4 Object-Oriented Implementation of the DIM[1]

After the definition of the information model, a proper implementation in form of a data model has to be done. As BIM relies on IFC, the IFC standard is used as basis for the data model. Furthermore, the data model shall be visualized by available IFC software. Hence, testing and extending existing software is explained as well.

4.1 Object-oriented Implementation of the DIM based on IFC

Based on the model given in Figure 3.16, an implementation shall be done using an established AEC data model. As explained in Section 2.5, IFC has been chosen as a suitable data format. IFC already contains numerous classes to model building elements, materials, persons, processes, and more. Figure 4.1 shows an overview of the mappings between IFC entities and the classes in the object-oriented model. In the middle are the classes from the object-oriented model and on the outer left and right are the related IFC entities. The defect annotation may be represented by four different IFC entities. A proxy element is the most generic approach to represent a defect including a geometry independently from the damage type. However, proxy geometries are treated like a geometries of building

[1] This chapter contains republished work of a retracted article from ASCE [135]. The article has been retracted by the authors because of copy right issues [136]. All content, which was affected by the copy right issues, has been replaced, i.e., IFC code snippets in Figures 4.2, 4.4 to 4.6 and 4.9 have been revised, Figure 15 from the article has been replaced with Figure 4.10.

elements, i.e., they are visualized as spatial elements, which could be counter-intuitive in case of cracks or spalling. Better would be that a crack or spalling is subtracted from the geometry of the building element.

Annotations allow to add information to a building element. Despite of the traditional thought of a defect is an annotation, this entity may be used in seldom cases only because it is limited to 2D geometry as stated by the formal proposals of the IFC standard [13]. 2D geometries are suitable for defects, like cracks, in the form of crack maps or abrasion as marked area on a surface. Similarly, surface features may also represent defects that mainly affect the surface of a component. However, to accommodate the BIM concept, it should be omitted to use annotations for defects because according to the IFC standard, 'An annotation is a graphical representation [...] that adds a note or meaning to the objects [...]' [13].

In case of damage types that include geometry subtractions of the affected component, a voiding feature is suited best because the dedicated relationship *IfcRelVoidsElement* implies that the geometry of the voiding feature is subtracted from the related building element. Some examples for these damage types are cracks, spalling, and voids. Summarizing, depending on the damage type, there are four IFC entities that may be used for a single defect and the best suitable has to be chosen. Nonetheless, none of these entities represent a defect semantically correct, hence, a distinctive defect element should be included in the IFC standard to properly include damage information. In contrast to the suggestion of Tanaka, Nakajima, Egusa, *et al.* - adding three additional entities - it would be enough to add a single entity to properly include defects [93].

4.1.1 IFC Classes for Semantic Data

Up to now, there are no specific defect entities implemented in the IFC. However, the IFC offers several alternatives: *IfcProxy*, *IfcAnnotation*, *IfcSurfaceFeature*, and *IfcVoidingFeature*. Table 4.1 presents a comparison of the advantages and disadvantages of these IFC entities. *IfcProxy* is a generic entity, but the IFC 4 lists the proxy as deprecated and recommend using *IfcBuildingElementProxy* instead. A look at newer versions of the IFC reveals that the proxy is marked deprecated no longer. The DIM should be usable in future

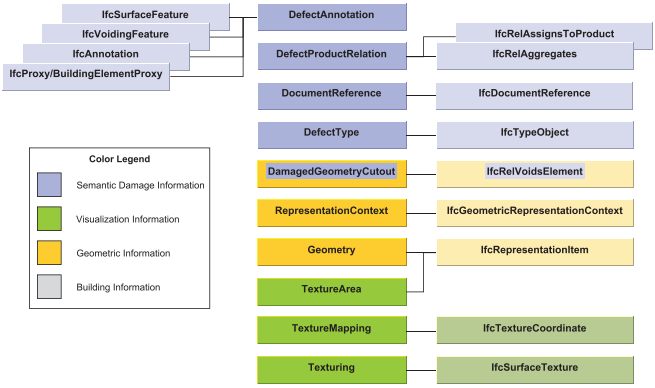


Figure 4.1: Mapping of the IFC entities onto the classes of the object-oriented Model.

standards, and hence, the proxy is taken as an option instead of *BuildingElementProxy*. A proxy is very flexible and thus suitable for every defect type. However, a proxy is treated as an individual element or building element, which conflicts with the nature of a defect. A defect cannot exist without the affected component. An *IfcAnnotation* may be used to add further annotations to a component. However, *IfcAnnotations* are only meant to have 0D, 1D, or 2D geometries. This circumstance limits their applicability to damages without or geometries or surface damages. *IfcSurfaceFeatures* and *IfcVoidingFeatures* are suitable for specific defects. Surface features are suitable for modeling corrosion or other defects, which only affect the surface of a component. Voiding features are suitable for defects like cracks or spalling, which add a void to a component. However, they are not suitable for modeling further damage types, e.g., material changes below the surface, divergences from specifications, or wash-outs. On the whole, depending on the defect, a suitable IFC entity has to be chosen. Corrosion or other surface changes claim for surface features. Cracks and spalling are represented by voiding features at best. Other defects could use either annotations or proxies.

In the next step, the analysis of suitable relationships from the IFC standard is presented. *IfcRel-Assigns* "is a generalization of 'link' relationships among instances of IfcObjects and its various 1st level subtypes" [144]. A specific identification of the relationship is

Table 4.1: Overview of the possible IFC entities with advantages and disadvantages

	IfcProxy	IfcAnnotation	IfcSurfaceFeature	IfcVoidingFeature
+	interpretable by most applications	add (textual) information about defect to component	suitable for specific defects,	suitable for specific defects,
			geometry is not visualized like geometries of components	geometry is not visualized like geometries of components
-	generic container,	less supported by applications,	only designed for modifications at surface,	only designed to reduce volume of element,
	independent object in contrast to a dependent defect	limited representations,	less supported by applications,	less supported by applications,
		modeling defects as annotations conflicts with the original meaning of annotations	modeling defects as surface features may conflict with original meaning of surface feature	modeling defects as voiding features may conflict with original meaning of voiding feature

Table 4.2: Overview of the possible IFC entities with advantages and disadvantages

	IfcRelAssignsToProduct	**IfcRelAggregates**	**IfcRelVoidsElement**
+	interpretable by most applications	interpretable by most applications	avoids additional data for geometry
-	not usable for defects, which are part of a component (cracks, spalling ...),	some defects are not part of a component (e.g. vegetation) parts	designed for voids only
		representation results from geometry of sub	less supported by applications

stored as the name of the relationship. In case of the *DefectProductRelation*, the name of *IfcRelAssignsToProdcut* would be "Defect product relation." However, a defect is part of a component and if the component is destroyed, the defect no longer exists. Hence, a composition is more precise. Strict compositions are modeled with *IfcRelAggregates*. Construction and design practice understands aggregations as a sum of different products. This would imply that a defect is a product if an aggregation is used, which is questionable. Altogether, aggregations seem to be the most precise relationship for physical defects. Both relationships the aggregation and the assignment may be used for other defects. In case of using *IfcVoidingFeature* to represent defects, the decomposition relationship *IfcRelVoidsElement* is suitable. "IfcRelVoidsElement is an objectified relationship between a building element and one opening element that creates a void in the element." [144]. As stated earlier, the voiding feature and the voids relationship are only applicable to cracks or spalling and not in case of material changes or other damage types. An example is depicted by Listing 4.2. *IfcRelAssignsToProdcut* may be used for effect-cause relations with the name "cause" or "reason." Additional information about the relation might be given by the description of the relationship. Table 4.2 provides an overview of the existing relationships and related advantages and disadvantages.

Figure 4.3 shows a schematic overview of one possible implementation of the DIM in IFC.

```
/* Building element */
#244= IFCBEAM('2tso43_ekkqjB6caA5ViEg',#42,
      'Test Beam', $,$,#242,#233,$,.BEAM.);

/* Defect Spalling */
#9002= IFCVOIDINGFEATURE('0nlZskSHuEqdlb4p01O5hg',
       #42,'Spalling','Spalling at beam',
       'Defect - Spalling', #8556,$,$,.CUTOUT.);
#9004= IFCRELVOIDSELEMENT('2hSrdH4wY0ynUPTlSyLhxw',
       #42,$,$,#244,#9002);
```

Figure 4.2: Extract of an IFC file modeling a damaged beam (#244) as damaged building element. The defect is represented by a voiding feature (#9002) and the voids relationship (#9004) models the relationship to the beam.

The defect is implemented as voiding feature in the middle. Properties are used to include measurements and other alpha-numeric data related to the defect. Type objects are utilized to add classifications to defects, such as spalling or crack. Document associations are able to relate external documents to the defect, for example, photos, reports, or testing results. On top, the relationship *IfcRelVoidsElement* connects the defect to the affected bridge element.

Figure 4.4 shows an excerpt to illustrate the incorporation of classification, measurements, and external documents. Entity #9000 is once again the defect. This defect is classified as spalling by the type object #9011. This classification may be hierarchical, for instance a classification for defects and a sub-class for spalling. #9021 is a property set of measurements for the spalling containing a diameter (#9022) and depth (#9023) with a unit (#43). At the end of the excerpt, is a reference to an external report of an ultrasonic investigation (#9031). Such external documents could also be photos and included together together with the IFC step file into an IFC-zip file.

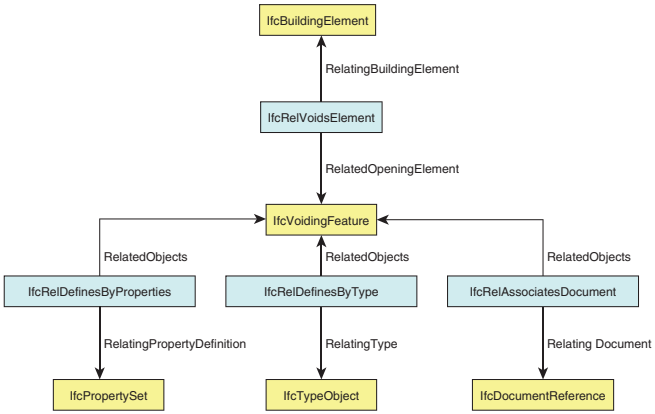


Figure 4.3: Block diagram of the resulting IFC structure. Yellow elements are instances and blue elements are relationships.

4.1.2 IFC Classes for Geometry Data

This subsection discusses the implementation of geometries of the DIM by using the IFC. Besides modeling the geometry of the damaged component, the method of modeling a defect geometry and the use of geometric representation contexts are illustrated.

Relationship-based geometry

This paragraph illustrates the implementation of the geometry model described in the Relationship-Based Geometry section and is related to Figure 3.14. Listing 4.5 shows an extract of an IFC file that contains an *IfcVoidingFeature* (#9002) as defect entity and a related component (#244). The *IfcRelVoidsElement* (#9004) represents the relationship between the defect and component and implies cutting the defect geometry out of the component geometry.

```
/* Defect Spalling */
#9000= IFCPROXY('0igGRCoTwk6XcjC1hqayEA',#42,
        'Spalling',$,'Defect',#242,$,.NOTDEFINED.,$);

/* Damage type */
#9010= IFCRELDEFINESBYTYPE('1bIoUtPdkkqxQGAgf-5vpA',
        #42,'Damage type','Typification of a defect',(#9000),
        #9011);
#9011= IFCTYPEOBJECT('10gzfsrKgEihXsn84QV_hQ',#42,
        'Damage type Spalling',$,'IfcProxy/Defect',$);

/* Measurements */
#9020= IFCRELDEFINESBYPROPERTIES('1S7kZBQd3USDFsV8uKQvDA',#42,
        'Defect Measurements','Diameter and depth of the
        spalling',(#9000),#9021);
#9021= IFCPROPERTYSET('0zky6s7LQ0CXauZHIiqYTA',#42,
        'Diameter and Depth',$,(#9022,#9023));
#9022= IFCPROPERTYSINGLEVALUE('Diameter',$,IFCREAL(151.0),#43);
#9023= IFCPROPERTYSINGLEVALUE('depth',$,IFCREAL(12.0),#43);
#43= IFCSIUNIT(*,.LENGTHUNIT.,.MILLI.,.METRE.);

/* Document */
#9030= IFCRELASSOCIATESDOCUMENT('2Nv5qvEsoku-QzHBcSpXXA',
    #42,'Report of ultra sonic survey',$,(#9000),#9031);
#9031= IFCDOCUMENTREFERENCE('http://standards.buildingsmart.org/
        IFC/RELEASE/IFC4_1/FINAL/EXPRESS/IFC4x1.exp',
        'U_S_16092020-42','Ultra Sonic report 16092020-42 from
        the 16th September 2020',$);
```

Figure 4.4: Part of an IFC file modeling a proxy for the defect (#9000) and a type object (#9011) to define a damage type namely 'Damage type Spalling'. Additionally, some measurements (#9021) and a reference to an external document (#9031) are included.

```
/* Building element */
#244= IFCBEAM('2tso43_ekkqjB6caA5ViEg',#42,
        'Test Beam',$,$,#242,#233,$,.BEAM.);
#233= IFCPRODUCTDEFINITIONSHAPE($,$,(#227,#231));
#231= IFCSHAPEREPRESENTATION(#103,'Axis','MappedRepresentation',
        (#229));

/* Defect Spalling */
#9002= IFCVOIDINGFEATURE('3-7hkhVEek2181I_ZN1Gzg',#42,
        'Spalling','Spalling at beam','Defect - Spalling',
        #8556,$,$,.CUTOUT.);
#9004= IFCRELVOIDSELEMENT('191hmq9RzkaK7Q65Oo6tKw',
        #42,$,$,#244,#9002);

        /* Defect Geometry */
#9003= IFCPRODUCTDEFINITIONSHAPE($,$,(#8548));
#8548= IFCSHAPEREPRESENTATION(#105,'Body',
        'MappedRepresentation',(#8546));
#8546= IFCMAPPEDITEM(#8542,#232);
#8542= IFCREPRESENTATIONMAP(#8541,#8539);
#8539= IFCSHAPEREPRESENTATION(#105,'Body','SweptSolid',(#8538));
#8538= IFCEXTRUDEDAREASOLID(#8534,#8537,#20,250.);
#8526= IFCCARTESIANPOINTLIST2D(((-125.,-30.),(125.,-30.),
        (125.,30.),(-125.,30.),(-125.,-30.)));
#8533= IFCINDEXEDPOLYCURVE(#8526,$,.F.);
#8534= IFCARBITRARYCLOSEDPROFILEDEF(.AREA.,'Box',#8533);
```

Figure 4.5: Extract of an IFC file modeling a beam (#244) as damaged building element. The defect is represented by a voiding feature (#9002) and the voids relationship (#9004) represents the relationship between the beam and the defect.

Independent relationship and geometry

This section illustrates the implementation of the geometry model described in the Independent Relationship and Geometry section and is related to Figure 3.15. In case of storing the geometry of the damaged component in the IFC, representation contexts are chosen to distinguish the geometries of intact and damaged components. A product might have multiple representations and every representation has a different representation context. Listing 4.6 illustrates the use of multiple geometries and representation contexts. The defect and the beam have their own geometries as shown by entities #9003 and #233. In this context, the damaged geometry of the beam, entity #9100, is a CSG geometry with a subtraction of the undamaged beam and the defect geometry. An example is depicted in Figure 4.7. The beam without any defects is shown on the left. In the middle is an exemplary cuboid defect and on the right is the beam with the cuboid damage geometry as cutout.

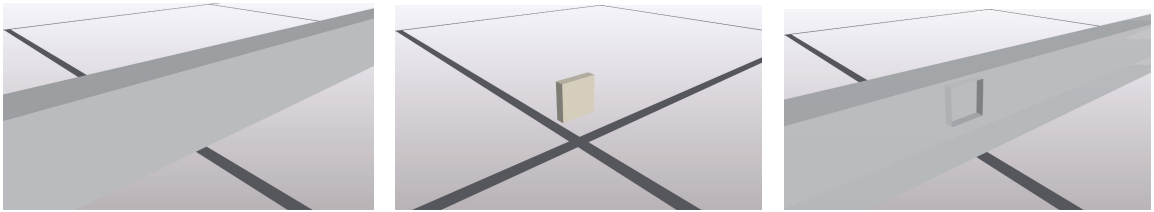
4.1.3 IFC Classes for Geometric-semantic Data

Geometric-semantic data may be stored as document references or as textures, which are depicted on a 3D surface. Document references have been discussed in the FC Classes for Semantic Data section. Coming to the implementation of textures, Figure 4.8 illustrates how to include a texture in an IFC file. To position an image, for example, a PNG-file, within the 3D model, a geometry is necessary. This geometry is represented by the *IfcRepresentationItem*. Such a geometry could be a plane, which carries the texture slightly above the related position of the affected component. A listing example can be found in the study by [89]. In addition to the texture itself, the mapping is necessary. The IFC offers the class *IfcTextureCoordinate* to add texture-mapping information and subclasses, such as *IfcTextureCoordinateGenerator* and *IfcIndexedTriangleTextureMap* to either define an algorithmic or point based texture mapping.

Texturing is a special method to include geometric-semantic data and needs a mapping algorithm to correctly depict the texture on the geometry. Figure 4.9 shows how to achieve that using IFC. Again, the defect is defined as proxy (#9000) with a simple cuboid geometry

```
/* Building Element */
#244= IFCBEAM('2tso43_ekkqjB6caA5ViEg',#42,
        'Test Beam', $,$,#242,#233,$,.BEAM.);

/* Undamaged Geometry */
#105= IFCGEOMETRICREPRESENTATIONSUBCONTEXT('Body','Model',*,*,
        *,*,#99,$,.MODEL_VIEW.,$);
#155= IFCEXTRUDEDAREASOLID(#149,#154,#20,12125.4);
#187= IFCREPRESENTATIONMAP(#186,#165);
#225= IFCMAPPEDITEM(#187,#224);
#227= IFCSHAPEREPRESENTATION(#105,'Body','MappedRepresentation',
        (#225));
#233= IFCPRODUCTDEFINITIONSHAPE($,$,(#227,#231,#9100));

/* Defect Spalling */
#9000= IFCPROXY('0igGRCoTwk6XcjC1hqayEA',#42,
        'Spalling',$,'Defect',#242,#9003,.NOTDEFINED.,$);

/* Defect Geometry */
#8526= IFCCARTESIANPOINTLIST2D(((-125.,-30.),(125.,-30.),
        (125.,30.),(-125.,30.),(-125.,-30.)));
#8533= IFCINDEXEDPOLYCURVE(#8526,$,.F.);
#8534= IFCARBITRARYCLOSEDPROFILEDEF(.AREA.,'Box',#8533);
#8538= IFCEXTRUDEDAREASOLID(#8534,#8537,#20,250.);
#8539= IFCSHAPEREPRESENTATION(#9050,'Body','SweptSolid',(#8538));
#9003= IFCPRODUCTDEFINITIONSHAPE($,$,(#8539));
#9050= IFCGEOMETRICREPRESENTATIONSUBCONTEXT('Defect Geometry',
        'Defect Geometry',*,*,*,*,#9051,$,.MODEL_VIEW.,$);
#9051= IFCGEOMETRICREPRESENTATIONCONTEXT('Defect',
        'Model' ,3,0.01,#96,#97);

/* Damaged Component Geometry*/
#9100= IFCSHAPEREPRESENTATION(#9150,'Body','CSG',(#9101));
#9101= IFCCSGSOLID(#9102);
#9102= IFCBOOLEANRESULT(.DIFFERENCE.,#155,#8538);
#9150= IFCGEOMETRICREPRESENTATIONSUBCONTEXT('Damaged Components',
        'Damage Model',*,*,*,*,#9151,$,.MODEL_VIEW.,$);
#9151= IFCGEOMETRICREPRESENTATIONCONTEXT('Damaged-geometry',
        'Model',3,0.01,#96,#97);
```

Figure 4.6: Excerpt of an IFC file modeling a distinct geometry and relationship. An assignment (#9001) represents the relationship between the beam (#244) and the defect (#9000). The beam has two geometries: a damaged geometry (#9100) and an undamaged geometry (#227).



a) b) c)

Figure 4.7: A geometric damage representation by using CSG in the 3D view. a) shows the undamaged beam, b) the defect geometry, and c) the damaged beam.

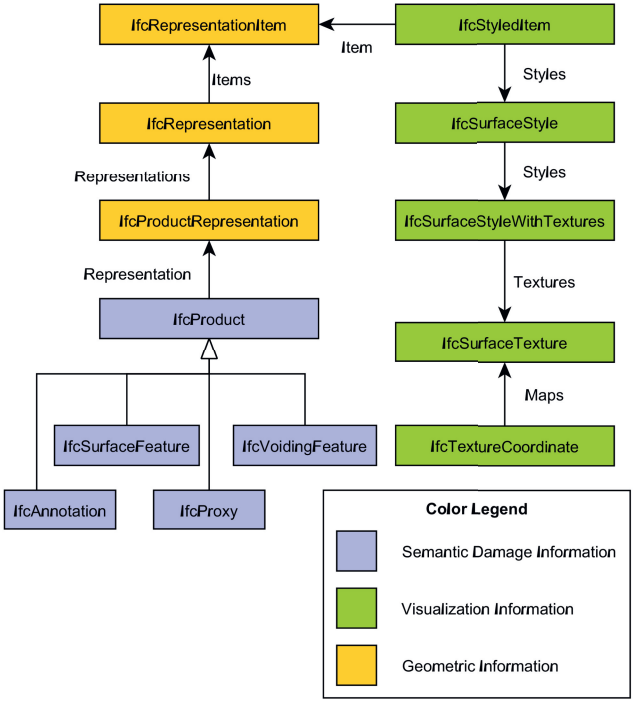


Figure 4.8: Damage model with texture using *IfcSurfaceFeature* and related elements. The defect is shown in orange. Multiple superclasses, subclasses and selects are omitted for simplicity. [89]

Table 4.3: Overview of tested BIM authoring software and IFC viewers

Authoring software		**IFC viewers**	
Autodesk Revit 2019	[17]	apstex IFC viewer	[23]
		BIM Vision	[22]
		Desite BIM	[19]
		Solibri Model Viewer	[24]
		usBIM	[20]
		xBIM Xplorer	[21]

(#8538). In addition to that, a texture in form of a JPG-file shall be depicted on the geometry (#10000). To create a correct visualization, the spherical mapping algorithm shall be used for this texture (#10061). Further details about the modeling may be found in Chapter 3.

4.2 IFC Software Verification and Extension

As explained in Section 3.3 and Figure 3.10, a verification or testing of the model has to be done to eventually adopt the model and/or existing software. This has been done by using a broad variety of existing software. Table 4.3 gives an overview of all examined software applications. This study focused on modeling data and not on the usability of authoring software. Hence, only Revit was tested as representative of authoring tools. Future research should investigate editing possibilities as well.

4.2.1 Verification of Semantic Data

In the first step, the functionality of visualizing semantic data has been tested. All four IFC entities, i.e., *IfcAnnotation*, *IfcProxy*, *IfSurfaceFeature*, and *IfcVoidingFeature* were tested. The expectation is that the software provides a geometric view, a hierarchical tree view, and a view for the properties. Table 4.4 presents an overview of the test results. Revit, Desite BIM, and Solibri Model Viewer lack the hierarchical view of the model, and hence, the

```
/* Defect Spalling */
#9000= IFCPROXY('0igGRCoTwk6XcjC1hqayEA',#42,'Spalling',
        'Spalling at beam','Defect -Spalling',#242,#9003,
        .NOTDEFINED.,$);
#9001= IFCRELAGGREGATES('3--Gt7_4p0qqpOGbNAh_sw',#42,
        'Damage to product','The related product is damaged',
        #244,(#9000));

        /* Defect Geometry */
#9003= IFCPRODUCTDEFINITIONSHAPE($,$,(#8539));
#8539= IFCSHAPEREPRESENTATION(#105,'Body','SweptSolid',(#8538));
#8538= IFCEXTRUDEDAREASOLID(#8534,#8537,#20,250.);
#8526= IFCCARTESIANPOINTLIST2D(((-125.,-30.),(125.,-30.),(125.,30.),
        (-125.,30.),(-125.,-30.)));
#8533= IFCINDEXEDPOLYCURVE(#8526,$,.F.);
#8534= IFCARBITRARYCLOSEDPROFILEDEF(.AREA.,'Box',#8533);

        /* Texture */
#10000= IFCSTYLEDITEM(#8538,(#10010),$);
#10010= IFCSURFACESTYLE('Damage Texture',.BOTH.,(#10020));
#10020= IFCSURFACESTYLEWITHTEXTURES((#10030));
#10030= IFCIMAGETEXTURE(.T.,.T.,'TEXTURE',#10040,$,'./Texture.JPG');
#10040= IFCCARTESIANTRANSFORMATIONOPERATOR2D(#10050,$,#10060,1.0);
#10050= IFCDIRECTION((1.,0.));
#10060= IFCCARTESIANPOINT((0.0,0.0));
#10061= IFCTEXTURECOORDINATEGENERATOR((#10030),'SPHERE',$);
```

Figure 4.9: Part of an IFC file modeling a proxy for the defect (#9000) and add an image as texture (#10030) to the entire defect geometry (#10000). The texture mapping is defined as spherical mapping (#10061).

Table 4.4: Shows which software has visualized the defect information in a hierarchical or properties view.

Defect types	Autodesk Revit	Apstex IFC Viewer	BIM Vision	Desite BIM	Solibri Model Viewer	usBIM	xBIM Xplorer
Annotation		x	(x)			x	x
Proxy		x	x			x	x
Surface Feature		x	x			x	x
Voiding Feature		x	x			x	x

defects without geometries could not be selected. Furthermore, none of the three includes a hierarchical view of the model. All other software visualizes the test files properly.

Next, the visualization of the relationships was tested. For this purpose, typification, external references, and defect relationships were added. Classification could be visualized via a property view or by using the correct product type. Table 4.5 summarizes the test results. IFC viewers do not access product catalogs, and hence, the type is shown as property in the view. Revit uses its internal type catalog to select the corresponding type of an entity. However, this is only possible if the typification is stored with correct Revit family names. The same problem arises with measurements or properties in Revit. External references should be shown at least in the property view with their URI. The Apstex IFC viewer and xBIM show external references in such a way. None of the other software tools showed the external document references. Last, defect relationships, i.e., aggregation, association or voids element, should be shown in the hierarchical view or as properties. xBIM and Apstex show aggregations in the hierarchical view and associations as properties. BIM Vision was able to show aggregations but not the associations.

Table 4.5: Performance of the software regarding relationships.

Defect Information	Autodesk Revit	Apstex IFC Viewer	BIM Vision	Desite MD	Solibri Model Viewer	usBIM	xBIM Xplorer
Classification	(x)	x				x	x
External References		x					x
Measurements	(x)	x					x
Defect Relationship		x	(x)				x

4.2.2 Verification Texture Implementation

Textures are the second requirement in the data model. To test texturing, an image has been attached to an additional plane, which is at the defect position. Other geometries may be used instead of a plane. As depicted by the last row in Table 4.6, none of the available software was able to properly visualize the texture. Most of them ignored the texture parameter. usBIM only shows the plane where the texture should be depicted.

4.2.3 Verification of Geometry Data

Geometric representations are very common in the AEC sector. However, the software programs support the geometric representations in different quality, which is evidenced in Table 4.6. The visualization of CSG geometries was done properly by all IFC viewers except Desite BIM and the Solibri Model Viewer. None of the viewers that are available by the software vendors offers a selection of representation context. This requirement was

achieved only by Revit. Revit includes 2D plans and 3D views for its building models; however, multiple 3D geometries are not possible in Revit.

The next step tested the visualization of an *IfcVoidingFeature* with an *IfcRelVoidsElement* relationship in accordance with the relationship-based cut-out. The voiding feature is correctly supported by apstex's IFC Viewer and xBIM Xplorer. Other programs do not respect an *IfcVoidingFeature* with an *IfcRelVoidsElement* relationship. Many viewers are able to handle an opening in conjunction with an *IfcRelVoidsElement*. However, defining a defect as an opening is semantically wrong. Figure 4.10 shows the visualization of an *IfcVoidingFeature* with an *IfcRelVoidsElement* relationship in the original xBIM Xplorer. 4.10 a) shows a beam with typical spalling. Figure 4.10 b) depicts a close-up screenshot of the cut-out of the defect in the beam. Lastly, in Figure 4.10 c) one can see the blue highlighted defect geometry of the spalling. A similar result is achieved with the Apstex IFC Viewer.

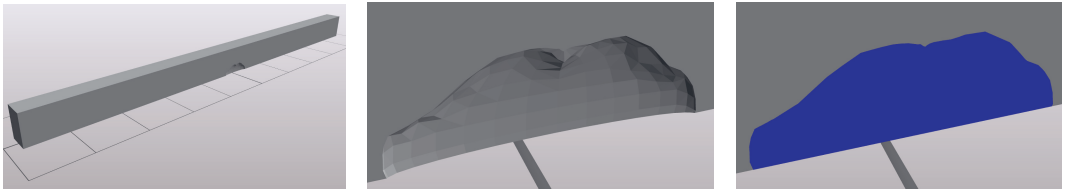
4.2.4 Extension of xBIM Xplorer

Although, IFC is an established standard and implemented in many software applications, several of them show limitations regarding geometry and texture visualization. Furthermore, only Revit supports different views. Hence, manual extensions have to be made to an existing application. Three possibilities exist to extend existing software: (1) developing a plugin or extension, (2) using an Application Programming Interface (API) to add code within the software, or (3) the software itself is open source. In the given software pool, only Revit provides an API. However, Revit shows errors already on the IFC import. Hence, a completely new importer would be necessary that would mean a huge effort. None of the software has a fully developed plugin system. xBIM has a plugin system, however, it is under development. Two of the viewers are (partly) open source: xBIM and apstex. apstex offer only their core IFC parser and model as open source. xBIM offer their complete software including the viewer as open source. So, xBIM was chosen for further extensions. During the development, xBIM has been extended with

1. making links to external references clickable
2. saving and restoring camera positions

Table 4.6: Performance of the software regarding different geometric representations and texture.

Geometry data		Autodesk Revit	Apstex IFC Viewer	BIM Vision	Desite BIM	Solibri Model Viewer	usBIM	xBIM Xplorer [original]	xBIM Xplorer [extended]
CSG + contexts	Context selectable	x							x
	Show different representations								x
	Show defect geometry		x	x			x	x	x
Voiding feature	Subtract geometry		x	x			x	x	x
	Show defect geometry		x	x			x	x	x
Texture									
Visualize Texture									x



a) b) c)

Figure 4.10: The visualization in xBIM Xplorer of a beam with spalling modeled by using a voiding feature a) and a close view at the spalling at the beam b). c) shows the typical spalling geometry. The transparency has been risen to improve the visibility of the cutout.

3. export selected elements to wavefront obj files
4. select a visualization context
5. manual triangular texture mapping
6. and spherical texture mapping

Point 1 was done to getting in touch with the software structure of xBIM and try a first implementation. External references including a path may be included in a document reference. At that time, the path was shown as a normal string. With some minor changes in the *IfcMetaDataControl*, a clickable link was created that automatically opens the given file in the default application, e.g., the browser.

Writing articles, conference papers and documentation required several screenshots of models, defects, and components. However, if the same view shall be used for different models or a another screenshot has to be taken after some model changes, it comes in handy to save and restore camera positions. This leads to the implementation of an export and import of camera view parameters. This function can be found in the top menu under camera/camera position. After saving the properties via save as, a text file is generated as shown in Figure 4.11. Three 3D vectors define the view: the camera position, look and upwards direction. The x, y, and z components of these parameters each are stored in one line. Hence, the first three lines define the x, y, and z component of the position. Followed by the look and upwards direction in the same way.

```
37.7465246782768
-2.11495031082126
2.48377096998818
-2.15067950496985
3.96105889744229
-1.33312049451982
-0.0374374448485882
0.068951195969613
0.996917333733132
```

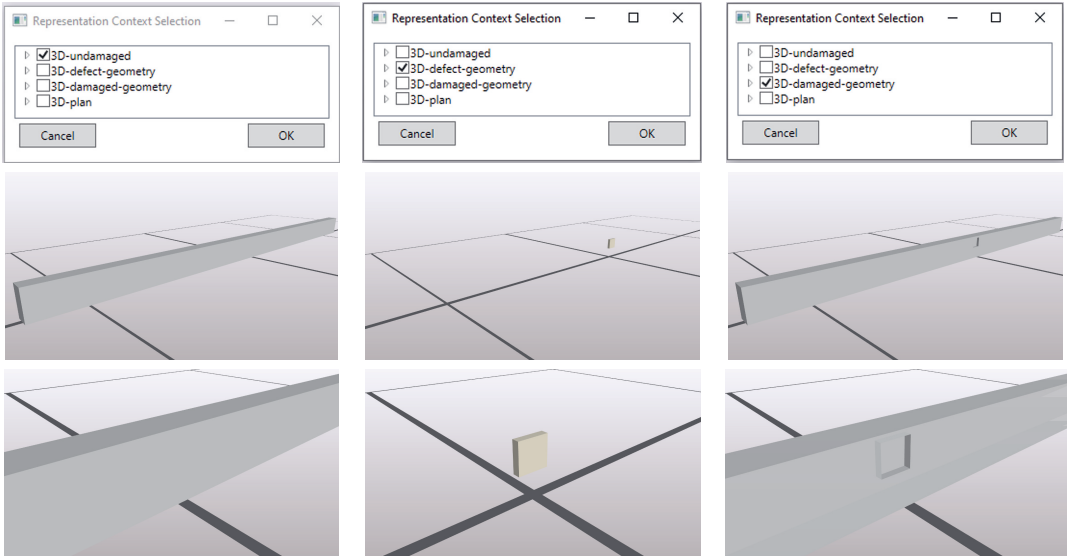
Figure 4.11: Exemplary camera position file containing three lines each for position, look and upwards direction.

Subsequent processes, for instance structural analyses with Ansys, required the geometry only. However, Ansys does not have a built in IFC import that meant another format was necessary. Wavefront files with the ending .obj come in handy in this case. IfcConvert is a usable tool for such tasks because it enables the transformation of IFC files into many other file types [25]. To enable also the selection of a specific representation context, the code has changed in that way that a representation context may be selected via its name [145]. So, the geometries of the selected context(s) are transformed only.

Unfortunately, IfcOpenShell is a command line tool and, hence, a bit cumbersome. After using IfcConvert several times, a graphical user interface found be much more practical. Therefore, the xBIM Xplorer has been extended with a small export function that allows to export selected geometries as wavefront files. Together with the selection of the representation context (4), any geometry may be exported in a more intuitive way.

Figure 4.12 shows the selection of different visualization contexts based on a model with undamaged (a) and damaged component geometries (c) as well as the damage geometry itself (b) in the extended xBIM Xplorer. The top line shows the selected representation context, the line below presents an overview of the model and the bottom line depicts a close-up view of the damaged section. If the defect geometry and the geometry of the damaged component are activated simultaneously, the used defect element, which is a proxy in this case, is shown as filling in the damaged beam. This is disadvantageous because the defect geometry should not be a filling. If the relationship-based cut out is used, i.e., *IfcVoidingFeature* with *IfcRelVoidsElement*, only the damaged component geometry is visible, but not the damage geometry solely. This is comprehensible because openings or voids are normally only visible as subtraction in another element and not as individual element.

Texture related information is represented by green boxes in the diagram. Besides the information, which image shall be used as texture, a texture area and texture mapping is required. Texture maps describe mathematically how to map photos as textures onto a given geometry. Such a texture map may be defined implicitly or explicitly. Multiple texture mapping algorithms exist. Two methods are implemented in the xBIM Xplorer to demonstrate the use of textures for DIM. First, a manual and a spherical texture mapping. Within the IFC file manual texture maps may be provided via the *IfcIndexedTriangleTex-*



a) b) c)

Figure 4.12: Model of a defect by using CSG and different visualization contexts in the 3D view. a) shows the undamaged beam, b) the defect geometry, and c) the damaged beam.

tureMap. It contains a mapping between triangles of the shape and related coordinates in the texture, i.e., a vertex has one or more related u-v-coordinates [13]. So, the creator of the IFC file has the full control about the mapping.

Second, in case of a texture on a sphere, spherical texture mapping may be used. This maps the spherical coordinates of the mesh vertices onto the u-v coordinates of the texture. By identifying the midpoint v_0 of the volume, vectors between the midpoint and all vertices v_n of the 3D model are calculated. Subsequently, the spherical coordinates of these vectors, consisting of r, ϕ, and θ, are calculated. Figure 4.13 shows a sketch of the polar coordinates with ϕ. Analog to ϕ, θ is calculated using the z and y axes. Last, the spherical angles θ and ϕ as radians between 0 and 2π are mapped onto the two texture coordinates u and v between 0 and 1 with

$$u = \frac{\theta}{2\pi}$$

$$v = \frac{\phi}{2\pi}$$

Figure 4.14 shows the Nassi-Shneiderman of the resulting algorithm for spherical mapping. *midPoint* is calculated based on the min and max values of the vertices shown in Figure 4.15. *midPoint* is equally to v_0. Based on that midpoint a vector *direction* as well as the related angles ϕ and θ are calculated. As aforementioned, the algorithms are implemented in C# within xBIM Xplorer. C# allows to parallelize operations by using the *Parallel* class within the System.Threading.Tasks namespace as depicted in Figure 4.16 [146].

In order to provide an extensible object-oriented implementation, a interface based structure has been used for the implementation depicted in Figure 4.17. Generally spoken, each texture mapping algorithm aims to provide a texture map based on the vertices, normals, and triangles. Hence, this can be abstracted into an interface, which is called *ITextureMapping*. Besides the texture map itself, this interface also forces the implementations to provide an information about their algorithm as an enumeration *TextureMapGenerationMethod* via the *GetTexturingMethod* Possible states are defined in the IfcTextureCoordinateGenerator of the IFC 4 standard [13].

The described interface is implemented by two classes: *ManualTriangularTextureMapping* and *SphericalTextureMap*. To create the correct instance, the static class *TextureMappingFactory* takes an *IfcTextureCoordinate* object as argument and returns the related

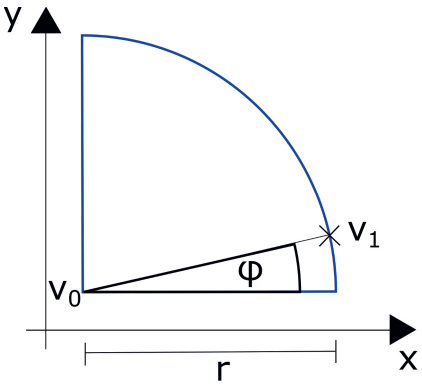


Figure 4.13: Exemplary sketch for calculation of φ for spherical mapping.

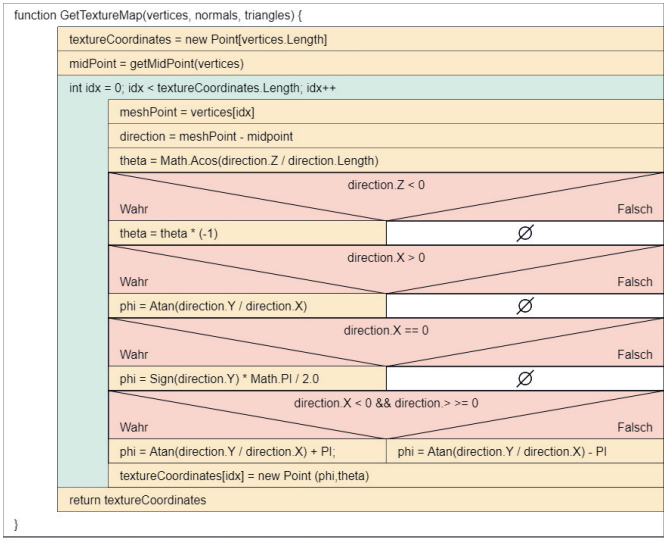


Figure 4.14: Nassi-Shneiderman diagram of the algorithm to create a spherical texture map.

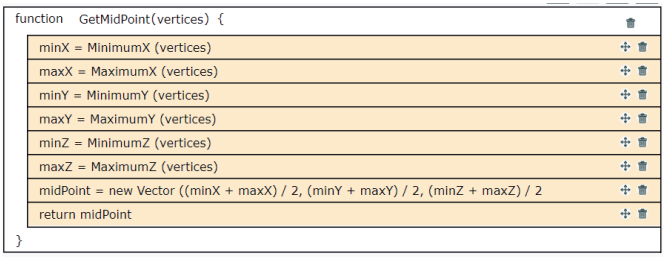


Figure 4.15: Nassi-Shneiderman diagram for the calculation of the midpoint.

implementation of *ITextureMapping*. Hence, if the provided texture coordinate generator has the mode sphere, a *SphericalTextureMap* is returned.

4.2.5 Comparing Test Results to Requirements

Altogether, with the use of IFC and an extension of the xBIM Xplorer, it was possible to address all requirements stated in the Requirement Analysis section. Table 4.7 shows an overview of the requirements and finally used entities of the standardized IFC 4. All implementations could be verified by using an extended version of the xBIM Xplorer.

```
public IEnumerable<Point> GetTextureMap(
IEnumerable<Point3D> vertices,
IEnumerable<Vector3D> normals, IEnumerable<int> triangles)
{
        Point[] textureCoordinates = new Point[vertices.Count()];
        Vector3D midPoint = this.GetMidPoint(vertices);

        Parallel.For(0, textureCoordinates.Length, (verticeIndex) =>
        {
                Point3D meshPoint = vertices.ElementAt(verticeIndex);
                Vector3D direction =
                (Vector3D)(meshPoint - midPoint);
                double theta = Math.Acos(direction.Z
                / direction.Length);
                if (direction.Z < 0)
                {
                        theta *= -1;
                }
                double phi;
                if (direction.X > 0)
                {
                        phi = Math.Atan(direction.Y / direction.X);
                }
                else if (direction.X == 0)
                {
                        phi = Math.Sign(direction.Y)
                        * Math.PI / 2.0;
                }
                else if (direction.X < 0 && direction.Y >= 0)
                {
                        phi = Math.Atan(direction.Y / direction.X)
                        + Math.PI;
                }
                else
                {
                        phi = Math.Atan(direction.Y / direction.X)
                        - Math.PI;
                }
                double u = phi;
                double v = theta;

                textureCoordinates[verticeIndex] = new Point(u, v);
        });
        return textureCoordinates;
}
```

Figure 4.16: Calculation of the spherical texture map in C# using the Parallel library to provide a faster computation.

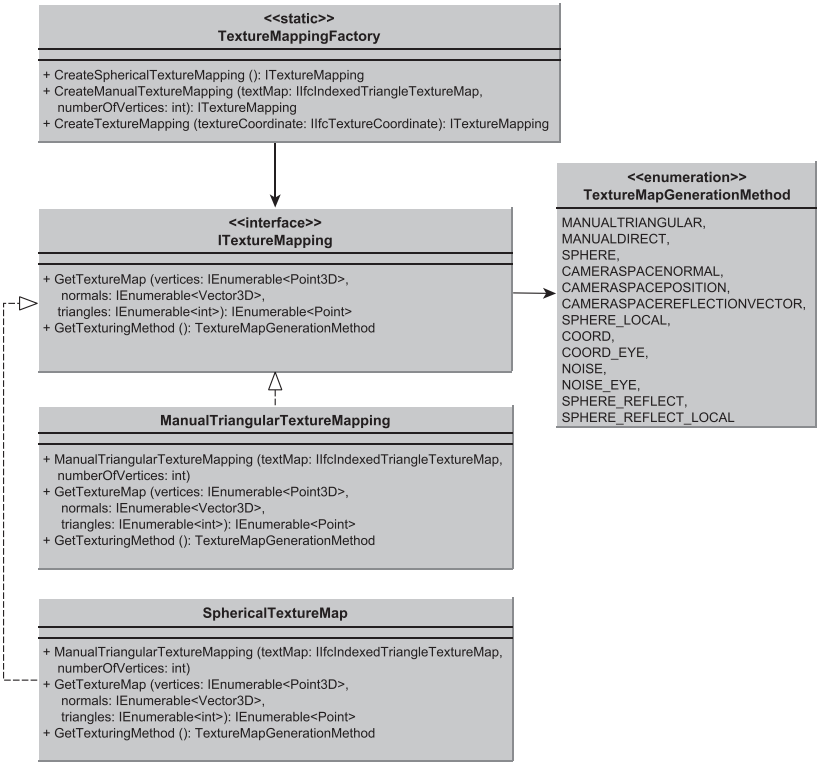


Figure 4.17: UML diagram of the structure for texture maps.

Table 4.7: Summary of test requirements and test results.

Requirement	Successfully tested solutions
Defect entity	IfcAnnotation, IfcProxy, IfcVoidingFeature, IfcSurfaceFeature
Relationship for damaged components	IfcRelAssociatesProduct, IfcRelAggregates, IfcRelVoidsElement
Relationship for defect groups	IfcRelAggregates
Relationship for cause and effect	IfcRelAssociates
Relationship for related documents	IfcRelAssociatesDocument
Classification	IfcTypeObject and IfcRelDefinesByType
Defect properties	IfcPropertySet and IfcProperty
Multiple photos, images, or videos	See relationships for documents
Textures	IfcImageTexture and IfcTextureCoordinate
1D, 2D, and 3D defect geometry	IfcProductDefinitionShape and subclasses
Multiple geometries and selection	IfcGeometricRepresentationContext