\RevitSchoolUCL\Input\rst_advanced_sample_project_ucl_v3.ifc
Loaded 18 walls from the IFC file
Loaded 16 slabs from the IFC file
Loaded 203 columns from the IFC file
Loaded 375 beams from the IFC file
###################################################
Removing mesh vertices duplicates
###################################################
Preparing Geometric Relationships Graph
Saved graph's nodes csv file C:\COGITO_Repos\COGITO_GeomQC\build\Output\RevitSchoolUCL\graphNodes.csv
Saved graph's edges csv file C:\COGITO_Repos\COGITO_GeomQC\build\Output\RevitSchoolUCL\graphEdges.csv
Saving BIM elements list info at C:\COGITO_Repos\COGITO_GeomQC\build\Output\RevitSchoolUCL\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\RevitSchoolUCL\ProjectQCList.json
Saving Offline Processing Project QC list at C:\COGITO_Repos\COGITO_GeomQC\build\Output\RevitSchoolUCL\UC2.1_06_GQC_to_DTP.json
```

Figure 14: GeometricQC tool offline processing output example

Figure 15 illustrates the graph network in which the nodes represent structural elements and the edges represent the geometric relationships between them.

After cross-referencing all the QC rule dictionary entries and the graph, the *GeometricQC offline processing* component obtained 4,569 geometric QC instances with the following breakdown:

- **QC 1 Inclination of a column/wall** *(EN 13670-2009 10.4 Columns and Walls, No a)*: **214**
- **QC 2: Deviation between centres** *(EN 13670-2009 10.4 Columns and Walls, No b)*: **126**
- **QC 3: Curvature of a column/wall between adjacent storey levels** *(EN 13670-2009 10.4 Columns and Walls, No c)*: **214**
- **QC 4: Location of a column/wall at any storey level with respect to base level** *(EN 13670-2009 10.4 Columns and Walls, No d)*: **126**
- **QC 5: Location of a beam-to-column connection measured relative to the column** *(EN 13670-2009 10.5 Beams and Slabs, No a)*: **530**
- **QC 6: Position of bearing axis of support** *(EN 13670-2009 10.5 Beams and Slabs, No b)*: **357**
- **QC 7: Cross-sectional dimensions** *(EN 13670-2009 10.6 Sections, No a)*: **566**
- **QC 8: Lap-joints** *(EN 13670-2009 10.6 Sections, No c)*: **31**
- **QC 9: Free space between adjacent columns/walls** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No c)*: **326**
- **QC 10: Horizontal straightness of beams** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No a)*: **357**
- **QC 11: Distance between adjacent beams** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No b)*: **222**
- **QC 12: Inclination of a beam/slab** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No c)*: **370**
- **QC 13: Level of adjacent beams** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No d)*: **222**
- **QC 14: Level of adjacent floors at supports** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No e)*: **33**
- **QC 15: Orthogonality of a cross-section** *(EN 13670-2009 Annex G – G.10.6 Sections, No a)*: **584**
- **QC 16: Column position in plane** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No a)*: **196**
- **QC 17: Wall position in plane** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No b)*: **18**
- **QC 18: Column height relative to adjacent levels** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B15, No 2)*: **7**
- **QC 19: Beam slope** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B15, No 3)*: **17**
- **QC 20: Levels of adjacent beams** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B15, No 6)*: **7**
- **QC 21: Spacing between beam centrelines** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B16, No 1)*: **7**

- **QC_22: Beam location at columns** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B16, No 2)*: **17**
- **QC_23: Inclination of columns** *(EN 1090-2:2018 Annex B – Erection tolerances, table B17, No 1 and Table B18, No 2)*: **7**
- **QC_24: Column location** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B20, No 1)*: **7**
- **QC_25: Column spacing** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B20, No 3)*: **8**
- **QC_26: Location of rail in plan** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B22, No 1)*: **0**
- **QC_27: Local alignment of rail** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B22, No 2)*: **0**
- **QC_28: Level of rail** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B22, No 3)*: **0**
- **QC_29: Relative levels of rails on the two sides of a runway** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B22, No 6)*: **0**
- **QC_30: Spacing over span S between centres of crane rails** *(EN 1090-2:2018 Annex B – Erection tolerances, Table B22, No 7)*: **0**

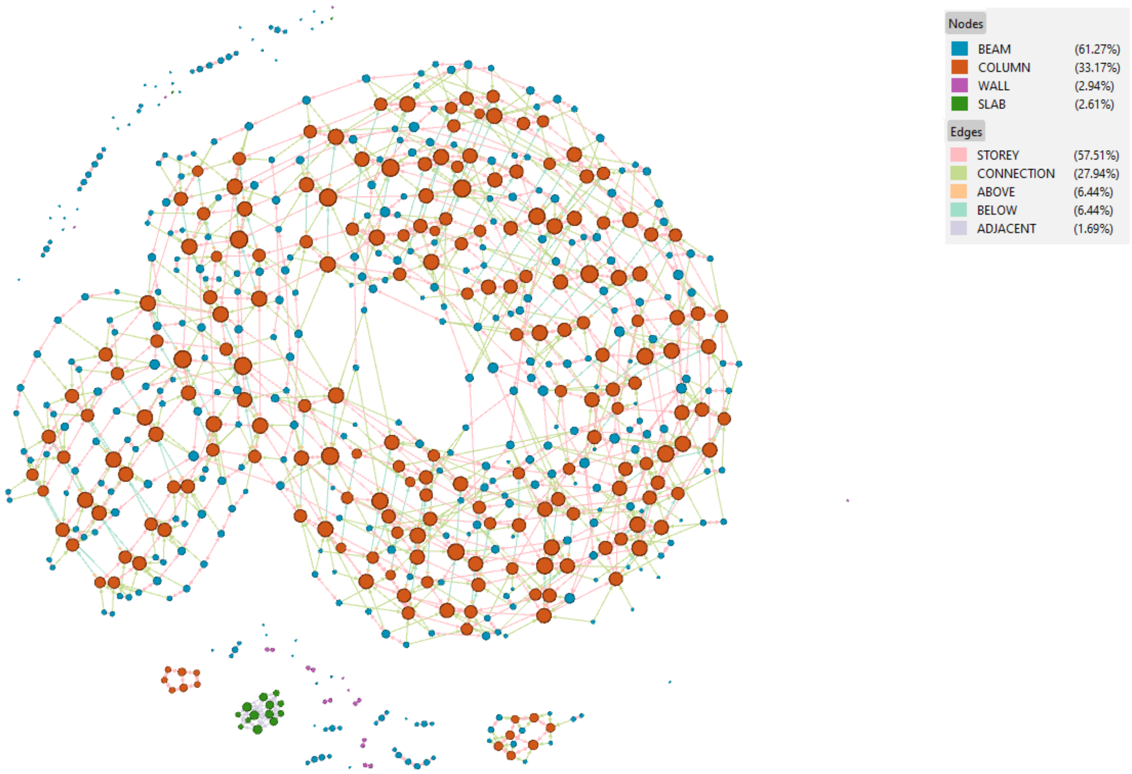
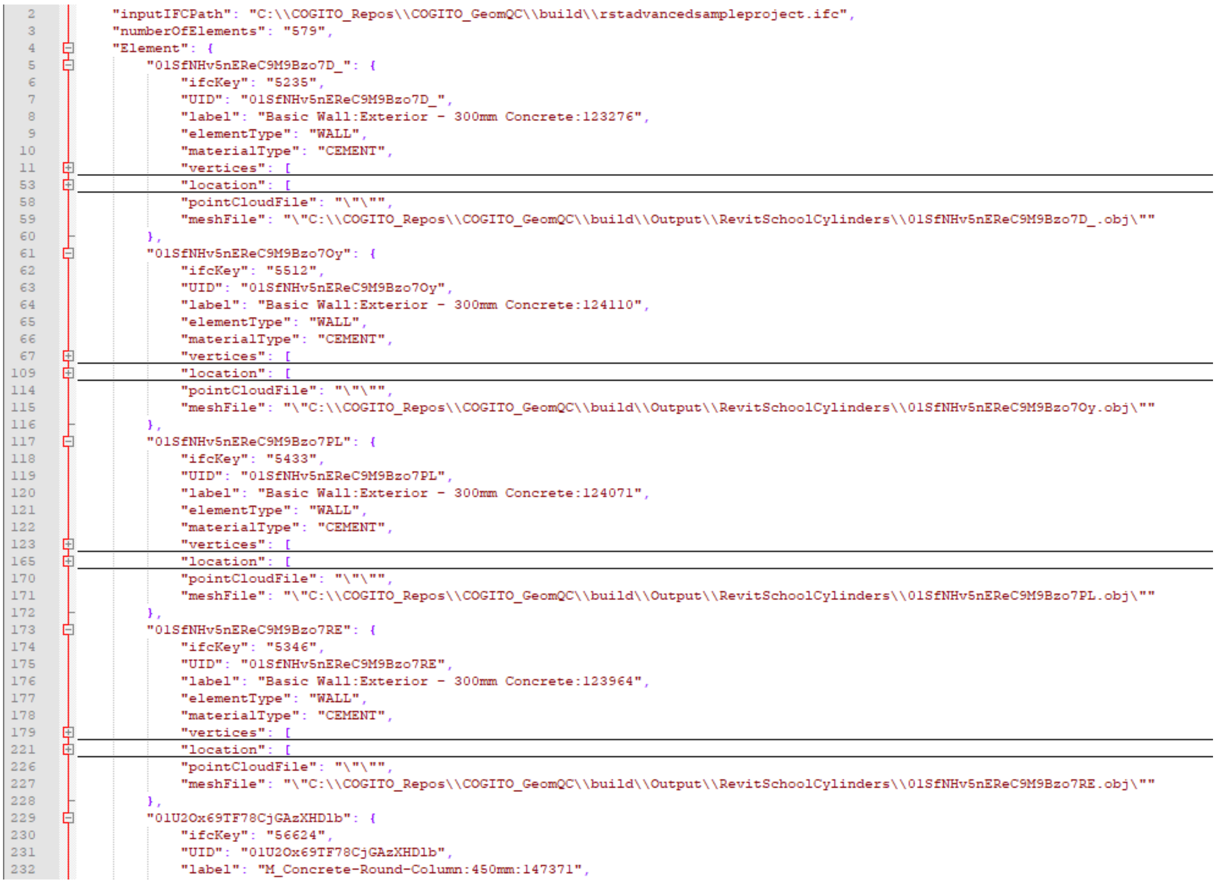


Figure 15: Graph representation of the Revit Technical School sample model after GeometricQC offline processing

Taking into account that we are using a limited QC dictionary with only 30 rules, we can see the significant amount of geometrical tolerance verification that needs to happen during the construction phase. The GeometricQC tool aims 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.

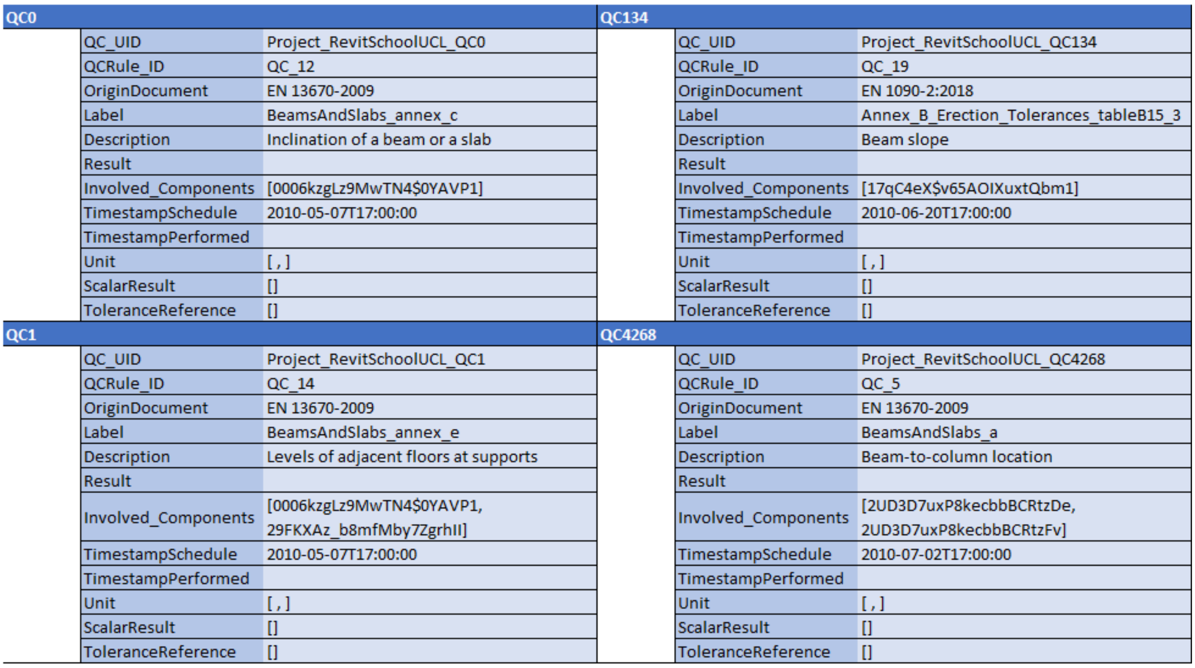
Finally, after all the offline processing is done, the offline processing component generates four (4) different internal files which are then internally available for the rest of the workflow and subsequent execution instances of the tool:

- **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 17).
- **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 16 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 16: BIM Element Manager JSON file example obtained from the Revit Technical School model using the GeometricQC tool

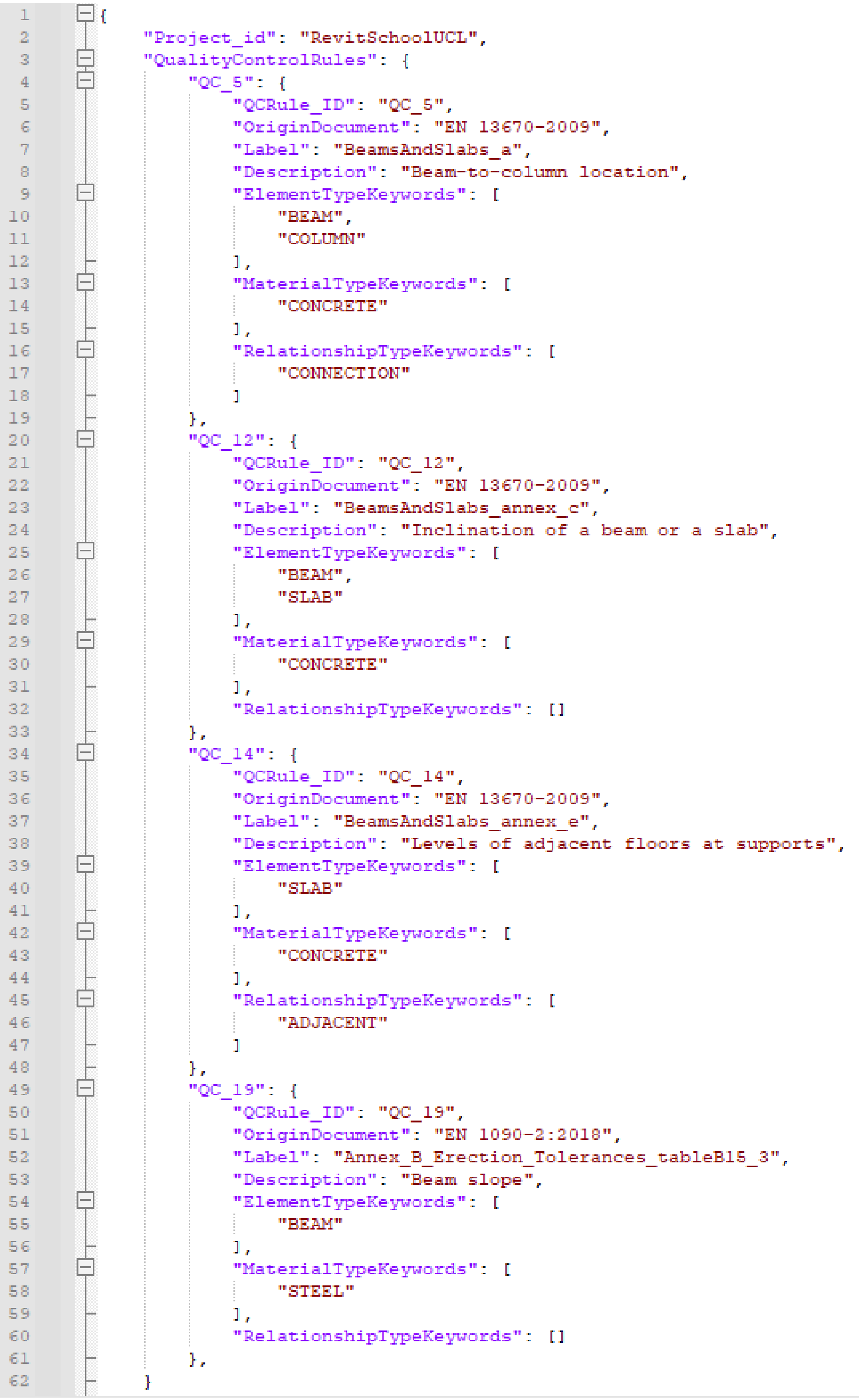


QC0			QC134		
	QC_UID	Project_RevitSchoolUCL_QC0		QC_UID	Project_RevitSchoolUCL_QC134
	QCRule_ID	QC_12		QCRule_ID	QC_19
	OriginDocument	EN 13670-2009		OriginDocument	EN 1090-2:2018
	Label	BeamsAndSlabs_annex_c		Label	Annex_B_Erection_Tolerances_tableB15_3
	Description	Inclination of a beam or a slab		Description	Beam slope
	Result			Result	
	Involved_Components	[0006kzgLz9MwTN4$0YAVP1]		Involved_Components	[17qC4eX$v65AOIXuxtQbm1]
	TimestampSchedule	2010-05-07T17:00:00		TimestampSchedule	2010-06-20T17:00:00
	TimestampPerformed			TimestampPerformed	
	Unit	[,]		Unit	[,]
	ScalarResult	[]		ScalarResult	[]
	ToleranceReference	[]		ToleranceReference	[]
QC1			QC4268		
	QC_UID	Project_RevitSchoolUCL_QC1		QC_UID	Project_RevitSchoolUCL_QC4268
	QCRule_ID	QC_14		QCRule_ID	QC_5
	OriginDocument	EN 13670-2009		OriginDocument	EN 13670-2009
	Label	BeamsAndSlabs_annex_e		Label	BeamsAndSlabs_a
	Description	Levels of adjacent floors at supports		Description	Beam-to-column location
	Result			Result	
	Involved_Components	[0006kzgLz9MwTN4$0YAVP1, 29FKXAz_b8mfMby7ZgrhII]		Involved_Components	[2UD3D7uxP8kecbbBCRtzDe, 2UD3D7uxP8kecbbBCRtzFv]
	TimestampSchedule	2010-05-07T17:00:00		TimestampSchedule	2010-07-02T17:00:00
	TimestampPerformed			TimestampPerformed	
	Unit	[,]		Unit	[,]
	ScalarResult	[]		ScalarResult	[]
	ToleranceReference	[]		ToleranceReference	[]

Figure 17: Revit Technical School geometrical tolerances list example

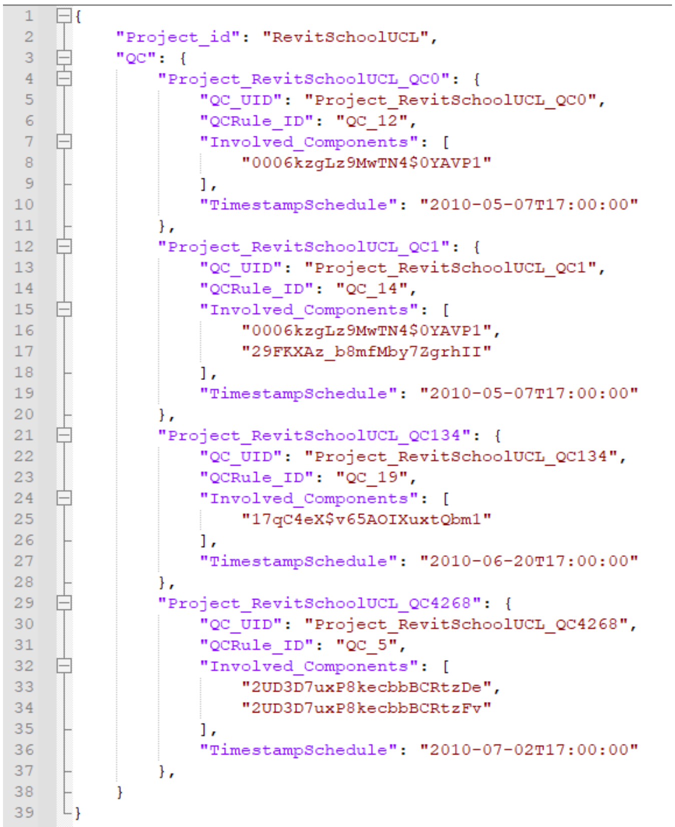
As mentioned in 3.3.2, the offline processing component also generates two (2) output files to be submitted to the DTP following the output template and API:

- **Project QC dictionary rules: (***UC2.1_05_GQC_to_DTP.json***):** output file containing all the QC dictionary rules with their keywords and information. Figure 18 shows an example of some of the rules used in the Revit Technical School model.
- **Project QC instances list (***UC2.1_06_GQC_to_DTP.json***):** this file contains the initial list of all the QC instances that belong to the project. Figure 19 shows an example of the same four QC instances that were obtained from the Revit Technical School model.



```
{
    "Project_id": "RevitSchoolUCL",
    "QualityControlRules": {
        "QC_5": {
            "QCRule_ID": "QC_5",
            "OriginDocument": "EN 13670-2009",
            "Label": "BeamsAndSlabs_a",
            "Description": "Beam-to-column location",
            "ElementTypeKeywords": [
                "BEAM",
                "COLUMN"
            ],
            "MaterialTypeKeywords": [
                "CONCRETE"
            ],
            "RelationshipTypeKeywords": [
                "CONNECTION"
            ]
        },
        "QC_12": {
            "QCRule_ID": "QC_12",
            "OriginDocument": "EN 13670-2009",
            "Label": "BeamsAndSlabs_annex_c",
            "Description": "Inclination of a beam or a slab",
            "ElementTypeKeywords": [
                "BEAM",
                "SLAB"
            ],
            "MaterialTypeKeywords": [
                "CONCRETE"
            ],
            "RelationshipTypeKeywords": []
        },
        "QC_14": {
            "QCRule_ID": "QC_14",
            "OriginDocument": "EN 13670-2009",
            "Label": "BeamsAndSlabs_annex_e",
            "Description": "Levels of adjacent floors at supports",
            "ElementTypeKeywords": [
                "SLAB"
            ],
            "MaterialTypeKeywords": [
                "CONCRETE"
            ],
            "RelationshipTypeKeywords": [
                "ADJACENT"
            ]
        },
        "QC_19": {
            "QCRule_ID": "QC_19",
            "OriginDocument": "EN 1090-2:2018",
            "Label": "Annex_B_Erection_Tolerances_tableB15_3",
            "Description": "Beam slope",
            "ElementTypeKeywords": [
                "BEAM"
            ],
            "MaterialTypeKeywords": [
                "STEEL"
            ],
            "RelationshipTypeKeywords": []
        },
    }
```

Figure 18: Revit Technical School QC rules example output



```
{
    "Project_id": "RevitSchoolUCL",
    "QC": {
        "Project_RevitSchoolUCL_QC0": {
            "QC_UID": "Project_RevitSchoolUCL_QC0",
            "QCRule_ID": "QC_12",
            "Involved_Components": [
                "0006kzgLz9MwTN4$0YAVP1"
            ],
            "TimestampSchedule": "2010-05-07T17:00:00"
        },
        "Project_RevitSchoolUCL_QC1": {
            "QC_UID": "Project_RevitSchoolUCL_QC1",
            "QCRule_ID": "QC_14",
            "Involved_Components": [
                "0006kzgLz9MwTN4$0YAVP1",
                "29FKXAz_b8mfMby7ZgrhII"
            ],
            "TimestampSchedule": "2010-05-07T17:00:00"
        },
        "Project_RevitSchoolUCL_QC134": {
            "QC_UID": "Project_RevitSchoolUCL_QC134",
            "QCRule_ID": "QC_19",
            "Involved_Components": [
                "17qC4eX$v65AOIXuxtQbm1"
            ],
            "TimestampSchedule": "2010-06-20T17:00:00"
        },
        "Project_RevitSchoolUCL_QC4268": {
            "QC_UID": "Project_RevitSchoolUCL_QC4268",
            "QCRule_ID": "QC_5",
            "Involved_Components": [
                "2UD3D7uxP8kecbbBCRtzDe",
                "2UD3D7uxP8kecbbBCRtzFv"
            ],
            "TimestampSchedule": "2010-07-02T17:00:00"
        },
    }
}
```

Figure 19: Revit Technical School QC instances offline processing example output

3.5 Licensing

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

3.6 Installation Instructions

No complex installation is required. The code will be provided in the Cyberbuild Datashare collections and Github. 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 tool offline processing* sub-component is considered to be finished. Extra functionalities may nonetheless be found to be missing during the pre-validation and validation phase, at which point they will be added. Regarding the integration status, the *GeometricQC tool offline processing* component only interacts directly with the GeometricQC tool communication component. However, as described in section 3.3, it has indirect interaction with the DTP. As such, it follows and uses the established templates described in sections 3.3.1 and 3.3.2 as input and output integration, and follows the API described in 3.3.3.

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 D2.5 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 3. Functional requirement Req-1.1 is covered using this component for the 4D BIM data. Another part of the GeometricQC tool has the charge of loading the point cloud data. 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 3: *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	Achieved
Req-2.1	Scalability	Non-Functional	Achieved
Req-2.2	Reusability	Non-Functional	Achieved
Req-2.3	Interoperability	Non-Functional	Achieved

Table 4 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 and steel structures. Regarding COGI-QC-11, the *GeometricQC offline processing* component allows to load and interpret the as-planned 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 4: *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 second and final 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. Support for IFC version 4.3 remains under development driven by the emerging needs of the pre-validation and validation activities and the capability of the Digital Twin Platform.

- It requires the input IFC file to be syntactically correct (i.e.consistent with the IFC 4.0 schema) and free of geometric errors.
- 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 project 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 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 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 QC results, and the relevant complementary files (such as 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 in terms of its functional sub-components and its implementation.

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 elements that have been completed and the as-built point clouds acquired of those elements (all defined in the QC work order received by the Communication component). Hereafter we refer to these elements that have just been built, scanned as the *active* structural elements. Figure 20 shows a detailed workflow for the component.

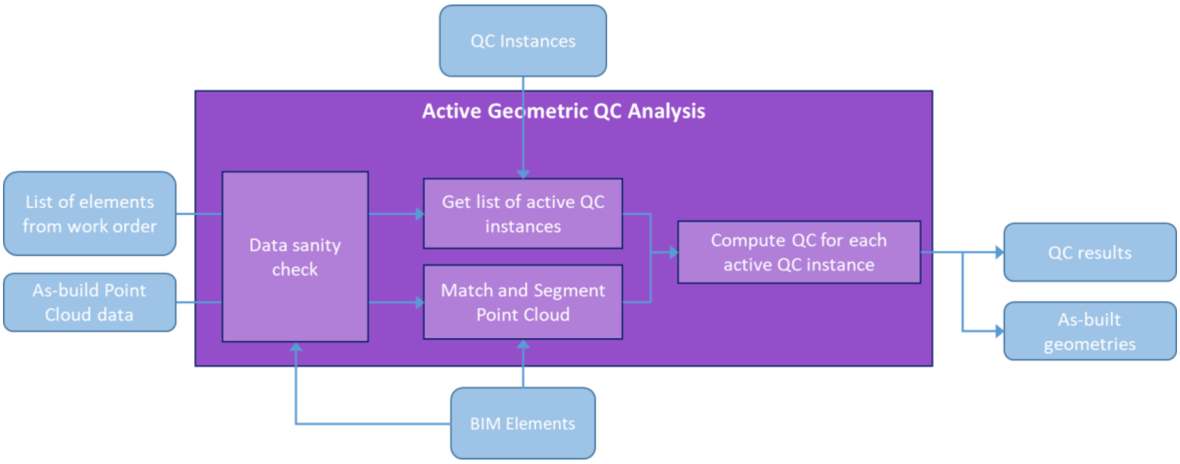

QC Instances
Active Geometric QC Analysis
List of elements from work order
As-build Point Cloud data
Data sanity check
Get list of active QC instances
Match and Segment Point Cloud
Compute QC for each active QC instance
QC results
As-built geometries
BIM Elements

Figure 20: 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 is at the interface between the DTP and the rest of the GeometricQC tool, and is detailed in Section 5. In that interface, the relevant input parameters are prepared and the relevant service call is made. The current version of the component requires:

- The project ID.
- The QC work order ID.
- The list of active elements of interest extracted from the QC work order. This is simply the list of the unique IDs (UIDs) of those elements, and thus can be passed as a text file.
- The (project-)referenced as-built point cloud data associated to the same QC work order with the data acquisition timestamp. The as-built point cloud data is required to be in PLY and/or E57 format.

The GeometricQC tool follows the following steps:

- Data sanity check: the first step of the *GeometricQC tool active geometric quality control analysis* component is to obtain the list of 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 component returns a result with a failure message accompanied with an explanation.
- Get list of active QC instances: if the previous step is satisfactory, the component obtains the QC instances whose contexts refer to the active elements. If a QC instance contains any other element, the component also checks that that element has also been constructed. If any element related to a QC instance is not constructed or the as-built data cannot be segmented with enough geometric information to obtain the as-built pose or perform geometric QC (including the current active elements), then the QC instance is discarded because it cannot be checked. The result is a list of active QC instances.
- Match and Segment Point Cloud: the component performs the matching and segmentation for each of the active elements and their active related elements over the input point clouds to get the point cloud subset corresponding to each active element (using the *Point Cloud Matching and Segmentation* sub-component described in D5.2).

Compute QC for each active QC instance: 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 that can calculate the as-built quantity that needs to be measured and compared against its as-planned value and the tolerance.

As explained in Section 3.1.2 and Section 3.1.4, all QC rules have the same output, making it easily interoperable. Once a QC instance is verified, it is updated with the result, all QC instance results are exported into a structured output file (JSON) and auxiliary files are generated. These files include the segmented point clouds as well as as-built meshes coloured according to the QC instance results, allowing to visually inspect the results. All the relevant output result information generated during the active QC analysis is also stored and sent to the DTP for project usage.

The current implementation of the *GeometricQC tool active geometric quality control analysis* component includes the QC algorithms for 18 entries in the Geometric QC Dictionary:

- **QC 1: Inclination of a column/wall** *(EN 13670-2009 10.4 Columns and Walls, No a)*
- **QC 2: Deviation between centres** *(EN 13670-2009 10.4 Columns and Walls, No b)*
- **QC 5: Location of a beam-to-column connection measured relative to the column** *(EN 13670-2009 10.5 Beams and Slabs, No a)*
- **QC 9: Free space between adjacent columns/walls** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No c)*
- **QC 11: Distance between adjacent beams** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No b)*
- **QC 12: Inclination of a beam/slab** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No c)*
- **QC 13: Level of adjacent beams** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No d)*
- **QC 14: Level of adjacent floors at supports** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No e)*
- **QC 16: Column position in plane** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No a)*
- **QC 17: Wall position in plane** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No b)*
- **QC 18: Column height relative to adjacent levels** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B15 – 2)*
- **QC 19: Beam slope** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B15 – 3)*
- **QC 20: Levels of adjacent beams** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B15 – 6)*
- **QC 21: Spacing between beam centrelines** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B16 – 1)*
- **QC 22: Location at columns** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B16 – 2)*
- **QC 23: Inclination of columns** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B17 – 1 and Table B17-2)*

- <u>**QC 24**</u>**: Column location** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B20 – 1)*
- <u>**QC 25**</u>**: Column spacing** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B20 – 3)*

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 5: 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
LibE57	1.1.312	MIT License

4.3 Input, Output and API Documentation

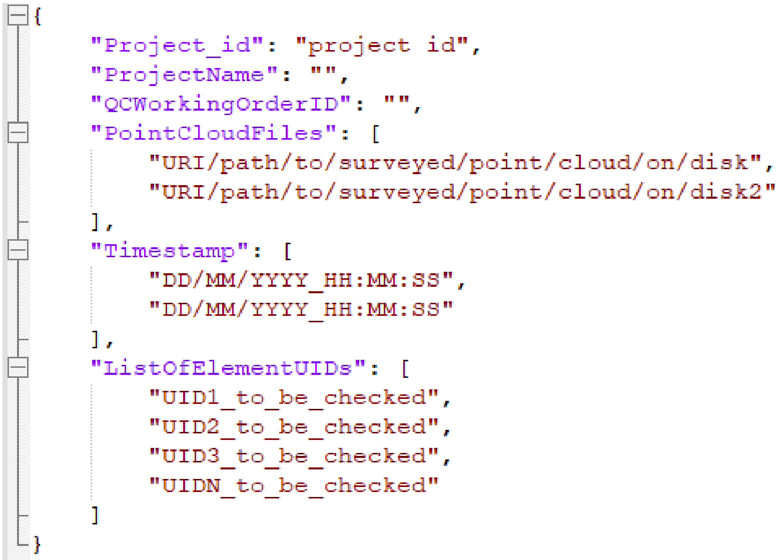
4.3.1 Inputs

The *GeometricQC tool active geometric quality control analysis* 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:

- **Project's Name/ID:** the construction project name or ID given by the user or the DTP. This project identification will be used to retrieve the internal output folder where to collect all data created during the offline pre-processing phase and store all the generated data by the GeometricQC tool. In the case of the *GeometricQC tool active geometric quality control analysis* component.
- **Geometric QC work order ID:** the work order ID that links the active structural elements of interest and the QC analysis.
- **As-built point clouds list file path:** file path of the text file with the list of file paths of the project-referenced as-built point clouds that will be used by the *Point Cloud Matching and Segmentation* component. The point clouds must be in E57 or PLY file format.
- **As-built point clouds timestamps file path:** file path of the text file containing the timestamps of the as-built point clouds provided in the previous input. These timestamps should correspond on the date where the point clouds were acquired, reflecting the as-built snapshot of the construction project, and making certain the GeometricQC tool has the newest version of the as-built elements for performing the QC analysis.

- **List of active structural elements UIDs file path:** file path of the text file containing the list of active structural elements UIDs that appear in the same QC work order listed before.

When the *GeometricQC tool active geometric quality control analysis* component is executed by the *GeometricQC communication component*, the inputs are encoded in a JSON file as can be seen in Figure 24. In all cases the input information must follow the API described in 4.3.3.



```
{
    "Project_id": "project id",
    "ProjectName": "",
    "QCWorkingOrderID": "",
    "PointCloudFiles": [
        "URI/path/to/surveyed/point/cloud/on/disk",
        "URI/path/to/surveyed/point/cloud/on/disk2"
    ],
    "Timestamp": [
        "DD/MM/YYYY_HH:MM:SS",
        "DD/MM/YYYY_HH:MM:SS"
    ],
    "ListOfElementUIDs": [
        "UID1_to_be_checked",
        "UID2_to_be_checked",
        "UID3_to_be_checked",
        "UIDN_to_be_checked"
    ]
}
```

Figure 21: GeometricQC tool active geometric QC analysis DTP input template

4.3.2 Outputs

The *GeometricQC tool active geometric quality control analysis* component produces a couple of outputs, both for internal use and to be sent to the DTP:

- **As-built geometries:** the list of as-built segmented point clouds and as-built meshes, stored in a JSON file to be sent to the DTP. These as-built geometries can be used by the DCC and DigitAR to visualise the different QC results, applying a colour code, and see in detail the reason why the geometric tolerances were satisfied or not.
- **Geometric QC results:** the results of each of the executed QC instances obtained during the active geometric QC analysis. Since the DTP already contains the list of QC instances from the pre-processing phase, the generated output only contains the data pertaining to QC results fields.

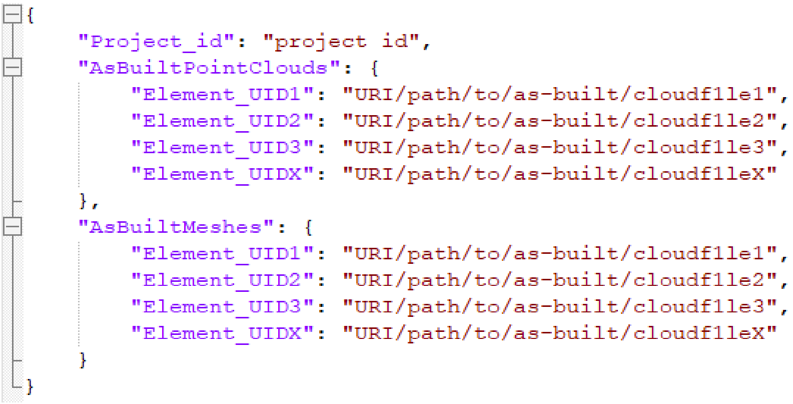
The next sub-section shows the different templates that are used and expected for the correct execution of the *GeometricQC tool active geometric quality control analysis* component.

4.3.3 API Documentation

The *GeometricQC tool active geometric quality control analysis* component is executed via the GeometricQC tool communication component using basic command lines executions, with the following format:

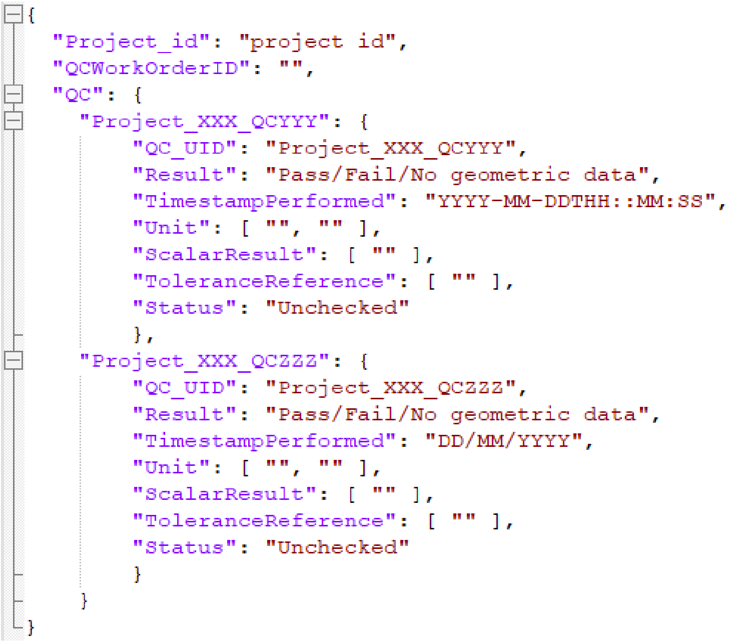
```
“COGITO_GeomQC.exe -ToolMode active -projectID ProjectID –workOrder WOID –pointCloudList pointCloudList.txt -pointCloudTimestamps pointCloudTimestamps.txt –UIDs elementsList.txt”
```

The first parameter is self-explanatory, it triggers the executable of the active geometric quality control analysis. The other parameters are obtained by the *GeometricQC tool communication component* through its communication with DTP. All the information is contained in the template in Figure 21.As for the outputs, the *GeometricQC tool active geometric quality control analysis* component generates two JSON files, the first one containing the list of as-built geometries, both point clouds and meshes, and the second one the results of each of the QC instances verified during that execution. Figure 22 and Figure 23 show templates for those outputs.



```
{
    "Project_id": "project id",
    "AsBuiltPointClouds": {
        "Element_UID1": "URI/path/to/as-built/cloudf1le1",
        "Element_UID2": "URI/path/to/as-built/cloudf1le2",
        "Element_UID3": "URI/path/to/as-built/cloudf1le3",
        "Element_UIDX": "URI/path/to/as-built/cloudf1leX"
    },
    "AsBuiltMeshes": {
        "Element_UID1": "URI/path/to/as-built/cloudf1le1",
        "Element_UID2": "URI/path/to/as-built/cloudf1le2",
        "Element_UID3": "URI/path/to/as-built/cloudf1le3",
        "Element_UIDX": "URI/path/to/as-built/cloudf1leX"
    }
}
```

Figure 22: As-built geometries output template



```
{
    "Project_id": "project id",
    "QCWorkOrderID": "",
    "QC": {
      "Project_XXX_QCYYY": {
          "QC_UID": "Project_XXX_QCYYY",
          "Result": "Pass/Fail/No geometric data",
          "TimestampPerformed": "YYYY-MM-DDTHH::MM:SS",
          "Unit": [ "", "" ],
          "ScalarResult": [ "" ],
          "ToleranceReference": [ "" ],
          "Status": "Unchecked"
          },
      "Project_XXX_QCZZZ": {
          "QC_UID": "Project_XXX_QCZZZ",
          "Result": "Pass/Fail/No geometric data",
          "TimestampPerformed": "DD/MM/YYYY",
          "Unit": [ "", "" ],
          "ScalarResult": [ "" ],
          "ToleranceReference": [ "" ],
          "Status": "Unchecked"
          }
      }
}
```

Figure 23: QC results output template

4.4 Application Example

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 (2.5 mm standard deviation) to mimic potential construction errors and a realistic laser scanning result. We can see the generated point cloud next to the BIM model in Figure 24.

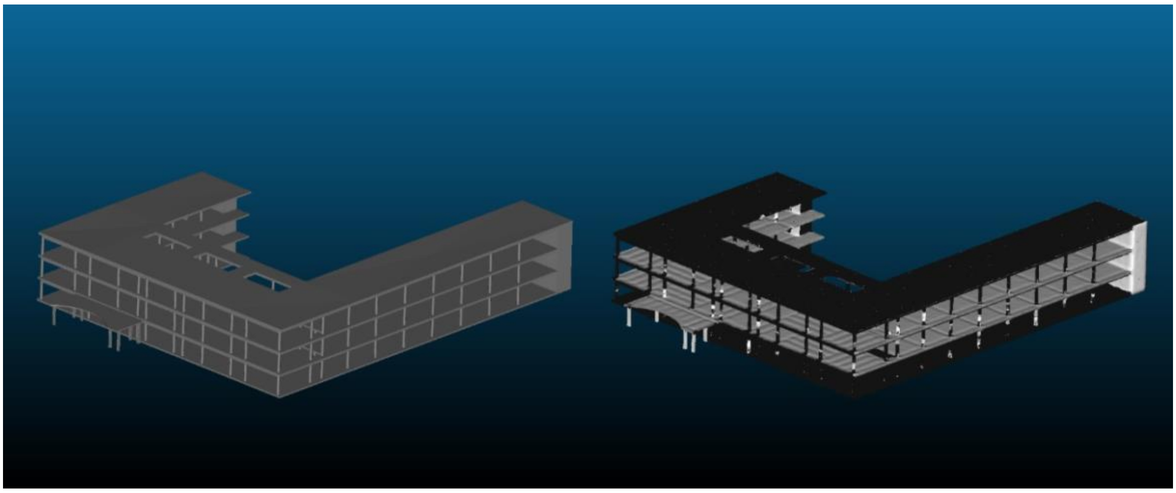


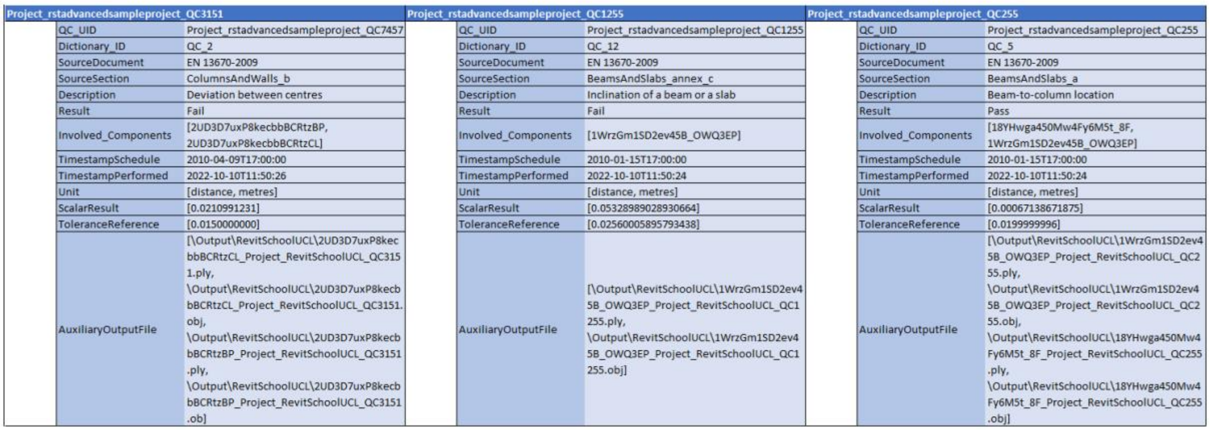
Figure 24: Revit Sample Project Technical School BIM model (left) and synthetic point cloud (right)

From the *GeometricQC offline processing* component we obtained the list of QC instances (3.4.1), so now we start from a list of active structural elements, including at least one of each of the 4 types, and the synthetic point cloud. The list of active structural elements includes 5 different UIDs. The *Active Geometric Control Analysis* component then checks, matches, and segments the as-built point cloud, and then computes the as-built pose of the active elements. In this example, the component successfully achieves this for all 5 elements. The component then finds all QC instances that relate to these active structural elements. If one of those QC instances relates to more than one object, the component discards those who other elements have not been constructed yet. The result is a reduced list of QC instances that relate only to elements that have constructed, including the current active ones. In this example, this results in final list of 36 QC instances broken down as follows:

- **QC 1: Inclination of a column/wall** *(EN 13670-2009 10.4 Columns and Walls, No a)*: **2**
- **QC 2: Deviation between centres** *(EN 13670-2009 10.4 Columns and Walls, No b)*: **4**
- **QC 5: Location of a beam-to-column connection measured relative to the column** *(EN 13670-2009 10.5 Beams and Slabs, No a)*: **7**
- **QC 9: Free space between adjacent columns/walls** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No c)*: **7**
- **QC 11: Distance between adjacent beams** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No b)*: **1**
- **QC 12: Inclination of a beam/slab** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No c)*: **1**
- **QC 13: Level of adjacent beams** *(EN 13670-2009 Annex G – G.10.5 Beams and Slabs, No d)*: **1**
- **QC 16: Column position in plane** *(EN 13670-2009 Annex G – G.10.4 Columns and Walls, No a)*: **2**
- **QC 18: Column height relative to adjacent levels** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B15 – 2)*: **1**
- **QC 19: Beam slope** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B15 – 3)*: **1**
- **QC 20: Levels of adjacent beams** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B15 -6)*: **1**
- **QC 21: Spacing between beam centrelines** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B16 – 1)*: **1**
- **QC 22: Location at columns** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B16 - 2)*: **3**
- **QC 23: Inclination of columns** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B17 – 1 and Table B17-2)*: **1**
- **QC 24: Column location** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B20 – 1)*: **1**
- **QC 25: Column spacing** *(EN 1090-2:2018 Annex B – Erection Tolerances, Table B20 – 3)*: **2**

From the execution of the QC algorithm corresponding to each rule, the 36 QC instances from the project's list were updated with the QC results. A new output file is generated containing the results of those 36 QC instances. Figure 25 shows a couple of examples in that output file. We can see that in the first case, the QC instance refers to the dictionary entry **"QC_2: Deviation between centres"**, that the result is a ***"Fail",*** which means the geometric tolerance is not satisfied. We can also analyse the auxiliary files that were generated, corresponding to the meshes and segmented point clouds for the two columns (Figure 26). In this case, since the QC result is not satisfactory, the objects are coloured red and we can see from the detailed views that there is a significant deviation in the position of the columns.

The other results of Figure 25 show the outcome of performing two QC instances referring to a beam that has incorrect inclination but correct positioning with the column it is connected to. 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 there is no vertical deviation. Figure 27 and Figure 28 show the as-built point clouds alongside the as-planned meshes, where we can see that both meshes and point clouds are coloured red in the case when the tolerance is not satisfied, and coloured green when it is satisfied, allowing the user to visualise the problems. 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 one of the QCs to fail. In Figure 29 and Figure 30 we can see both output JSON files (following the templates described in Figure 22 and Figure 23) corresponding to the as-built geometries and the QC results.



Project_rstadvancedsampleproject_QC3151			Project_rstadvancedsampleproject_QC1255			Project_rstadvancedsampleproject_QC255		
	QC_UID	Project_rstadvancedsampleproject_QC7457		QC_UID	Project_rstadvancedsampleproject_QC1255		QC_UID	Project_rstadvancedsampleproject_QC255
	Dictionary_ID	QC_2		Dictionary_ID	QC_12		Dictionary_ID	QC_5
	SourceDocument	EN 13670-2009		SourceDocument	EN 13670-2009		SourceDocument	EN 13670-2009
	SourceSection	ColumnsAndWalls_b		SourceSection	BeamsAndSlabs_annex_c		SourceSection	BeamsAndSlabs_a
	Description	Deviation between centres		Description	Inclination of a beam or a slab		Description	Beam-to-column location
	Result	Fail		Result	Fail		Result	Pass
	Involved_Components	[2UD3D7uxP8kecbbBCRtzBP, 2UD3D7uxP8kecbbBCRtzCL]		Involved_Components	[1WrzGm1SD2ev45B_OWQ3EP]		Involved_Components	[18YHwga450Mw4Fy6M5t_8F, 1WrzGm1SD2ev45B_OWQ3EP]
	TimestampSchedule	2010-04-09T17:00:00		TimestampSchedule	2010-01-15T17:00:00		TimestampSchedule	2010-01-15T17:00:00
	TimestampPerformed	2022-10-10T11:50:26		TimestampPerformed	2022-10-10T11:50:24		TimestampPerformed	2022-10-10T11:50:24
	Unit	[distance, metres]		Unit	[distance, metres]		Unit	[distance, metres]
	ScalarResult	[0.0210991231]		ScalarResult	[0.05328989028930664]		ScalarResult	[0.00067138671875]
	ToleranceReference	[0.0150000000]		ToleranceReference	[0.02560005895793438]		ToleranceReference	[0.0199999996]
	AuxiliaryOutputFile	[\Output\RevitSchoolUCL\2UD3D7uxP8kecbbBCRtzCL_Project_RevitSchoolUCL_QC3151.ply, \Output\RevitSchoolUCL\2UD3D7uxP8kecbbBCRtzCL_Project_RevitSchoolUCL_QC3151.obj, \Output\RevitSchoolUCL\2UD3D7uxP8kecbbBCRtzBP_Project_RevitSchoolUCL_QC3151.ply, \Output\RevitSchoolUCL\2UD3D7uxP8kecbbBCRtzBP_Project_RevitSchoolUCL_QC3151.ob]		AuxiliaryOutputFile	[\Output\RevitSchoolUCL\1WrzGm1SD2ev45B_OWQ3EP_Project_RevitSchoolUCL_QC1255.ply, \Output\RevitSchoolUCL\1WrzGm1SD2ev45B_OWQ3EP_Project_RevitSchoolUCL_QC1255.obj]		AuxiliaryOutputFile	[\Output\RevitSchoolUCL\1WrzGm1SD2ev45B_OWQ3EP_Project_RevitSchoolUCL_QC255.ply, \Output\RevitSchoolUCL\1WrzGm1SD2ev45B_OWQ3EP_Project_RevitSchoolUCL_QC255.obj, \Output\RevitSchoolUCL\18YHwga450Mw4Fy6M5t_8F_Project_RevitSchoolUCL_QC255.ply, \Output\RevitSchoolUCL\18YHwga450Mw4Fy6M5t_8F_Project_RevitSchoolUCL_QC255.obj]

Figure 25: Example results obtained using the GeometricQC tool for Revit Sample Project Technical School model

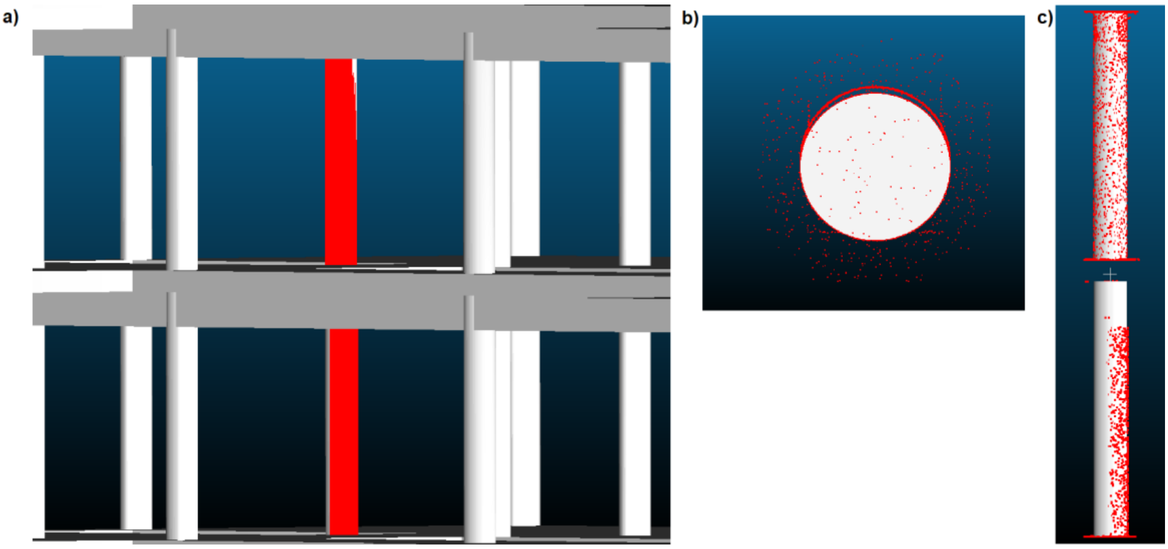


Figure 26: Example of a not satisfactory geometric QC result

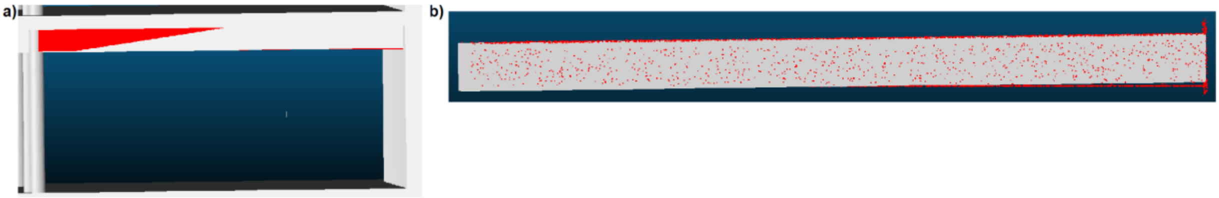


Figure 27: Example of a combined not satisfactory geometric QC result

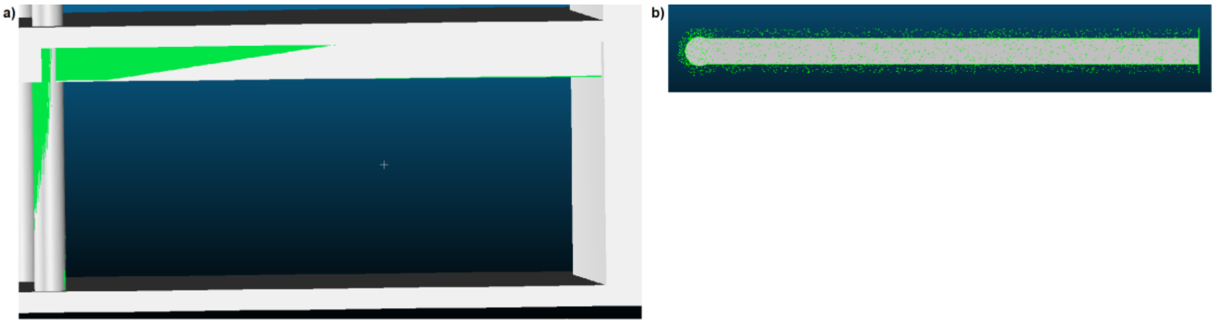
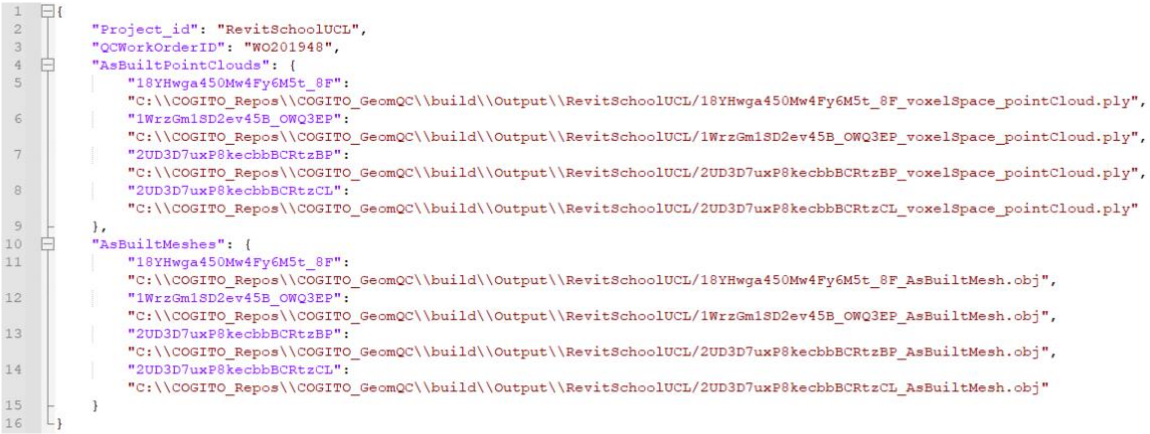
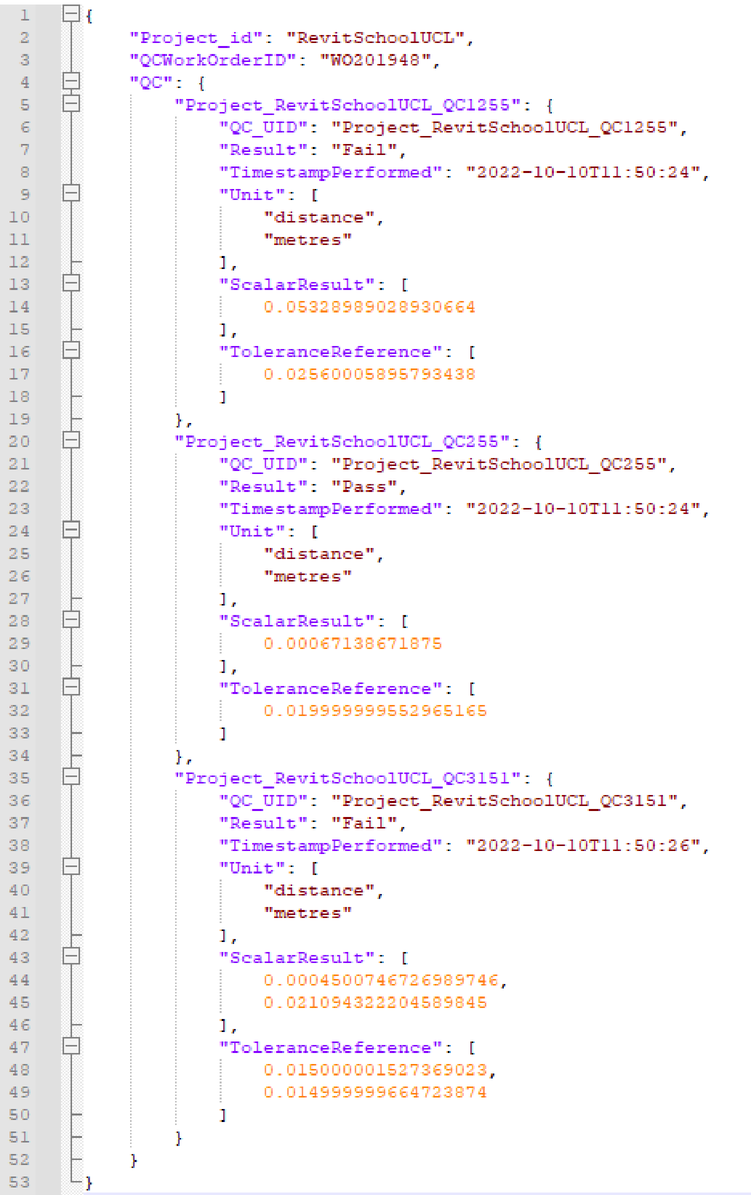


Figure 28: Example of a combined satisfactory geometric QC result



```
{
    "Project_id": "RevitSchoolUCL",
    "QCWorkOrderID": "WO201948",
    "AsBuiltPointClouds": {
        "18YHwga450Mw4Fy6M5t_8F":
        "C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolUCL/18YHwga450Mw4Fy6M5t_8F_voxelSpace_pointCloud.ply",
        "1WrzGm1SD2ev45B_OWQ3EP":
        "C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolUCL/1WrzGm1SD2ev45B_OWQ3EP_voxelSpace_pointCloud.ply",
        "2UD3D7uxP8kecbbBCRtzBP":
        "C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolUCL/2UD3D7uxP8kecbbBCRtzBP_voxelSpace_pointCloud.ply",
        "2UD3D7uxP8kecbbBCRtzCL":
        "C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolUCL/2UD3D7uxP8kecbbBCRtzCL_voxelSpace_pointCloud.ply"
    },
    "AsBuiltMeshes": {
        "18YHwga450Mw4Fy6M5t_8F":
        "C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolUCL/18YHwga450Mw4Fy6M5t_8F_AsBuiltMesh.obj",
        "1WrzGm1SD2ev45B_OWQ3EP":
        "C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolUCL/1WrzGm1SD2ev45B_OWQ3EP_AsBuiltMesh.obj",
        "2UD3D7uxP8kecbbBCRtzBP":
        "C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolUCL/2UD3D7uxP8kecbbBCRtzBP_AsBuiltMesh.obj",
        "2UD3D7uxP8kecbbBCRtzCL":
        "C:\\COGITO_Repos\\COGITO_GeomQC\\build\\Output\\RevitSchoolUCL/2UD3D7uxP8kecbbBCRtzCL_AsBuiltMesh.obj"
    }
}
```

Figure 29: Revit Technical School QC as-built geometries example output



```
{
    "Project_id": "RevitSchoolUCL",
    "QCWorkOrderID": "WO201948",
    "QC": {
        "Project_RevitSchoolUCL_QC1255": {
            "QC_UID": "Project_RevitSchoolUCL_QC1255",
            "Result": "Fail",
            "TimestampPerformed": "2022-10-10T11:50:24",
            "Unit": [
                "distance",
                "metres"
            ],
            "ScalarResult": [
                0.0532898902893066 4
            ],
            "ToleranceReference": [
                0.0256000589579343 8
            ]
        },
        "Project_RevitSchoolUCL_QC255": {
            "QC_UID": "Project_RevitSchoolUCL_QC255",
            "Result": "Pass",
            "TimestampPerformed": "2022-10-10T11:50:24",
            "Unit": [
                "distance",
                "metres"
            ],
            "ScalarResult": [
                0.00067138671875
            ],
            "ToleranceReference": [
                0.019999999552965165
            ]
        },
        "Project_RevitSchoolUCL_QC3151": {
            "QC_UID": "Project_RevitSchoolUCL_QC3151",
            "Result": "Fail",
            "TimestampPerformed": "2022-10-10T11:50:26",
            "Unit": [
                "distance",
                "metres"
            ],
            "ScalarResult": [
                0.0004500746726989746,
                0.021094322204589845
            ],
            "ToleranceReference": [
                0.015000001527369023,
                0.014999999664723874
            ]
        }
    }
}
```

Figure 30: Revit Technical School QC results output

4.5 Licensing

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

4.6 Installation Instructions

No complex installation is required. The code will be provided in the Cyberbuild Datashare collections and Github. 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 considered to be finished. Extra functionalities may nonetheless be found to be missing during the pre-validation and validation phase, at which point they will be added. Regarding the integration status, the *GeometricQC tool active geometric quality control analysis* component only interacts directly with the GeometricQC tool communication component. However, as described in section 4.3, it has indirect interaction with the DTP. As such, it follows and uses the established templates described in sections 4.3.1 and 4.3.2 as input and output integration, and follows the API described in 4.3.3.

4.8 Requirements Coverage

Despite the fact that the *GeometricQC tool active geometric quality control analysis* component is just a back-end part of the GeometricQC tool, it already covers a couple of the requirements defined in D2.5 and D2.1.

The functional and non-functional requirements from the COGITO System Architecture that are already covered are presented in Table 6. Functional requirement Req-1.1 is covered by the integration of the *BIM Element Manager,* the *Scan-vs-BIM solution*, and the Open3D library. Another part of the GeometricQC tool has the charge of dealing with the schedule part of the 4D BIM. Req-1.2 is 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 EN 13670:2009 [2] and the steel standard EN1090-2:2018 [1], 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 6: *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	Achieved
Req-1.2	Object detection in point cloud	Functional	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 7 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 achieved by using the Open3D library, which is able to handle the point cloud PLY format and the E57 library and wrapper. 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 7: *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	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 second and final version of the *GeometricQC tool active geometric quality control analysis* 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. Support for IFC version 4.3 remains under development driven by the emerging needs of the pre-validation and validation activities and the capability of the Digital Twin Platform.
- It requires the input IFC file to be syntactically correct (i.e. consistent with the IFC 4.0 schema) and free of geometric errors.
- 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 project 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 acquired by a TLS.
- The input as-built data has to be accurately pre-registered in the same coordinate system as the BIM file (i.e. the project coordinate system).

- 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 GeometricQC tool communication component

In this section we describe the *GeometricQC tool communication* component. This component is an internet application that serves as a middleware connecting the GeometricQC tool with the DTP. The main responsibilities of this component are:

- Listening for requests from the Digital Twin Platform to execute the GeometricQC tool.
- Request input data for the GeometricQC tool.
- Data exchange between the DTP and the GeometricQC tool.

The component uses HTTPS protocol for retrieving the parameters files required for the GeometricQC tool.

The *GeometricQC tool communication* component runs on the same platform as the Geometric QC tool.

5.1 Prototype Overview

The objective of the *GeometricQC communication* component is to enable the GeometricQC tool to receive requests from the DTP and to send back the output results to it, so they can be used by the rest of the COGITO tools, such as DigitAR and the DCC.

The geometric quality control sequence is initiated by the DTP sending a notification to the *GeometricQC tool communication* component. First, the component has to identify which of the two other GeometriQC tool components has to be executed (offline processing or active geometric QC analysis). After that, it requests the relevant input data from the DTP, and once it is downloaded, the component formats the data following the templates described in sections 3.3.1 and 4.3.1. The *GeometricQC communication* component then executes the relevant GeometricQC processing, and waits for it to finish. When the process is finished, the tool uploads the results to the DTP. Figure 31 shows a detailed component workflow.

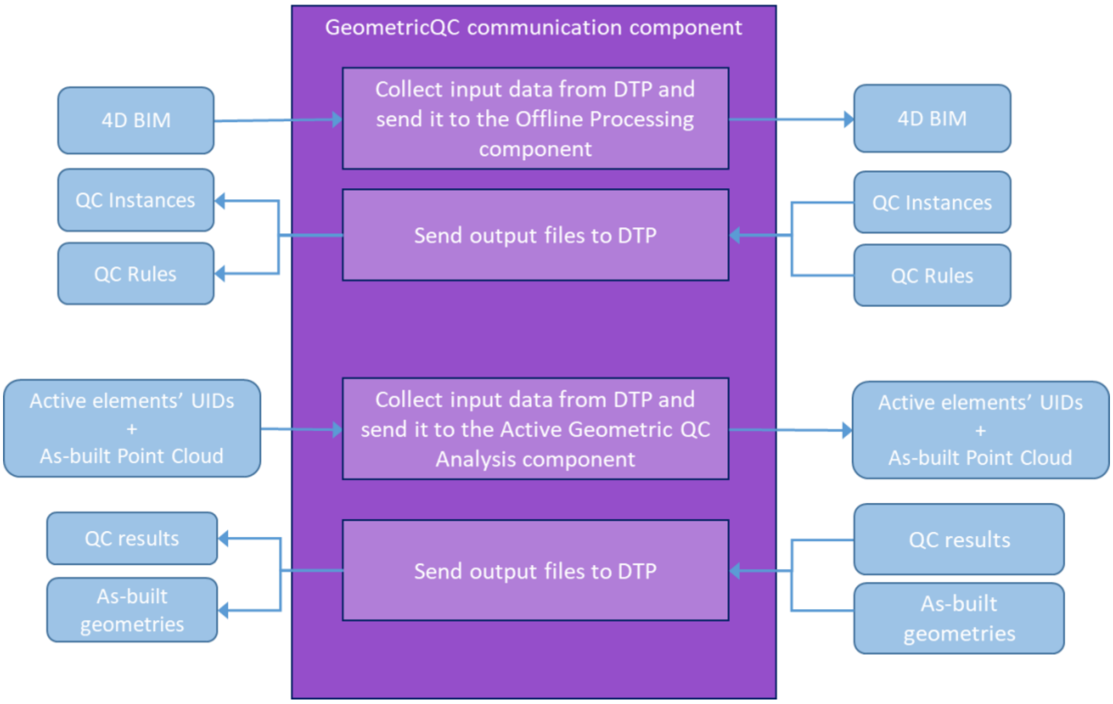

GeometricQC communication component
4D BIM
Collect input data from DTP and send it to the Offline Processing component
4D BIM
QC Instances
QC Rules
Send output files to DTP
QC Instances
QC Rules
Active elements' UIDs + As-built Point Cloud
Collect input data from DTP and send it to the Active Geometric QC Analysis component
Active elements' UIDs + As-built Point Cloud
QC results
As-built geometries
Send output files to DTP
QC results
As-built geometries

Figure 31: GeometricQC tool communication component detailed workflow

5.2 Technology Stack and Implementation Tools

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

- **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.
- **OpenSSL**: a library providing tools for secure and encrypted internet communication. Used for establishing a transport layer security connection with the Digital Twin Platform.

Table 8: Libraries and Technologies used in the *GeometricQC communication* sub-component

Library/Technology Name	Version	License
Boost	1.76.0	Boost Software License 1.0
OpenSSL	3.0	Apache License v2

5.3 Input, Output and API Documentation

The GeometricQC tool is used as the processing tool for UC2.1, as defined in D2.1. As can be seen in Figure 32, the two different GeometricQC tool modes (offline pre-processing and active processing) are well defined and differentiated. The GeometricQC tool communication component is in charge of receiving, requesting, formatting the different communications and data exchanges between the DTP and the GeometricQC (see arrows between DT Platform and GeometricQC in sequence diagram) as well as triggering the relevant GeometricQC data processing services.

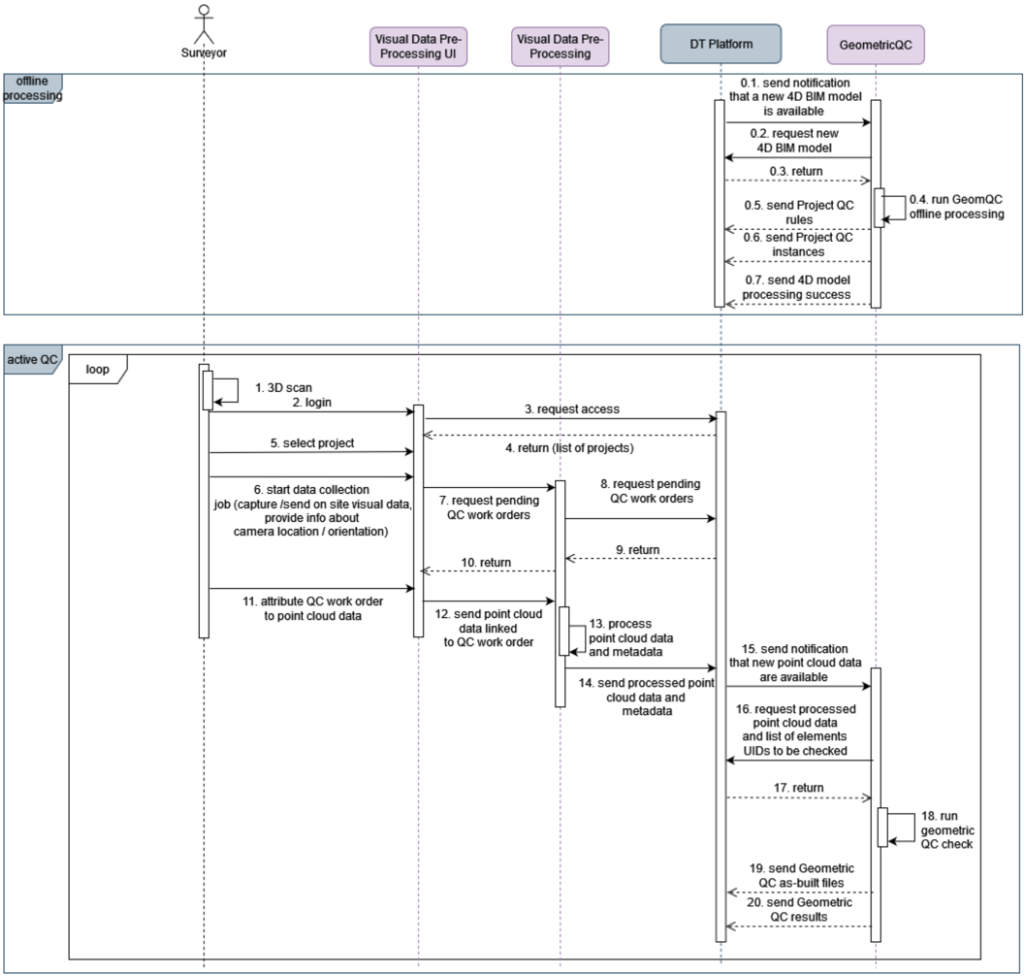
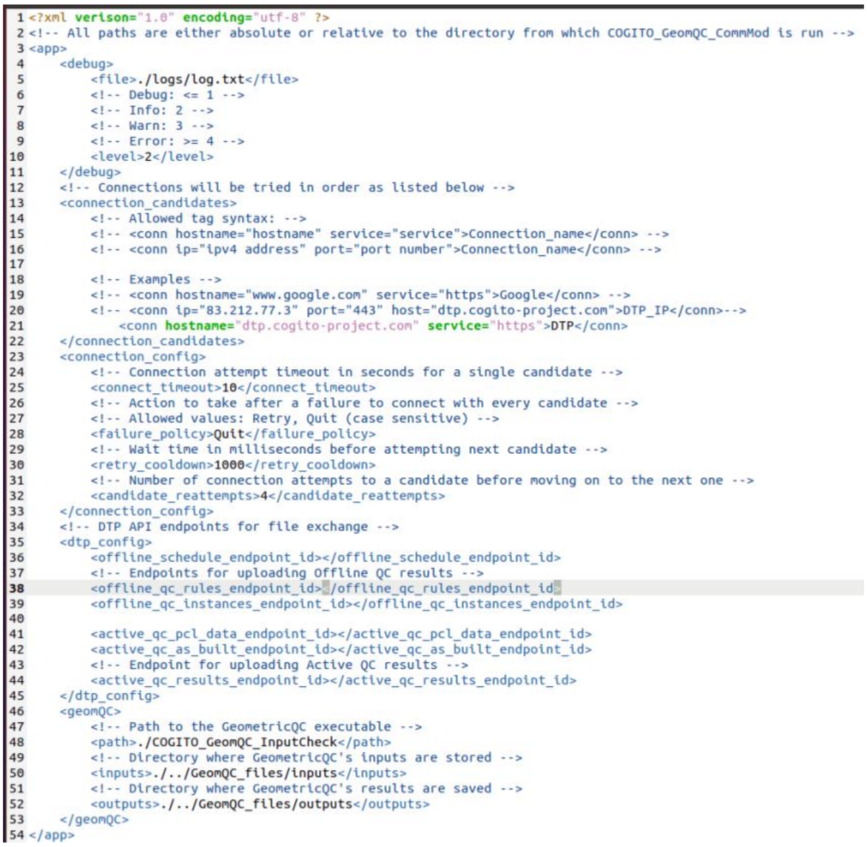

Surveyor
Visual Data Pre-Processing UI
Visual Data Pre-Processing
DT Platform
GeometricQC
offline processing
0.1. send notification that a new 4D BIM model is available
0.2. request new 4D BIM model
0.3. return
0.4. run GeomQC offline processing
0.5. send Project QC rules
0.6. send Project QC instances
0.7. send 4D model processing success
active QC
loop
1. 3D scan
2. login
3. request access
4. return (list of projects)
5. select project
6. start data collection job (capture /send on site visual data, provide info about camera location / orientation)
7. request pending QC work orders
8. request pending QC work orders
9. return
10. return
11. attribute QC work order to point cloud data
12. send point cloud data linked to QC work order
13. process point cloud data and metadata
14. send processed point cloud data and metadata
15. send notification that new point cloud data are available
16. request processed point cloud data and list of elements UIDs to be checked
17. return
18. run geometric QC check
19. send Geometric QC as-built files
20. send Geometric QC results

Figure 32: UC2.1 Sequence diagram

5.3.1 Input

Before executing the *GeometricQC tool communication* component, the environmental variable ***DTP_API_KEY*** needs to be set on the system. The *GeometricQC tool communication* component uses the variable value as an API key for requesting and uploading files to the DTP.

The *GeometricQC tool communication* component also requires a configuration file, defined using the XML format shown in Figure 33.



```xml
<?xml verison="1.0" encoding="utf-8" ?>
<!-- All paths are either absolute or relative to the directory from which COGITO_GeomQC_CommMod is run -->
<app>
    <debug>
        <file>./logs/log.txt</file>
        <!-- Debug: <= 1 -->
        <!-- Info: 2 -->
        <!-- Warn: 3 -->
        <!-- Error: >= 4 -->
        <level>2</level>
    </debug>
    <!-- Connections will be tried in order as listed below -->
    <connection_candidates>
        <!-- Allowed tag syntax: -->
        <!-- <conn hostname="hostname" service="service">Connection_name</conn> -->
        <!-- <conn ip="ipv4 address" port="port number">Connection_name</conn> -->

        <!-- Examples -->
        <!-- <conn hostname="www.google.com" service="https">Google</conn> -->
        <!-- <conn ip="83.212.77.3" port="443" host="dtp.cogito-project.com">DTP_IP</conn>-->
            <conn hostname="dtp.cogito-project.com" service="https">DTP</conn>
    </connection_candidates>
    <connection_config>
        <!-- Connection attempt timeout in seconds for a single candidate -->
        <connect_timeout>10</connect_timeout>
        <!-- Action to take after a failure to connect with every candidate -->
        <!-- Allowed values: Retry, Quit (case sensitive) -->
        <failure_policy>Quit</failure_policy>
        <!-- Wait time in milliseconds before attempting next candidate -->
        <retry_cooldown>1000</retry_cooldown>
        <!-- Number of connection attempts to a candidate before moving on to the next one -->
        <candidate_reattempts>4</candidate_reattempts>
    </connection_config>
    <!-- DTP API endpoints for file exchange -->
    <dtp_config>
        <offline_schedule_endpoint_id></offline_schedule_endpoint_id>
        <!-- Endpoints for uploading Offline QC results -->
        <offline_qc_rules_endpoint_id></offline_qc_rules_endpoint_id>
        <offline_qc_instances_endpoint_id></offline_qc_instances_endpoint_id>

        <active_qc_pcl_data_endpoint_id></active_qc_pcl_data_endpoint_id>
        <active_qc_as_built_endpoint_id></active_qc_as_built_endpoint_id>
        <!-- Endpoint for uploading Active QC results -->
        <active_qc_results_endpoint_id></active_qc_results_endpoint_id>
    </dtp_config>
    <geomQC>
        <!-- Path to the GeometricQC executable -->
        <path>./COGITO_GeomQC_InputCheck</path>
        <!-- Directory where GeometricQC's inputs are stored -->
        <inputs>./../GeomQC_files/inputs</inputs>
        <!-- Directory where GeometricQC's results are saved -->
        <outputs>./../GeomQC_files/outputs</outputs>
    </geomQC>
</app>
```

Figure 33: GeometricQC tool communication component XML configuration file

The GeometricQC tool inputs are stored in the DTP API's collections. The *GeometricQC tool communication* component parses XML file and requests data from endpoints specified by IDs in the configuration file.

The different inputs, for the *GeometricQC tool offline processing* and *GeometricQC tool active geometric quality control analysis* components (arrow 0.3 in and arrow 17, respectively, in the sequence diagram) can be seen in Figure 10 and Figure 21.

5.3.2 Output

In order to send back the generated output results from the GeometricQC tool, the *GeometricQC communication* component has to create different collections in the DTP API (endpoints specified in the XML file), and upload the Geometric QC results there. The output files, as already explained in sections 3.3.2 and 4.3.2, are JSON files and contain the content defined in the templates presented in Figure 12, Figure 11, Figure 22 and Figure 23.

5.3.3 API documentation

As part of the API to be able to run the *GeometricQC tool communication* component (and by extension the GeometricQC tool), the only requirement is that the environmental variable ***DTP_API_KEY*** needs to be set in the system. Once the variable is set and the XML file is ready, the *GeometricQC tool communication* component can be started with the command:

".\COGITO_GeomQC_CommMod.exe .\config\win_config.xml"

The only argument the component requires is a path to the configuration file.

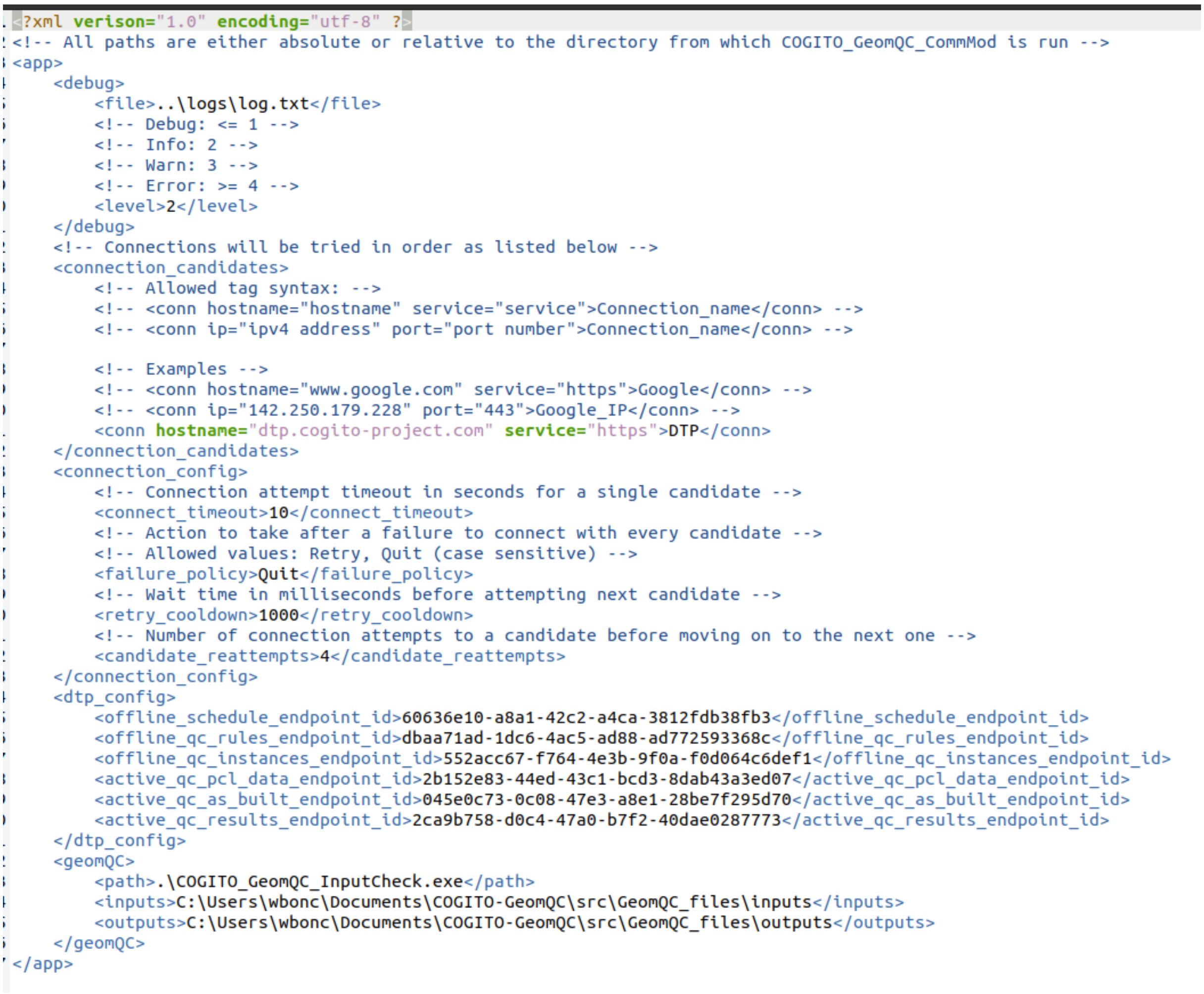
5.4 Application Example

The *GeometricQC tool communication* component is essentially used to interact with the DTP, obtain and prepare all the input data, execute the relevant processing mode, and return the generated output to the DTP. For such, it only requires to be an active running process and have internet connection to keep the communication with the DTP.

5.4.1 Revit Technical School model

The Revit Sample Project Technical School is a structural sample model provided by Autodesk [5]. Please check sections 3.4.1 and 4.4.1 for more details.

The *GeometricQC tool communication* component is launched with the command described in 5.3.3. DTP endpoints that are used by the *GeometricQC tool communication* component are inserted in appropriate XML tags:

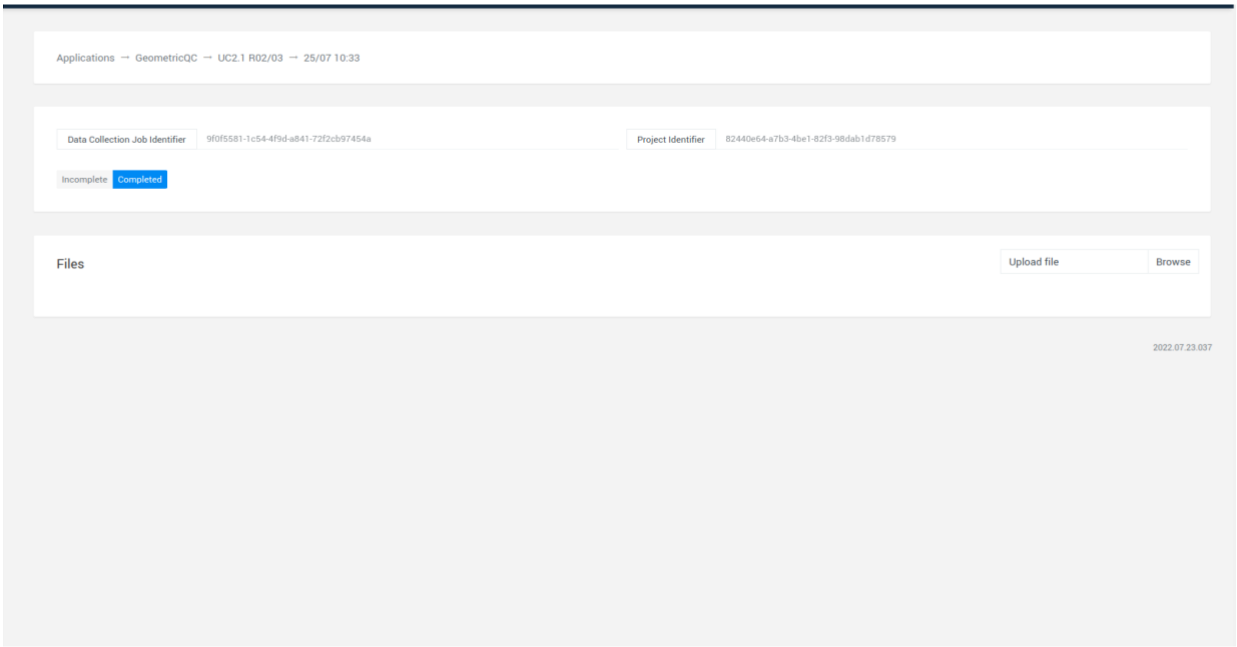


```
<?xml verison="1.0" encoding="utf-8" ?>
<!-- All paths are either absolute or relative to the directory from which COGITO_GeomQC_CommMod is run -->
<app>
    <debug>
        <file>..\logs\log.txt</file>
        <!-- Debug: <= 1 -->
        <!-- Info: 2 -->
        <!-- Warn: 3 -->
        <!-- Error: >= 4 -->
        <level>2</level>
    </debug>
    <!-- Connections will be tried in order as listed below -->
    <connection_candidates>
        <!-- Allowed tag syntax: -->
        <!-- <conn hostname="hostname" service="service">Connection_name</conn> -->
        <!-- <conn ip="ipv4 address" port="port number">Connection_name</conn> -->

        <!-- Examples -->
        <!-- <conn hostname="www.google.com" service="https">Google</conn> -->
        <!-- <conn ip="142.250.179.228" port="443">Google_IP</conn> -->
        <conn hostname="dtp.cogito-project.com" service="https">DTP</conn>
    </connection_candidates>
    <connection_config>
        <!-- Connection attempt timeout in seconds for a single candidate -->
        <connect_timeout>10</connect_timeout>
        <!-- Action to take after a failure to connect with every candidate -->
        <!-- Allowed values: Retry, Quit (case sensitive) -->
        <failure_policy>Quit</failure_policy>
        <!-- Wait time in milliseconds before attempting next candidate -->
        <retry_cooldown>1000</retry_cooldown>
        <!-- Number of connection attempts to a candidate before moving on to the next one -->
        <candidate_reattempts>4</candidate_reattempts>
    </connection_config>
    <dtp_config>
        <offline_schedule_endpoint_id>60636e10-a8a1-42c2-a4ca-3812fdb38fb3</offline_schedule_endpoint_id>
        <offline_qc_rules_endpoint_id>dbaa71ad-1dc6-4ac5-ad88-ad772593368c</offline_qc_rules_endpoint_id>
        <offline_qc_instances_endpoint_id>552acc67-f764-4e3b-9f0a-f0d064c6def1</offline_qc_instances_endpoint_id>
        <active_qc_pcl_data_endpoint_id>2b152e83-44ed-43c1-bcd3-8dab43a3ed07</active_qc_pcl_data_endpoint_id>
        <active_qc_as_built_endpoint_id>045e0c73-0c08-47e3-a8e1-28be7f295d70</active_qc_as_built_endpoint_id>
        <active_qc_results_endpoint_id>2ca9b758-d0c4-47a0-b7f2-40dae0287773</active_qc_results_endpoint_id>
    </dtp_config>
    <geomQC>
        <path>.\COGITO_GeomQC_InputCheck.exe</path>
        <inputs>C:\Users\wbonc\Documents\COGITO-GeomQC\src\GeomQC_files\inputs</inputs>
        <outputs>C:\Users\wbonc\Documents\COGITO-GeomQC\src\GeomQC_files\outputs</outputs>
    </geomQC>
</app>
```

Figure 34: The configuration file used in the example. Endpoints specified by the content of *dtp_config* tag must be created before running the application.

Initially, the *offline_schedule_endpoint_id* endpoint contains an empty data collection (Figure 35).



Applications → GeometricQC → UC2.1 R02/03 → 25/07 10:33

Data Collection Job Identifier 9f0f5581-1c54-4f9d-a841-72f2cb97454a

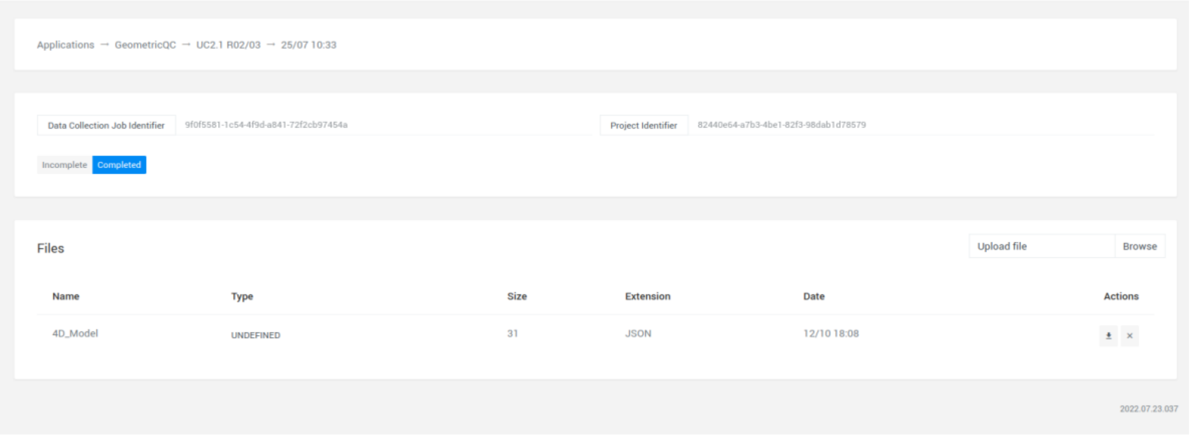
Project Identifier 82440e64-a7b3-4be1-82f3-98dab1d78579

Files

2022.07.23.037

Figure 35: A data collection for schedule files.

The *GeometricQC tool communication* component requests a new 4D BIM. Then, a new JSON file is available to download from the data collection (Figure 36).



Applications → GeometricQC → UC2.1 R02/03 → 25/07 10:33

Data Collection Job Identifier 9f0f5581-1c54-4f9d-a841-72f2cb97454a

Project Identifier 82440e64-a7b3-4be1-82f3-98dab1d78579

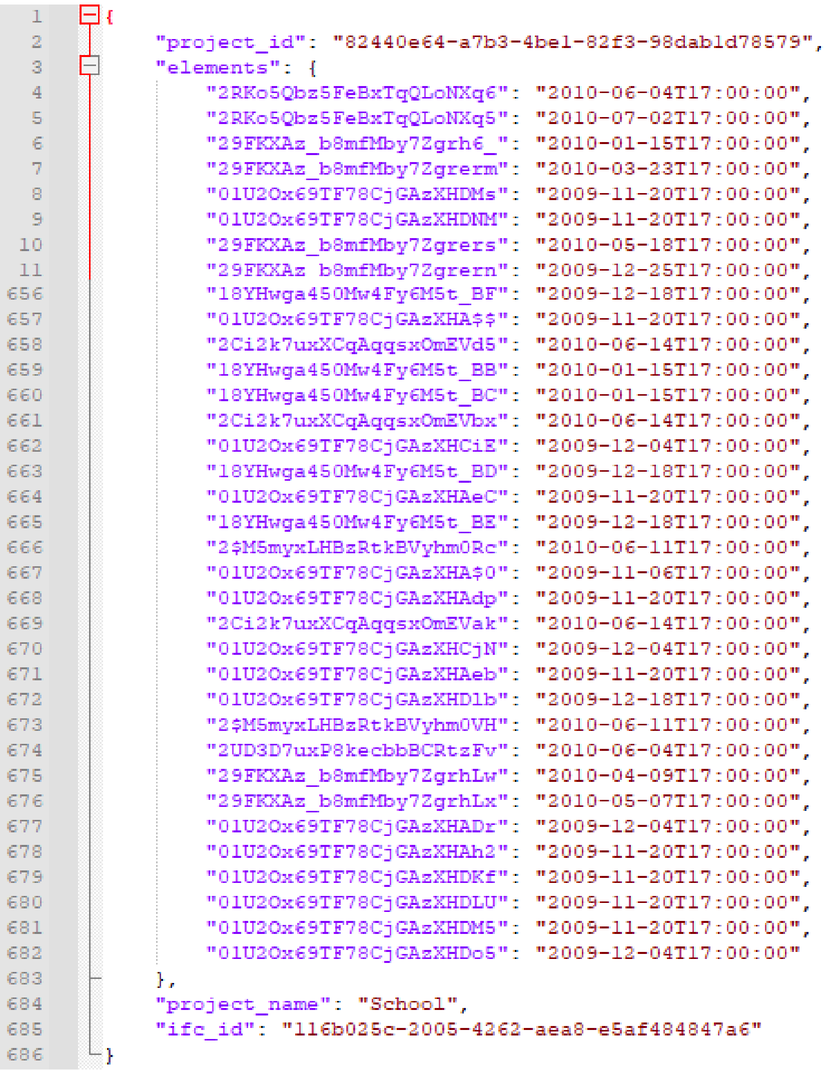
Files

Name	Type	Size	Extension	Date	Actions
4D_Model	UNDEFINED	31	JSON	12/10 18:08	⭳ ×

2022.07.23.037

Figure 36: A new schedule file appears in the schedule data collection after requesting a 4D BIM.

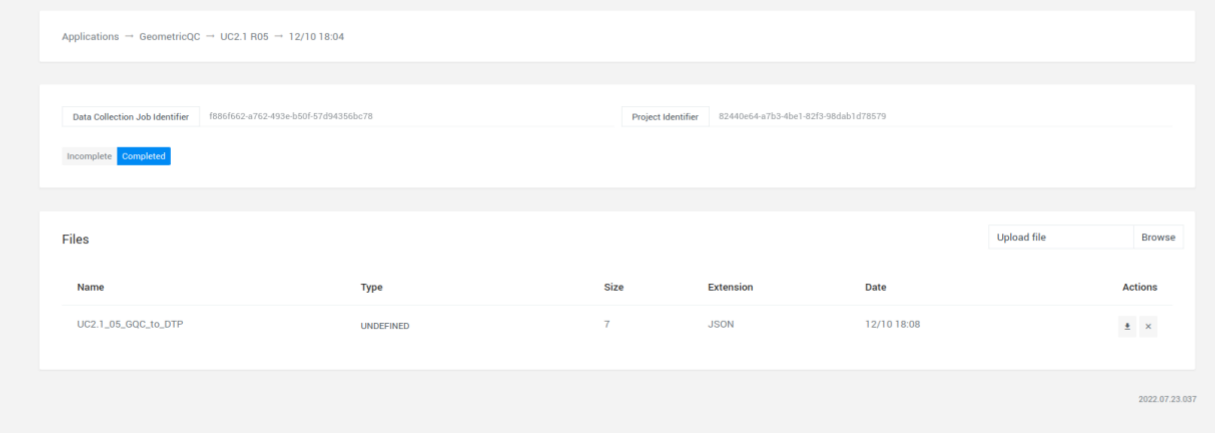
The new file contains the project’s id and list of elements and their timestamps (Figure 37).



```
{
    "project_id": "82440e64-a7b3-4be1-82f3-98dab1d78579",
    "elements": {
        "2RKo5Qbz5FeBxTqQLoNXq6": "2010-06-04T17:00:00",
        "2RKo5Qbz5FeBxTqQLoNXq5": "2010-07-02T17:00:00",
        "29FKXAz_b8mfMby7Zgrh6_": "2010-01-15T17:00:00",
        "29FKXAz_b8mfMby7Zgrerm": "2010-03-23T17:00:00",
        "01U2Ox69TF78CjGAzXHDMs": "2009-11-20T17:00:00",
        "01U2Ox69TF78CjGAzXHDNM": "2009-11-20T17:00:00",
        "29FKXAz_b8mfMby7Zgrers": "2010-05-18T17:00:00",
        "29FKXAz b8mfMby7Zgrern": "2009-12-25T17:00:00",
        "18YHwga450Mw4Fy6M5t_BF": "2009-12-18T17:00:00",
        "01U2Ox69TF78CjGAzXHA$$": "2009-11-20T17:00:00",
        "2Ci2k7uxXCqAqqsxOmEVd5": "2010-06-14T17:00:00",
        "18YHwga450Mw4Fy6M5t_BB": "2010-01-15T17:00:00",
        "18YHwga450Mw4Fy6M5t_BC": "2010-01-15T17:00:00",
        "2Ci2k7uxXCqAqqsxOmEVbx": "2010-06-14T17:00:00",
        "01U2Ox69TF78CjGAzXHCiE": "2009-12-04T17:00:00",
        "18YHwga450Mw4Fy6M5t_BD": "2009-12-18T17:00:00",
        "01U2Ox69TF78CjGAzXHAeC": "2009-11-20T17:00:00",
        "18YHwga450Mw4Fy6M5t_BE": "2009-12-18T17:00:00",
        "2$M5myxLHBzRtkBVyhm0Rc": "2010-06-11T17:00:00",
        "01U2Ox69TF78CjGAzXHA$0": "2009-11-06T17:00:00",
        "01U2Ox69TF78CjGAzXHAdp": "2009-11-20T17:00:00",
        "2Ci2k7uxXCqAqqsxOmEVak": "2010-06-14T17:00:00",
        "01U2Ox69TF78CjGAzXHCjN": "2009-12-04T17:00:00",
        "01U2Ox69TF78CjGAzXHAeb": "2009-11-20T17:00:00",
        "01U2Ox69TF78CjGAzXHD1b": "2009-12-18T17:00:00",
        "2$M5myxLHBzRtkBVyhm0VH": "2010-06-11T17:00:00",
        "2UD3D7uxP8kecbbBCRtzFv": "2010-06-04T17:00:00",
        "29FKXAz_b8mfMby7ZgrhLw": "2010-04-09T17:00:00",
        "29FKXAz_b8mfMby7ZgrhLx": "2010-05-07T17:00:00",
        "01U2Ox69TF78CjGAzXHADr": "2009-12-04T17:00:00",
        "01U2Ox69TF78CjGAzXHAh2": "2009-11-20T17:00:00",
        "01U2Ox69TF78CjGAzXHDKf": "2009-11-20T17:00:00",
        "01U2Ox69TF78CjGAzXHDLU": "2009-11-20T17:00:00",
        "01U2Ox69TF78CjGAzXHDM5": "2009-11-20T17:00:00",
        "01U2Ox69TF78CjGAzXHDo5": "2009-12-04T17:00:00"
    },
    "project_name": "School",
    "ifc_id": "116b025c-2005-4262-aea8-e5af484847a6"
}
```

Figure 37: The schedule file example.

In order to prepare the input data for the *GeometricQC tool offline processing*, the *GeometricQC tool communication* component retrieves the IFC file from the *ifc_id* value encoded in the schedule file, and downloads it from the relevant GeometricQC endpoint. Once the IFC and JSON files are available, all the required inputs are ready to be passed to the *GeometricQC offline processing.* The component then executes it and waits until the results are generated. Both results are then uploaded to DTP endpoints specified by *offline_qc_rules_endpoint_id* (Figure 38) and *offilne_qc_instances_endpoint_id* (Figure 39).



Applications → GeometricQC → UC2.1 R05 → 12/10 18:04

Data Collection Job Identifier: f886f662-a762-493e-b50f-57d94356bc78

Project Identifier: 82440e64-a7b3-4be1-82f3-98dab1d78579

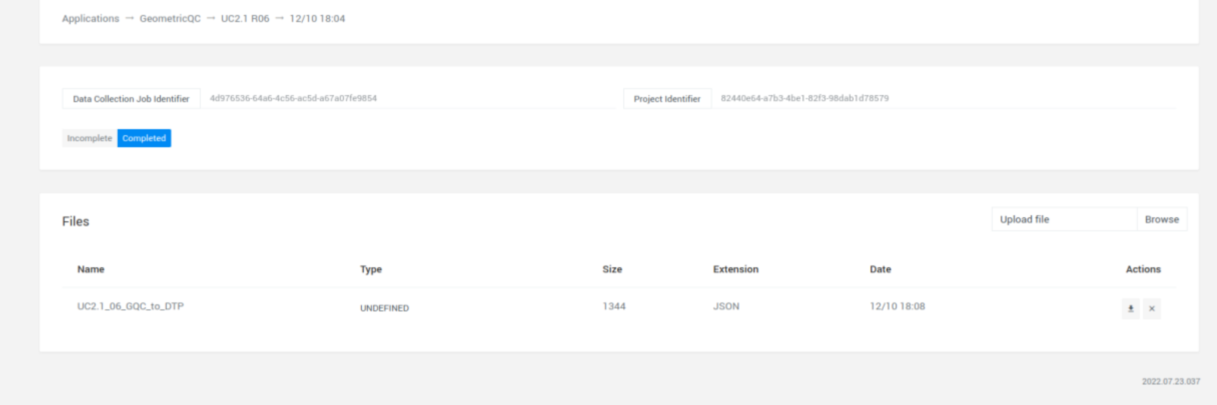
Incomplete | Completed

Files

Name	Type	Size	Extension	Date	Actions
UC2.1_05_GQC_to_DTP	UNDEFINED	7	JSON	12/10 18:08	⭳ ×

2022.07.23.037

Figure 38: The GeometricQC tool communication component creates new data collections in offline_qc_rules_endpoint_id DTP endpoint.



Applications → GeometricQC → UC2.1 R06 → 12/10 18:04

Data Collection Job Identifier: 4d976536-64a6-4c56-ac5d-a67a07fe9854

Project Identifier: 82440e64-a7b3-4be1-82f3-98dab1d78579

Incomplete | Completed

Files

Name	Type	Size	Extension	Date	Actions
UC2.1_06_GQC_to_DTP	UNDEFINED	1344	JSON	12/10 18:08	⭳ ×

2022.07.23.037

Figure 39: The GeometricQC tool communication component creates new data collections in offline_qc_instances_endpoint_id DTP endpoint.

5.5 Licensing

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

5.6 Installation Instructions

No complex installation is required. The code will be provided in the Cyberbuild Datashare collections and Github. 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.

5.7 Development and integration status

The *GeometricQC tool communication* component is still under development. Some features required to have a fully automated workflow are still pending and will be completed in the coming months as part of the integration work in WP8in WP8. At this stage, the component is able to request and download the required input data for the offline processing as described in the sequence diagram, and download manually the input data for the active processing. In both cases, the output can be uploaded to the DTP automatically. The notification interaction between the DTP and the *GeometricQC tool communication* component remains under development, however the communication exchange protocols have already been agreed.

5.8 Requirements Coverage

Table 9 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. Regarding COGI-QC-11, the *GeometricQC tool communication* component allows to gather input data, and generate the different calls required to execute the different GeometricQC tool modes, enabling to obtain the list of QC instances generated by the *GeometricQC offline processing* component and perform the geometric and dimensional tolerance verifications.

Table 9: *GeometricQC communication* 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-QC-11	Automates QC-related activities	• Functional • Operational	Must	Achieved

5.9 Assumptions and Restrictions

The *GeometricQC tool communication* component, at this stage, has been developed under the following assumptions:

- A notification system is in place between the DTP and the GeometricQC tool
- The DTP API's collections required to query the input data for both GeometricQC tool modes are in place and functioning.
- The DTP API's collections required to upload the output data for both GeometricQC tool modes are in place and functioning.

6 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 throughout the construction delivery phase. The BIM model data are provided in an IFC 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 (D5.6), the core functionalities of the GeometricQC tool are delivered by three components. The first component, namely the *GeometricQC offline processing* component, performs all the as-planned data analysis, by obtaining the 3D geometry from the IFC file, and generating a geometric relationship graph structure containing all the elements and relationships used to generate the list of QC instances. 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 D5.2) 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 as defined in the QC instances. The third component, the GeometricQC tool communication component, is in charge of getting data processing requests (with the relevant data) from the DTP, triggering the other two components as required, and sending the generated output files back to the DTP. These components have also been designed to be modular, and scalable, allowing the creation of a toolbox, which can be used independently, outside the COGITO solution.

The GeometricQC tool's core functionality have been demonstrated using a realistic example of a high school building structure. Integration with the rest of the COGITO solution, principally the DTP, is being completed within WP8. Adjustments and improvements to the GeometricQC components and sub-components will also be performed as the needs arise during the COGITO pre-validation and validation activities, that are also conducted in WP8.

References

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

[2] BS EN 13670:2009, "Execution of concrete structures," 2010.

[3] 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.

[4] 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.

[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