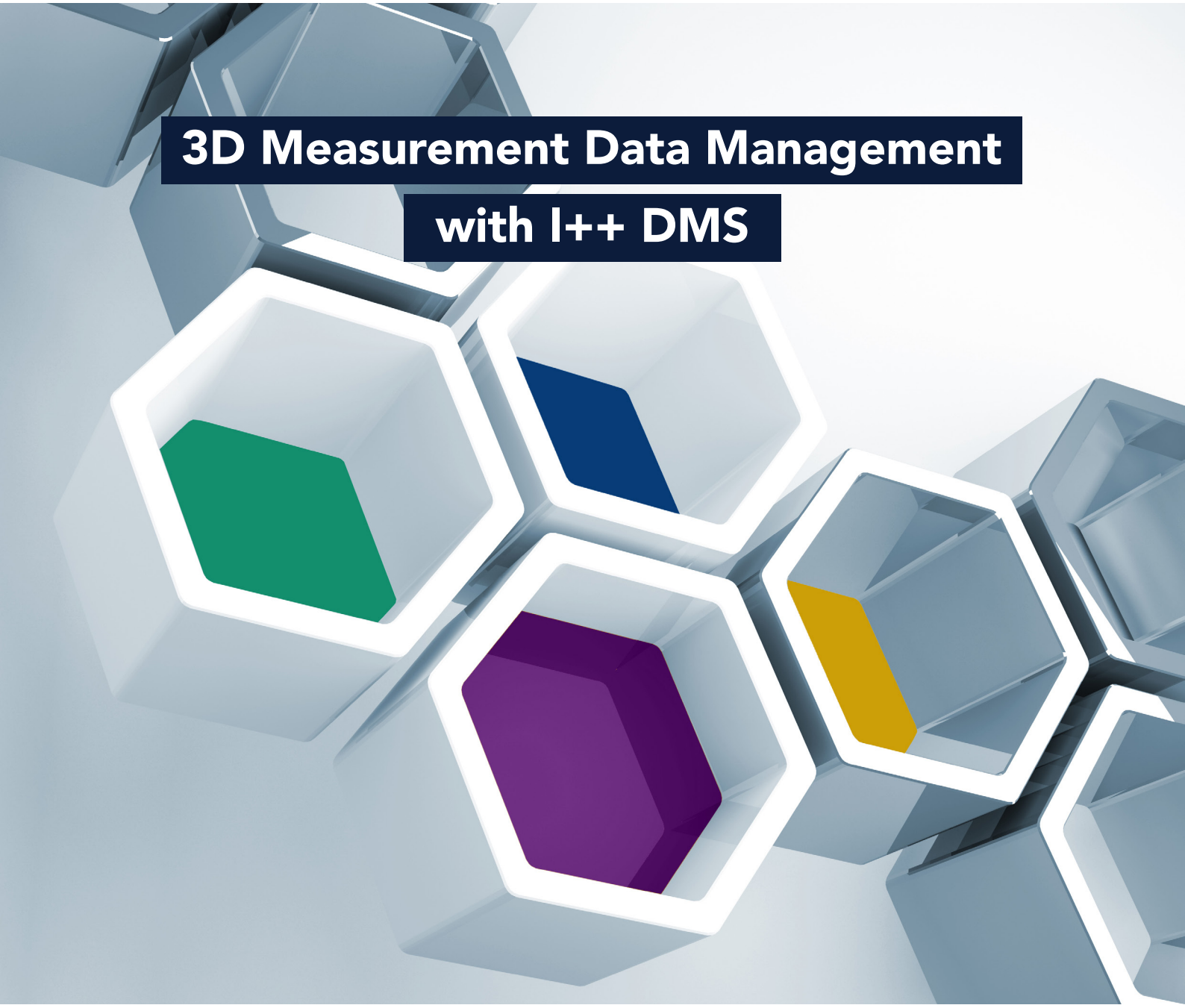
prostep IVIP

VDA | Verband der Automobilindustrie



3D Measurement Data Management with I++ DMS

prostep ivip/VDA Recommendation

3D Measurement Data Management with I++ DMS

3D MDM measurement data management

Disclaimer

VDA/prostep ivip Recommendations (VDA/PSI Recommendations) are recommendations that are available for general use. Anyone using these recommendations is responsible for ensuring that they are used correctly. This VDA/PSI Recommendation gives due consideration to the prevailing state-of-the-art at the time of publication. Anyone using VDA/PSI Recommendations must assume responsibility for his or her actions and acts at her/his own risk. The prostep ivip Association, the VDA and the parties involved in drawing up the VDA/PSI Recommendation assume no liability whatsoever. We request that anyone encountering an error or the possibility of an incorrect interpretation when using the VDA/PSI Recommendation should contact the prostep ivip Association (psiissues@prostep.com) immediately so that any errors can be rectified.

Copyright

Table of Contents

Table of Contents

Figures

List of annexes

Annex A: I++ DMS Documentation v3.0.zip
Annex B: I++ DMS Model v3.0.zip
Annex C: I++ DMS XML Schema v3.0.zip
Annex D: I++ DMS Service Definition v3.0.zip

Abbreviations and acronyms

Abbreviation	Meaning
ANSI	American National Standards Institute
CMM	Coordinate measuring machine
FDC	Functional dimension catalogue
I++	Inspection PlusPlus
I++ DME	Inspection PlusPlus Dimensional Measurement Equipment
I++ DMS	Inspection PlusPlus Data Management Services
ID	Identifier
IP	Inspection plan
IPE	Inspection plan element
ISO	International Organization for Standardization
MBD	Model-based definition
MDM	Measurement data management
NIST	National Institute of Standards and Technology
OEM	Original equipment manufacturer
PDM	Product data management
PLM	Product lifecycle management
PMI	Product and manufacturing information
PS	Product structure
QA	Quality assurance
QC	Quality criteria
QIF	Quality Information Framework
QMS	Quality management system
SOA	Service-oriented architecture
UML	Unified Modeling Language
VDA	Verband der Automobilindustrie (German Association of the Automotive Industry)
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition

1 Preamble

A variety of measurement methods and equipment are used in the manufacturing industry today to ensure a specified level of product quality. These differ in functional properties such as way in which data is collected (contact or non-contact), the way in which measurement data is processed and in the level of integration with manufacturing equipment. In addition to typical properties such as precision and speed, their performance also differs in terms of the degree to which they can be integrated in cross-domain PLM processes. The multitude of devices and processes found in the manufacturing industry always presents a sizable challenge for the harmonization of processes and methods. The desire for a standardized interface for the flexible design of the measurement process, with its numerous participants and objects, is therefore a logical consequence. An object model is required that includes not just information on the product model but also the equipment and tools, as well as the relevant test and tolerance data (part of which is referred to as the product and manufacturing information), and its relation to the 3D geometry.

Given that the quality process is increasingly drawing-free, the digital representation of product data – referred to as the digital master and digital twin – has a key function in this context, too.

Cross-domain data management also gives rise to an additional need for powerful measurement data management. Here, factors such as data-related recording, digital master/twin, control of the measurement process as well as IT systems and interfaces play a role. Companies are hoping that this will bring about an increase in the level of process automation, improvements to the change process, further stabilization in the process, consistent quality statements, enhanced performance in individual process steps, and the early identification of process risks before they become a problem. This challenge was addressed collaboratively and in a timely fashion through the Inspection PlusPlus (I++) initiative.

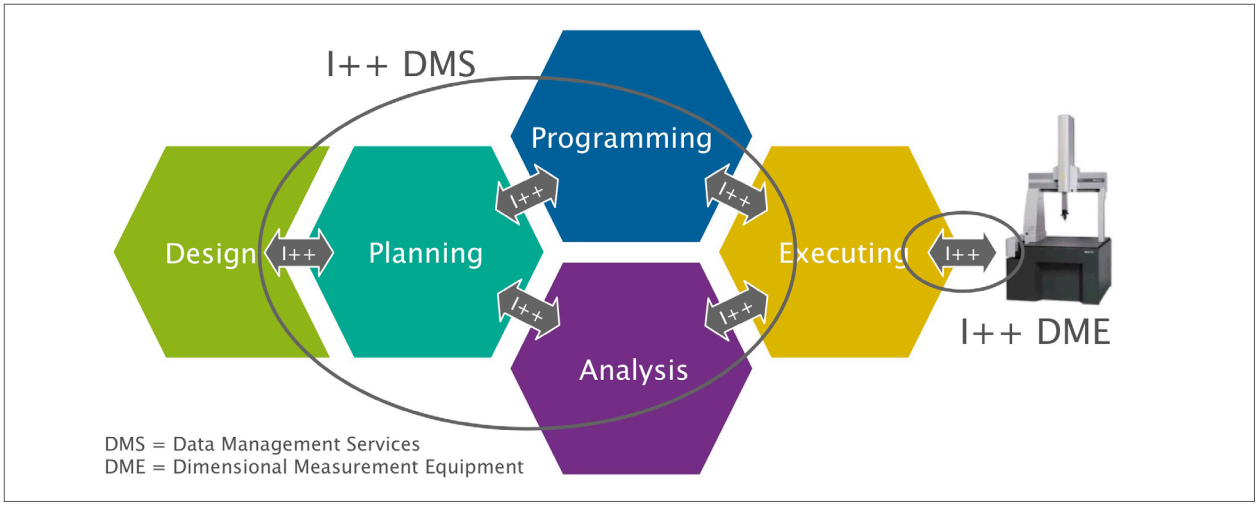


Figure 1: Measurement process covered by I++ DMS and I++ DME

I++ DMS is an interface definition that emerged from the automotive sector's Inspection PlusPlus initiative and which has been further developed by representatives of the automotive and aerospace industries in the 3D Measurement Data Management Workflow Forum (3D MDM WF) and Implementor Forum (3D MDM IF) project groups. The interface definition describes an interface for exchanging information between software applications in the field of dimensional quality assurance. The description covers design, planning, programming, analysis and execution in the quality process.

I++ DMS takes a service-based approach and is primarily comprised of a UML information model and an XML schema that describes the model. This specification and its use in quality management systems are described in this document.

2 Overview

In quality assurance, the systems the produce data and those that consume it do not usually exchange data directly. I++ DMS enables these systems to exchange data via an intermediate layer that provides persistent data storage. I++ DMS takes a service-based approach. It is primarily comprised of a UML information model created in Enterprise Architect and exported as an XMI file (see Annex B), its documentation (see Annex A), an XML schema that describes the model (see Annex C), and services and operations that support the online exchange of data between applications. These are defined in the model on the basis of interfaces, and the WSDL is provided along with this document in Annex D. The interfaces describe ways of accessing the data for the purpose dimensional quality assurance.

Companies that use I++ DMS must implement the I++ DMS interface themselves. In its most basic form, the interface enables data to be exchanged via XML schema-compliant XML files.

At its core, I++ DMS covers not just inspection planning but also areas relating to measurement programming, measurements (i.e. inspections), and analysis. This means that I++ DMS can provide support for dimensional quality management throughout almost the entire process chain. I++ DMS describes product structures that are relevant to manufacturing from a quality assurance perspective, as well as inspection plans, inspection tasks, measurements, measurement programming and analyses. Data sources for this information include applications used in design, manufacturing planning, and tolerance analysis.

This prostep ivip/VDA Recommendation covers I++ DMS Version 3.0. The basis for this was version 2.1 approved by the prostep ivip Association, which emerged from version 2.0 of the specification, which in turn was approved by the I++ OEM working group [Ipp2017]. It describes an interface and data structures for the exchange of information between software applications involved in dimensional quality assurance.

3 Scope

I++ DMS defines various means and technologies for exchanging quality-related data within an integrated process. It is a transport interface for data exchange between production applications, providing a persistence location that allows applications to exchange data. Version 3.0 of I++ DMS, defined here, consists of an interface description and information model.

I++ DMS focuses on the dimensional quality assurance process in automotive and aviation applications. However, it is also intended to be transferable to, and applicable in, other areas, such as mechanical engineering in general.

3.1 Current status

The flow of data in the quality assurance process is commonly siloed inasmuch as it occurs in isolated, standalone solutions. This is especially evident in the planning and analysis process stages, where it is attributable to a lack of suitable standard interfaces. For instance, no standard exists for exchanging the product structures associated with quality data.

The interfaces involved are typically file-based, and not related in any way. By and large, there is no version management in place, either.

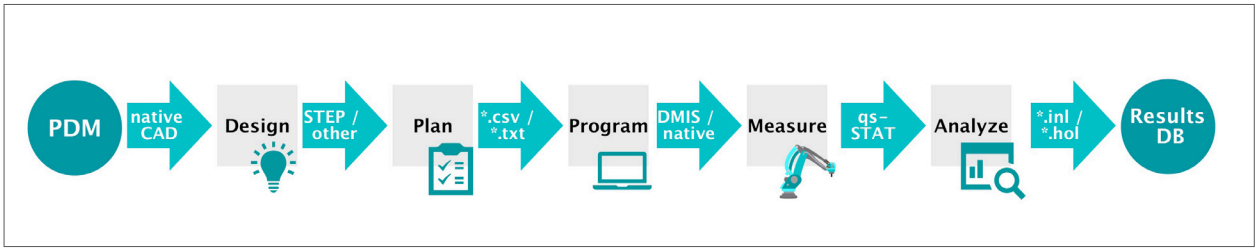


Figure 2: Data flow in today's quality assurance process

3.2 Purpose of I++ DMS

The purpose of I++ DMS is to enable the exchange of quality assurance information along the entire process chain, using a standard interface, in order to allow existing, hybrid systems to provide and receive data in a uniform manner. A jointly defined dimensional quality assurance process and a uniform view of that process allow the following to be accomplished:

Qualitative gain:

- Less conversion-related data loss on interfaces
- Comparability of measurement/inspection results

Qualitative gain:

- Fewer redundant activities
- Freedom to choose between products that offer optimum performance in their respective fields
- Less migration effort
- Lower hardware and software procurement costs (as the result of competition)
- Less programming effort for software vendors

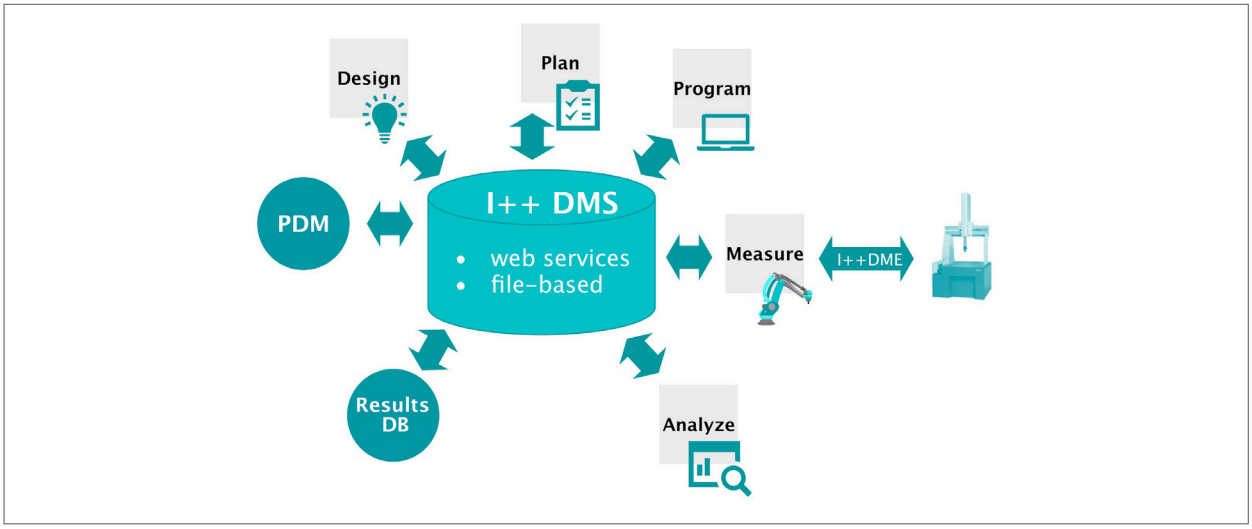


Figure 3: I++ DMS as the foundation of an integrated process

I++ DMS is based on an object-oriented UML information model. The model describes the relationships between all types of quality assurance information and the information that can be exchanged via I++ DMS operations. At the technical level, the exchange of data in I++ DMS is implemented using an XML schema file derived from the UML model. The model also includes several process stage-dependent interfaces (e.g. for product feature planning or programming) and a derived XML-based WSDL file that serves as the foundation for implementing web services.

3.3 Description of the use cases

The reference process created within the framework of the working group and I++ DMS comprises not only inspection planning but also areas relating to measurement programming, measurements and analysis. This means that the I++ DMS services can provide support for dimensional quality management throughout almost the entire process chain.

In the following, terms in *italics* refer to objects from I++ DMS. The defined data structures are quality-assurance product structures that are relevant to manufacturing (*ProductStructure*) as well as inspection plans (*InspectionPlan*), inspection tasks (*InspectionTask*), inspections/measurements (Inspection), measurement programming (*InspectionProgram*) and analyses (*Analysis*).

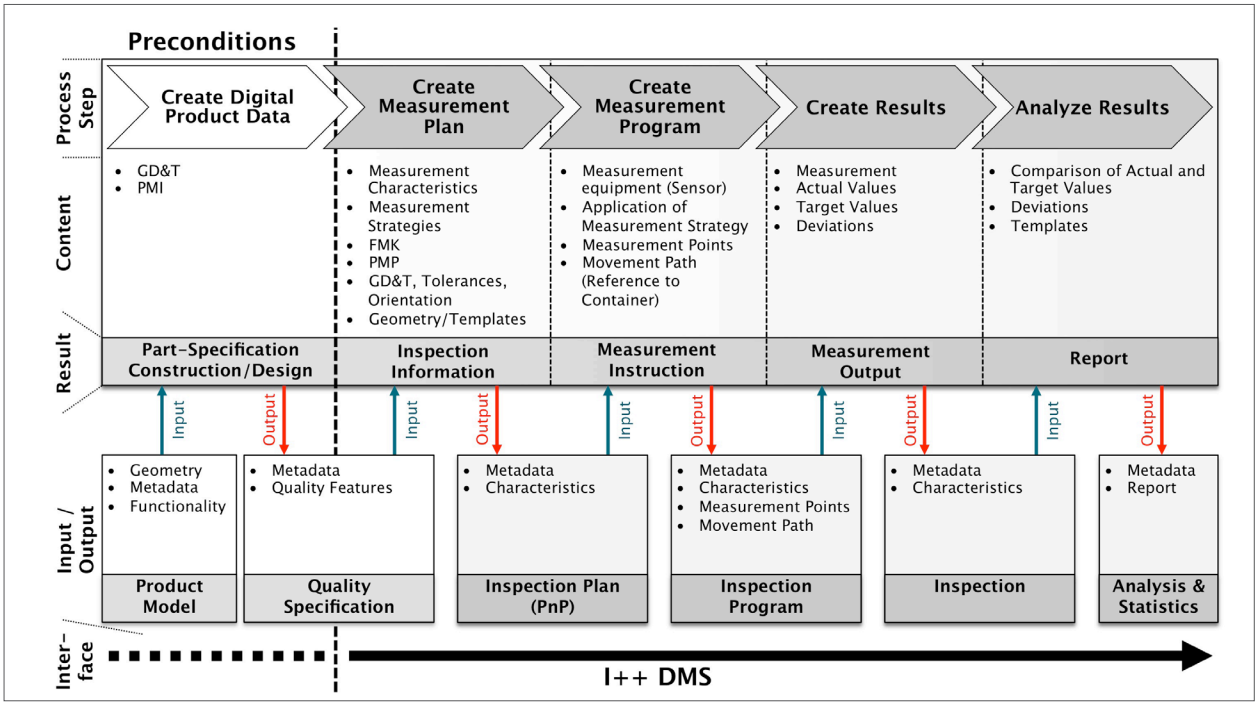


Figure 4: I++ DMS reference process

Engineering, manufacturing planning and tolerance analysis are the primary sources of data for planning the features to be inspected. This data is not exchanged directly in I++DMS, but it can be assigned logically to I++ objects by means of keys/foreign keys. I++ DMS thus provides a powerful basis for implementing software applications to support quality processes within the manufacturing industry.

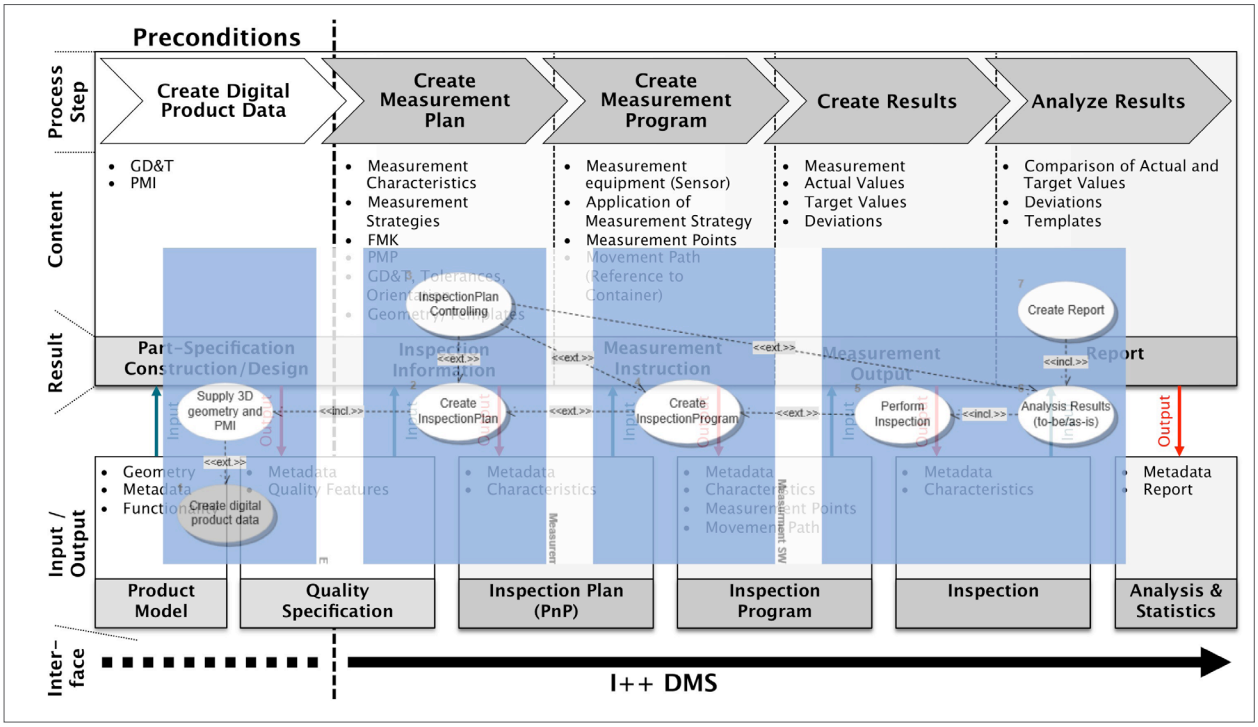


Figure 5: Reference process with use cases

Various use cases were defined within the framework of the project work. In Figure 5, the individual use cases have been superimposed on the reference process. The individual use cases are described in detail in chapter 4.

3.4 Differentiation from other initiatives

3.4.1 Quality Measurement Standards (QMS) Committee

The Quality Measurement Standards (QMS) Committee developed the Quality Information Framework (QIF) with the aim of supporting the exchange of metrological information and data throughout the product engineering process – from the design stage, through manufacturing, to the archival and analysis of product-related data. QIF records the natural structure of the flow of information in relation to component geometry, from the initial geometry description and additional information determined by the designer, to the statistical evaluation of inspection results for a large number of work pieces. The requisite information associated with each aspect of the overall process is recorded in a standard format. This allows greater flexibility when selecting tools and resources in the subsequent process step. [DMSC2015a], [DMSC2015b], [DMSC2015c], [DMSC2015d], [DMSC2015e], [Mor2016]

The QIF information models are contained in files written in the XML Schema Definition Language (XDSL). In QIF Version 3.0, the models consist of six application schema files, plus a library of schema files containing information used by all applications. The *QIF Library* forms the core; attached to it are six information models for the applications *MBD, Plans, Resources, Rules, Results, and Statistics* (see Figure 6). The *Execution* module is a placeholder for a forthcoming DMSC standard that is not part of QIF 3.0.

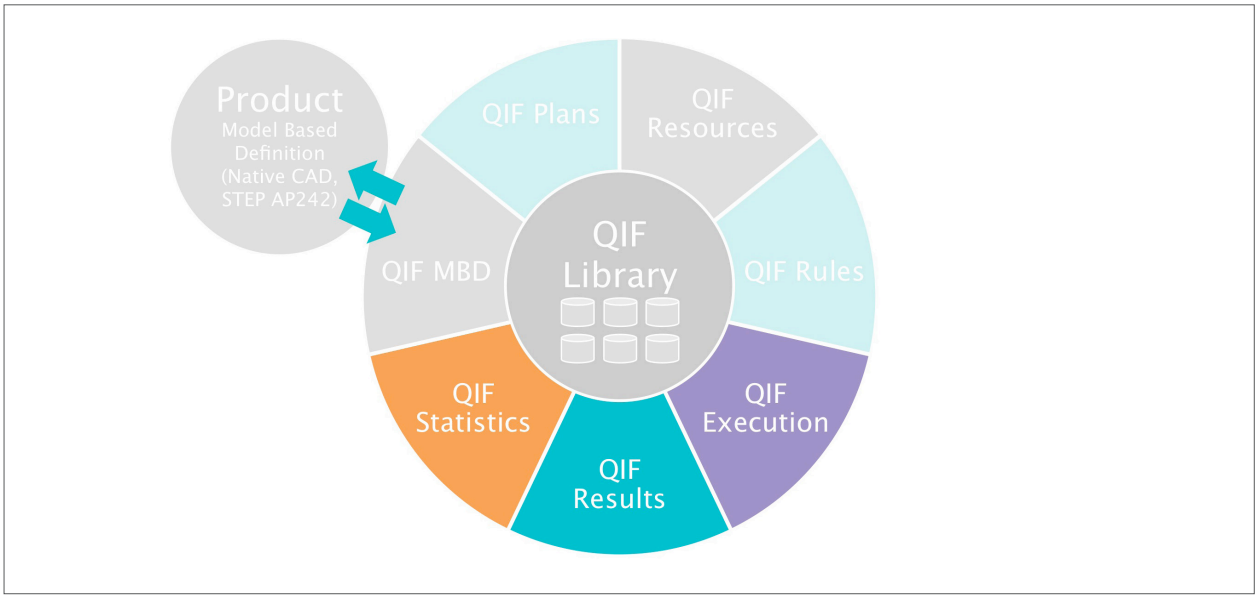


Figure 6: QIF Version 3.0 information architecture
Based on DMSC, Morse et alia (DMSC, 2015; MORSE u. a., 2016)

QIF is pursuing an exclusively file-based approach to data transfer. I++ DMS enables the communication of 3D MDM information via web services. A file-based exchange based on I++ DMS is also supported.

3.4.2 I++ Dimensional Measurement Equipment

I++ DME is a specification for the exchange of information between a coordinate measuring machine (CMM) and application software. A simple, scalable data model for communication with the measuring machine was created with the aid of an object-oriented information model. This standardized interface gives measurement software vendors transparent access to CMM functionalities without needing to know specifically how the machine works. Thus, CMM vendors can safeguard their core expertise yet still give users the open access required in order to control measuring devices. I++ DME also sets clear system boundaries and responsibilities with regard to the precision of coordinate measuring machines and application software. [Gla2010]

The exchange of data between a measuring machine and an application is also governed by another standard which, in part, shares content in common with I++ DME: the Dimensional Measuring Interface Standard (DMIS), which has been formalized as an official ANSI and ISO standard. Only the *DMISequip* part of DMIS Part 2 overlaps with I++ DME. Even though *DMISequip* forms part of an ISO standard, no products are known to have been based on it. By contrast, numerous products have been implemented using I++ DME, so I++ DME can be considered a de facto standard. [IMTI2016]

Because I++ DME only describes communication between measurement software and measuring machines, it is of just minor relevance in terms of the definition of unified data storage across the quality process as a whole. In the context of this document, I++ DME can thus be seen as a peripheral extension to I++ DMS and is therefore not covered in greater detail here.

3.5 Category of information technology

3.5.1 Service-oriented architecture

A service-oriented architecture (SOA) is an abstract software architecture whose primary characteristic is that it offers, searches for, and utilizes services across a network. These services are used almost exclusively by other applications or other services; the service providers' and service users' actual hardware and software configurations are unimportant.

The functionalities provided by individual services can be aligned to the needs of a business process through orchestration – the combining of more than one service. Because the services are loosely coupled, they can be combined in a new sequence if the business process changes.

Data is exchanged according to open standards like HTTPS and XML, so service users only need to understand the interface definition. The underlying programming language and system architecture are irrelevant.

The advantages of service-oriented architectures lie in their use of open standards, loosely coupled services, distributed applications, platform independence, and a process-oriented structure. [Mel08]

3.5.2 Web services

Web services were conceived to enable software applications to communicate with one another. Web services treat software as a set of services, accessible over networks, using web-based standards and protocols.

More specifically, a web service is a software component that can be accessed by another application (such as a client, a server, or another web service). The components that make up a web service are [Gar11], [Mel08]:

- *Web Services Description Language (WSDL)*, which describes web service interfaces
- *Universal Description, Discovery and Integration (UDDI)*, a directory service for available web services
- *Simple Object Access Protocol (SOAP)*, which describes the XML-based message format used in communication and how that format is embedded in a transport protocol

4 Use cases and processes

Figure 7 shows an overview of the simplified UML use case diagram with the key process roles. The use cases serve to illustrate the key relationships between the different process phases and process steps. The individual process phases are explained in more detail and specified using swim lane diagrams in the subsequent sections. The process chart also indicates which data can be exchanged or integrated via I++ DMS.

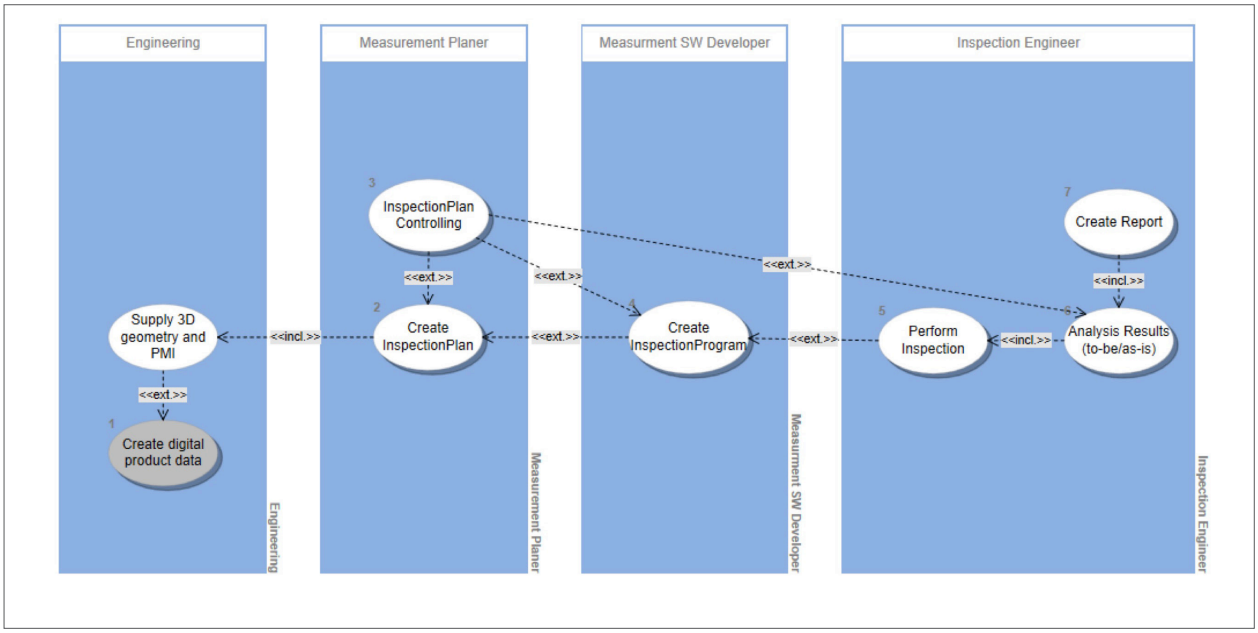


Figure 7: Overview of 3D MDM use case

The process described here begins in the design phase with the creation of the digital product data and subsequent provision of this data. The detailed process is described in section 4.1. This process includes creation of the geometry, PMI and metadata. Quality features are identified based on the geometry and the PMI.

In the next step, an *InspectionPlan* (measurement plan, description of the object can be found in section 5.2.3) is created by the measurement planner. This process step is described in detail in section 4.2. This step includes the transfer and addition of metadata and, if necessary, derivation of the geometric parameters from the Part geometry for the features to be checked. The creation of the *InspectionPlan* also includes the definition of the

IPEs (inspection plan elements) and the selection of the corresponding *MeasurementPrinciple* (measurement principles). If available, a *MeasurementPrincipleCatalog* can be used. One or more (alternative) *Measurement Principle(s)* can be described for a defined feature. The measurement characteristics to be processed are based on the *IPE* and provide input for creating the *InspectionProgram* (measurement programming; description of the object can be found in section 5.2.7).

If *InspectionPlan* control is needed, the inspection requirements are defined in this step. These include, for example, *InspectionSeverity* as well as specifications as to whether the inspection is related to safety, documentation or functions.

The *InspectionProgram* (measurement program) is created on the basis of the *InspectionPlan*. To do this, the QC (*QualityCriteria*) are broken down into measuring elements using measurement strategies (*MeasurementStrategy*), with due consideration given to the properties of a certain measuring technology. This process is described in section 4.4. The actual *Inspection* (measurement) is then carried out by executing the *InspectionProgram* (measurement program). The actual measuring process, i.e. execution of the *Inspection*, is described in the section 4.5. Depending on type and scope, the measurement results are then stored in I++ DMS with reference to the target specifications, or stored elsewhere and linked in I++ DME in an appropriate manner. I++ DME is used to exchange data between the measuring device and the application software.

During *Analysis*, the target and actual values are compared, using an *AnalysisTemplate* if necessary. The results obtained during the analysis phase are documented in a report. Report templates are created for documentation purposes.

The description of the process phases is divided into three sections: "Overview", "Description" and "Benefits". The section "Overview" briefly describes the objective of the process phase. The prerequisites are then explained and the preconditions that have to be fulfilled in order for the process to be implemented are described. The "Description" section provides a description of the use case or process scenario, as appropriate, followed by a description of the final state. This is followed by a simple representation of the process phase as a swim lane diagram. The "Benefits" section describes the main benefits that using this process provides for stakeholders.

4.1 Creating digital product data

4.1.1 Overview of "Create Digital Product Data"

The aim of this process phase is to design a *Part* (component or assembly) and to provide it with functions in accordance with a specification (this goes beyond the scope of this recommendation). Part of the digital description of these functions is a parameterized target specification regarding the required quality of the product/manufacturing. Metadata is also stored, the information content of which is processed in the subsequent process steps. The precondition for this phase is that the requirements for the *Part* (component) have been specified, including the design requirements, the functional requirements and the quality-related requirements.

4.1.2 Description of "Create Digital Product Data"

The geometry is generated in accordance with the *Part* specification, metadata is created and PMI is generated as required. Quality features are identified based on the PMI and the generated geometry. This provides the basis for deriving one or more substitute geometry(s) (for quality testing). The substitute geometries generated are instantiated in I++ DMS.

The quality parameters stored in the model are also transferred to I++ DMS and linked with the appropriate substitute geometries to create *QC (QualityCriteria)*. The metadata is transferred from the *Part* specification and linked in I++ DMS. The result of this process phase is a complete and consistent description in I++ DMS of the (functional) *QC* and metadata to be inspected. The digital product data is made available for use in the next step in a suitable format such as STEP, JT or 3DPDF.

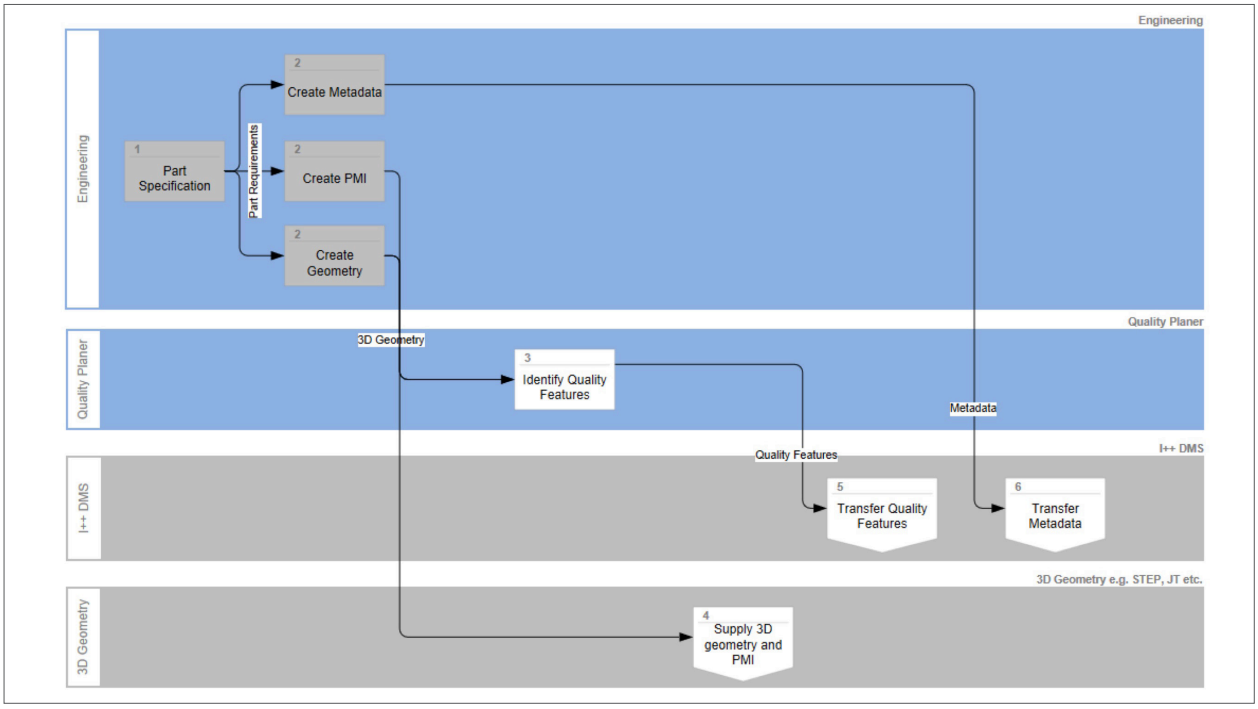


Figure 8: Swim lane diagram for "Create Digital Product Data"

4.1.3 Benefit of "Create Digital Product Data"

The aim and benefit of this process phase is the derivation of a complete description of the information about a *Part* (component or assembly) that is relevant to the quality process, independent of the CAD geometry representation.

4.2 Creating an *InspectionPlan* (measurement plan)

4.2.1 Overview of "Create *InspectionPlan*"

The aim of this process phase is to create, from all or a subset of the *QC (QualityCriteria)* for a Part (component or assembly), a set of *QC* to be inspected which belong together from a planning perspective. The precondition for this phase is a complete and consistent description of the relevant *QC* and metadata of the Part.

4.2.2 Description of "Create *InspectionPlan*"

The results from the previous process phase "Creating digital product data", with instantiation of the substitute geometries, *QC (QualityCriteria)* and metadata serve, as input for this process. The IPE (inspection plan elements) are defined based on the geometry and the QC, and the associated measuring principles are selected. It is then possible to create a subset of the existing *QC* for a *Part* which is based on the planning-related requirements.

This subset is not disjunct, i.e. elements of this *InspectionPlan* can also be part of other *InspectionPlans*. Metadata is transferred and supplemented with the metadata of the higher-level assembly. The result of this process phase is an *InspectionPlan* with the extended metadata. The detailed breakdown of the *InspectionPlan* comprises the *IPEs* to be inspected.

InspectionPlan control might be needed; *InspectionPlan* control is described in the section 4.3.

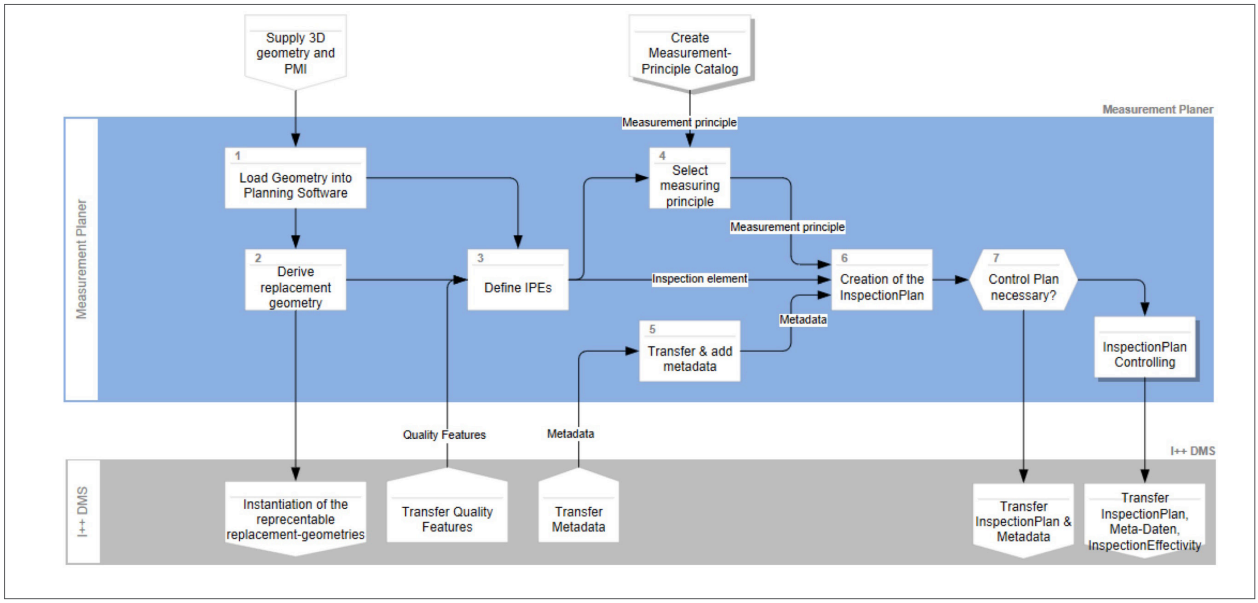


Figure 9: Swim lane diagram for "Create *InspectionPlan*"

4.2.3 Benefit of "Create *InspectionPlan*"

The aim and benefit of this process phase is to generate a consistent aggregate of subsets of the *QC (QualityCriteria)* for a Part (component or assembly) in the form of an *InspectionPlan* (measurement plan) that can be used in subsequent processes.

4.3 Managing an *InspectionPlan*

4.3.1 Overview of "Manage *InspectionPlan*"

The aim of this use case is, if necessary, to manage an *InspectionPlan*. In addition to the measurement characteristics that are to be inspected with the desired level of inspection severity, involves adding information about whether the inspection is related to safety, documentation or functions.

Preconditions for this use case are, first and foremost, an existing *InspectionPlan*, in addition to at least one *MeasurementPrinciple* and one *ReportingStrategy*.

4.3.2 Description of "Manage *InspectionPlan*"

On the basis of an *InspectionPlan* (measurement plan), a subset is created from the set of measurement characteristics in the *InspectionPlan*, taking into account the inspection severity to be achieved.

Each of measurement characteristics in this subset are linked to existing *MeasurementPrinciples* and reporting strategies *(ReportingStrategy)* that are suitable for achieving the required results during the measurement. If alternative measurement principles are assigned, or if there is ambiguity regarding the assignment of the measurement principles, the *InspectionPlan* must be extended to include validity control using *InspectionEffectivity*.

Information about whether the inspection is related to safety, documentation or functions is stored in *InspectionEffectivity* and is taken into account in the subsequent process step with regard to the type and form of evaluation.

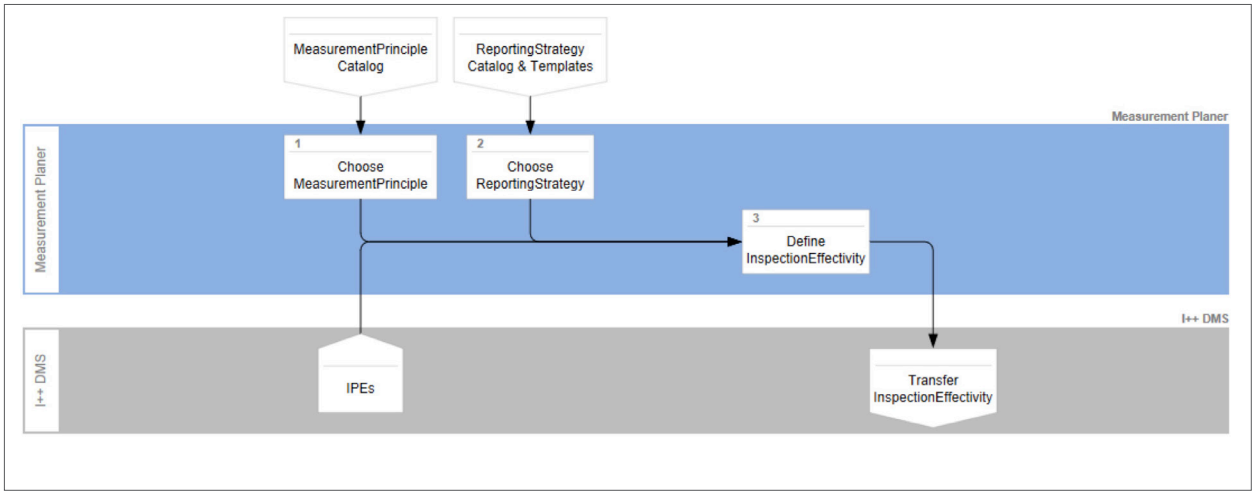


Figure 10: Swim lane diagram for "Manage *InspectionPlan*"

4.3.3 Benefit of "Manage *InspectionPlan*"

The benefit of this use case is the ability to manage an *InspectionPlan* and thus perform an unambiguous measurement due to the fact, for example, that the type of inspection involved (test severity) is specified. It also allows a *MeasurementStrategy,* a *MeasurmentPrinciple* and a *ReportingStrategy,* as well as inspection requirements, to be incorporated.

4.4 Creating an *InspectionProgram* (measurement program)

4.4.1 Overview of "Create *InspectionProgram*"

The aim of this process phase is to derive or create an executable *InspectionProgram* that is mapped to specific measurement principle on the basis of an *InspectionPlan* (measurement plan) and the set or subset of *QC* it contains as well as the metadata. It is also possible to derive multiple *InspectionPrograms* from an *InspectionPlan*, which taken together reference all or a certain subset of the *QC*. An *InspectionProgram* can also address *QC* from several different *InspectionPlans*. When it comes to this n:m relationship between *InspectionPlan* and *InspectionProgram*, it must be ensured that each *QC* is inspected at least once in an *InspectionProgram* , i.e. all *QCs* are taken into account.

A precondition for this phase is that at least one complete and consistent *InspectionPlan* (measurement plan) for a *Part* (component or assembly) including the metadata exists.

4.4.2 Description of "Create *InspectionProgram* (measurement program)"

The *QCs* are broken down into the smallest measuring elements required for the measurement (e.g. measuring points for tactile sensors or scan paths for optical sensors) using measurement strategies (*MeasurementStrategy*) and with due consideration of the properties of a certain *MeasurementPrinciple*.

The elements to be measured *(IPEs)* are combined in accordance with the requirements relating to speed, safety and other criteria to create an executable path and the *InspectionProgram*.

Quality parameters are taken from the *InspectionPlan* and stored in the *InspectionProgram* at measuring points at which quality statements are needed during the measurement.

Metadata is transferred and supplemented with metadata from a higher-level assembly, e.g. when measuring an A or B pillar in the context of measuring the door opening, metadata from the assembly level above the A or B pillar is added.

The result of this process phase is an executable *InspectionProgram* that has been mapped to a specific measuring technology with the corresponding metadata.

The program is initially machine neutral and can be sent to the machine via I++ DME (3.4.2). I++ DME provides a common machine interface which, when used with I++ DME-compatible CMMs, ensures that all CMMs behave in the same way with any I++ DME compatible software.

In the case of a manual inspection, the *InspectionProgram* usually takes the form of a process instruction.

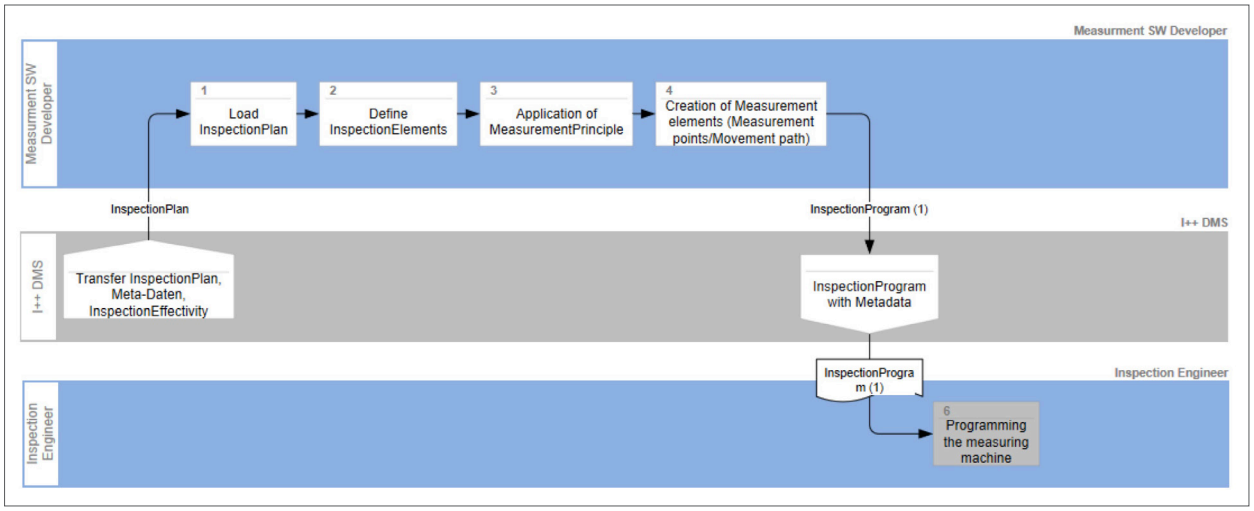


Figure 11: Swim lane diagram for "Create *InspectionProgram*"

4.4.3 Benefit of "Create *InspectionProgram* (measurement program)"

The aim and benefit of this process phase is to create an executable *InspectionProgram* (measuring program) utilizing the properties of a specific measuring technology for use in further process steps.

4.5 Performing an inspection

4.5.1 Overview "Perform Inspection"

The aim of this use case is to create an executable *InspectionProgram* (measurement program) that has been mapped to a specific measuring technology as well as the metadata and measurement results *(MeasurementResult)*.

The precondition for this use case is that an executable *InspectionProgram* (measuring program) that has been mapped to a specific measuring technology with metadata is available from previous process steps.

4.5.2 Description of "Perform Inspection"

An executable *InspectionProgram* (measuring program) is executed using a measuring device with the corresponding technology. Metadata from the *InspectionProgram* (measuring program) is taken into account.

In the simplest case, only actual values are transferred. If necessary, measurement results are correlated with the target specifications and evaluated using existing reference values where needed at this stage. In this case, the program flow will be influenced by the results of this evaluation.

If an optical measurement is involved, e.g. scanning an area, the person using the measurement software will use the sensor path defined in the *InspectionProgram*. The result of the measurement is that the user is provided with the scan data for the *InspectedPart* in the form of point clouds, gray-value images, feature lines or voxel models.

The transfer of the scan data takes place as a separate representation (point clouds, gray-value images or voxel models) externally but is referenced (link) by the I++ DMS representation. Additional metadata is required to transfer the scan / image data from the measuring system to the downstream process steps. This metadata describes the content of the data so that the consuming tool can interpret the data and should contain, at the very least, information about the *InspectionPlan* and the data type.

If quality statements are needed in the measurement, they are generated by evaluating the *QC*.

If the information required is available in the measurement, the measurement results are then stored, depending on type and scope, in I++ DMS with reference to the target specifications, or elsewhere and linked in I++ DME in an appropriate manner.

This use case makes the *MeasurementResults* available in I++ DMS with reference to the target specifications, or elsewhere linked and in an appropriate manner.

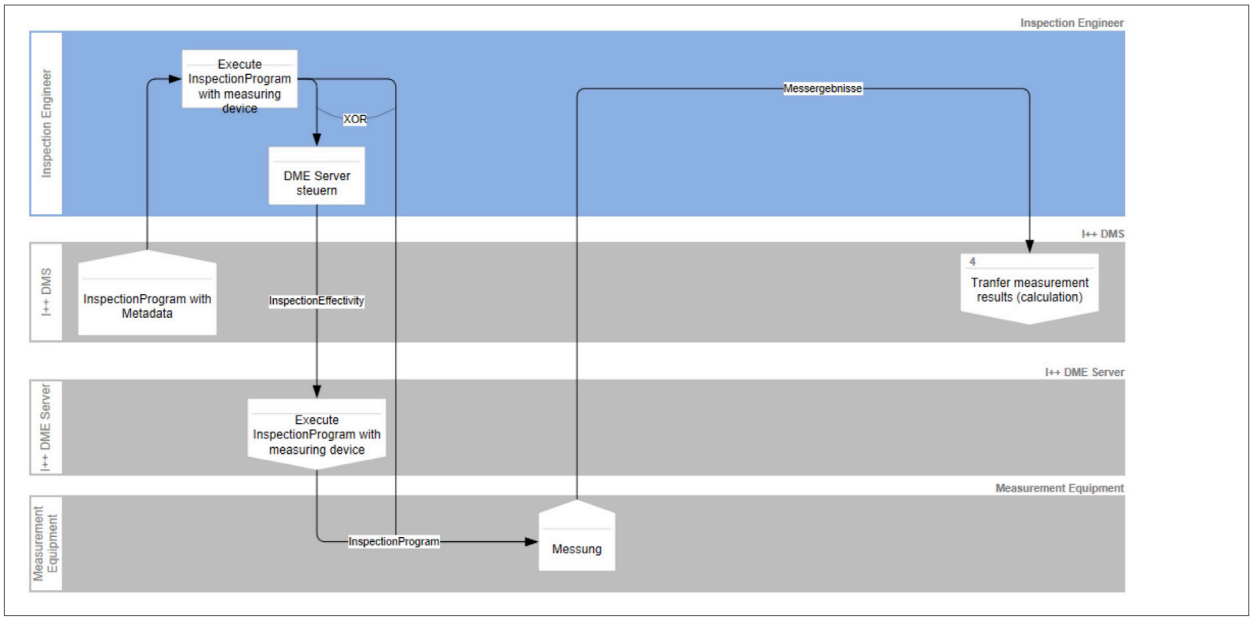


Figure 12: Swim lane diagram for "Perform Inspection"

4.5.3 Benefit of "Perform Inspection"

The benefit of this use case is that the measurement results are available in I++ DMS, or linked in appropriate manner, and are available for evaluation by down-stream process steps.

4.6 Result analysis

4.6.1 Overview of "Result Analysis"

The aim of this use case is to compare the recorded measurement data for the *InspectedIPE* (inspected elements) with the target values for the *IPEs* (inspection plan elements). The preconditions for this use case are the availability of the *MeasurementResults* as well as the *InspectionPlan* with the metadata.

4.6.2 Description of "Result Analysis"

The *MeasurementResults* for each *InspectedIPE* (inspected element) are evaluated and a target/actual comparison is performed based on an *InspectionPlan* and the metadata it contains. The data is then processed accordingly, depending on the type of measurement (discrete, continuous or attributive).

If an optical (continuous) measurement is involved, an area-based evaluation, for example, could follow. This first of all involves aligning the recorded scan data and comparing it with the target geometry (e.g. CAD geometry with tolerances). As a result, the user is provided with a false-color representation or, if appropriate, a textured 3D surface model.

A feature extraction based on the scan data can also be performed. This might involve consolidating the measurement data. The type and scope of any consolidation are the subject of the measuring principle. The actual elements are extracted from the aligned scan data on the basis of the target feature. This means that the aligned scan data, the target element (IPE), the *InspectionPlan* and the *MeasurementPrinciple* need to be available. The result is the *InspectedIPE* (actual element). This makes it possible to compare an actual element with the target element.

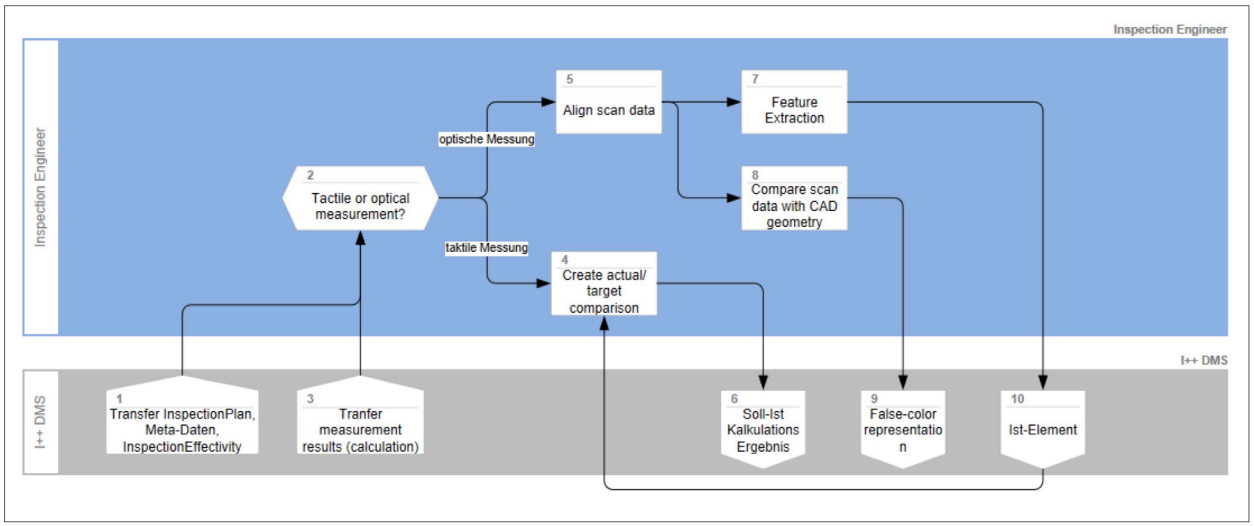


Figure 13: Swim lane diagram for "Result Analysis "

4.6.3 Benefit of "Result Analysis"

The benefit of this use case is that the target/actual comparison (calculation) or, in the case of an optical measurement, a false color representation is available for the reports.

4.7 Creating a report

4.7.1 Overview of "Create Report"

The aim of this use case is to define representations for measurement results suitable for the desired statement, divide the measurement characteristics to be evaluated into groups (reports) that are appropriate from an evaluation point of view, and evaluate the measurement results determined, or to be determined, using an *InspectionProgram* (measurement program) in accordance with these representations on the basis of an *InspectionPlan* (measurement plan).

The preconditions for this use case are the availability of an *InspectionPlan* (measurement plan) with metadata as well as *MeasurementResults* stored in I++ DMS with reference to the target specifications, or elsewhere and linked in an appropriate manner.

4.7.2 Description of "Create Report"

An *InspectionPlan* (measurement plan) and the metadata it contains is used to divide the measurement characteristics to be evaluated into groups (reports) in a manner that is appropriate from an evaluation point of view.

Representations of the measurement results appropriate to the desired or required statement are defined and incorporated in a layout. Metadata is processed and sup-plemented with metadata from a higher aggregation level.

The *MeasurementResults* resulting from the execution of an *InspectionProgram* are evaluated in terms of the *QC* (quality criteria) in accordance with the *AnalysisTemplates* (layouts) and consolidated to create a measurement report.

The result of this use case is a measurement report layout and a measurement report.

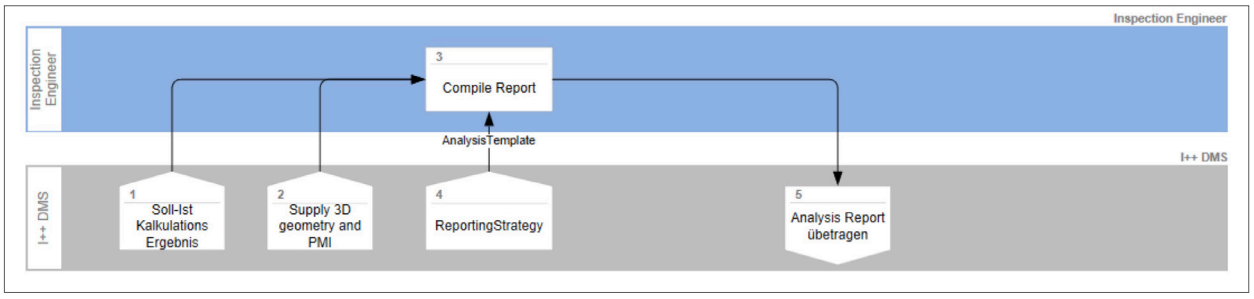


Figure 14: Swim lane diagram for "Create Report"

4.7.3 Benefit of "Create Report"

The benefit of this use case is that the selection of suitable representations results in the creation of meaningful measurement report layouts and measurement reports.

4.8 Creating a catalog of measurement principles

4.8.1 Overview of "Create Measurement Principle Catalog"

The aim of this use case is to create a *MeasurementPrincipleCatalog* (a catalog of measurement principles), expand a company's existing catalog or create a catalog containing *MeasurementPrinciples* for defined features. Another aim is to make the catalog or the referenceable measurement principles known.

Preconditions for this use case are that the abstraction of the measurement method (optical/tactile), the abstraction of the target element and the abstraction of the actual element are known.

4.8.2 Description of "Create Measurement Principle"

A *MeasurementPrinciple* describes how the abstracted actual element *(InspectedIPE)* is derived from the abstracted target element *(IPE)* using a (generic) measurement method. The result of the use case is an applicable measuring principle.

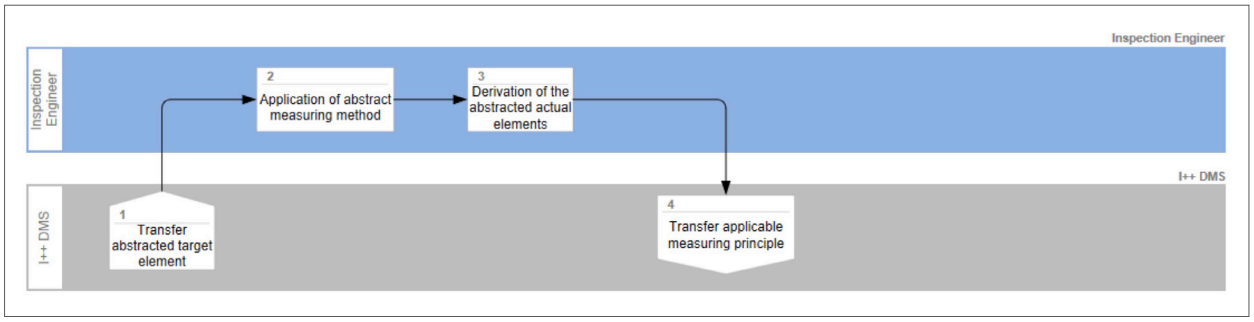


Figure 15: Swim lane diagram for "Create Measurement Principle"

4.8.3 Benefit of "Create Measurement Principle"

The benefit of this use case is that it defines an applicable measuring principle for downstream processes.

5 I++ DMS data model

This section provides a functional overview of the interface definition. Detailed descriptions of the classes, including their associations with one another, are provided in Annex A.

5.1 Creation of the I++ DMS XML schema

The data structure defined for the UML data model was designed to ensure that the associations between the classes are easy to read. To avoid redundancy in XML structures during the exchange of data, the following transformation needs to be performed when deriving the XML schema (XSD) from the UML information model:

- Associations marked with the "contains" stereotype are implemented in the I++ DMS XML schema as embeddings. The components (the parts) specified in a composite aggregation become variables in the parent class.
- Associations without the "contains" stereotype, however, are implemented as references (type ID4Referencing) from the parent class (the whole) to the components (the parts).

5.2 Brief overview

Version 3.0 of the I++ DMS information model contains a number of transport objects that reference actual user data. These are:

- *ProductStructures:*
  Used to read and write QA product structures.
- *AllPScontexts:*
  Used to enable users to navigate through QA product structures before a product structure is loaded.
- *InspectionTaskList:*
  Used to pass information related to inspection tasks.
- *InspectionPlanList:*
  Used to search for inspection plans.
- *InspectionPlan:*
  Used to transfer inspection plans.
- *InspectionProgramList:*
  Used to transfer meta information on inspection programs.
- *InspectionList:*
  Used to transfer inspections (measurements).
- *AnalysisTemplateList:*
  Used to search for analysis templates.

5.2.1 Unique IDs

To make the process reproducible, each data object written by a client must be written using the same unique ID so as to ensure that the data transferred is clearly assigned.

5.2.2 QA product structures

In essence, the *ProductStructure* and its parts transported in I++ DMS are a medium for organizing inspection plans. To meet this purpose, the product structure in I++ DMS is regarded, and processed, as a complete abstraction of the product structure received from engineers or production planners. To avoid ambiguity, the prefix QA (for quality assurance) is applied to a number of attribute names. In extreme instances, this can result in a QA product structure differing from a PDM product structure. For instance, a QA product structure can minimize redundancy so as to reflect an inspection plan's relations and area of application as clearly as possible and making it simple to process, whereas a PDM product structure generally represents a single, production-ready structure. Nonetheless, there needs to be a way to assign an individual Part instance to a specific PDM component instance or to a group of PDM component instances in order to determine how the component is represented geometrically. One of the things needed to accomplish this is the attribute *Part.PDMversion*.

If a *QA* part is suitable for more than one version of a PDM part, this is expressed using the *Comment* attribute, which is attached to the *Part* object. *Part.PDMversion* is the primary version associated with the part number *Part.Name*.

In terms of modeling QS product structures in I++ DMS, this means that a product structure representation includes parts of the PDM view as well as QA-specific parts: In I++ DMS, a part (Part) is described in the PDM view as a combination of a part number (*Part.Name*), PDM variant (*Part.PDMvariant*, a list) and *Part.PDMversion*. References can be made to the PDM data via this trio of information in the QA data.

Care must be taken to ensure that the logical part designation's attributes are populated correctly when the data is generated – for instance, when new geometry versions are edited. To enable inspection planners to take a logically consistent work approach, the logical part designation should be consistent, even after versioning or variant creation. However, it may not always be possible to ensure that part numbers are absolutely consistent. Provisional part numbers are an example of where deviations can occur.

5.2.2.1 Product structure version management

For quality assurance purposes, it is both practical and important to be able to manage versions of product structures and inspection plans, and to be able to edit these versions separately. In any company, there can potentially be multiple versions of any given assembly consisting of more than one component.

5.2.2.1.1 Fundamental solution

In the following, references to product structure version management mean version management for an entire structure, consisting both of a Part instance and of components assigned to that part through *PositionedPart* instances. The definition is recursive and, as such, incorporates all subordinate instances up to and including individual parts (*Part_Singles*). The structure, in its entirety, is described as a versionmanaged product structure, known as a "product structure package" or PS package for short. By contrast, the term "part" refers simply to an individual Part instance.

Version information is not applied to individual Part instances. Instead, in I++ DMS, version information is applied to the product structure packages described above. Wherever reference is made to a product structure as a whole, its associated version identifier is included. (In I++ DMS, a version identifier consists of a single string conforming to that XML data type.) This occurs as follows:

- When product structures are transferred.
- When product structures are referenced from the *InspectionPlan*.

Exactly how product structure version management is implemented depends on the given infrastructure. However, any I++ DMS-compatible software should meet the following conditions, unless noted otherwise here in this document: No client shall be allowed to actively influence the versioning, because this would equate to prescribing a particular versioning solution, and the manner in which a product structure's version identifier is determined is a matter of how the service is implemented. Instead, a client is put in a position to read, delete or write exactly one product version via the I++ DMS interface.

5.2.2.1.2 Version identifier processing by standard I++ DMS clients

The descriptions that follow apply to clients that do not rely on Extensions and can work solely on the basis of standard I++ DMS content. As long as the interface speci-fications are met, enterprisespecific clients can use Extension mechanisms to provide additional functionality.

The standard client only processes the version identifier in the following way:

- It is displayed in connection with the relevant PS package.
- In selection processes, the version information is saved along with the selected package and used for operation calls, e.g. for loading processes.

In the following, the term QAversion describes the version identifier of a PS package as seen from a QA perspective, not an actual physical attribute.

5.2.2.1.3 Maturity levels and version management

The *Part.QAmaturity* attribute, already introduced, is another non-identifying attribute: *Part.QAmaturity* maps the lifecycle of the QA view. *QAmaturity* is valid for a Part instance or *PositionedPart* instance. A Part instance whose *QAmaturity* attribute is set to released must not be modified; this is a mandatory requirement. It must be version-managed instead. The *QAmaturity* attribute of the new version is initially set to draft.

A client can identify from the *Part.QAmaturity* attribute which Part instances it may modify and which instances need to be replaced with a copy for versioning purposes. However, the client does not assign a version identifier, it saves the updated product structure via I++ DMS. The relevant version information is added to the data in the service implemented.

The contents of Maturity defined for *Part.QAmaturity* can therefore mean that, when a client modifies a released instance, a new version must also be created in the infrastructure.

Important: Making changes to an assembly also means changes to the attribute values or the components (i.e. the *PositionedParts*).

Client: Dialog sequence WITH version labels for the read operation:

1. All instances of *PScontext* are read: no change, no version info.
2. All product structures for a given *PScontext* are returned. The product structures are packed into intermediate packages that include the relevant version identifier.
3. A Part instance WITH version information from the previous step (i.e. from a PS package) is selected. The version information is taken from the transport object of the intermediate package into which the Part instance was packed.
4. The Part instance selected in the prior step is loaded along with all the components from the database (PS package). The PS package's version information is specified during this process.

The procedure is the same in principle when writing and deleting product structures. The version information is specified here as an operation parameter.

When browsing, reading or writing inspection plans, the relevant version identifier is included wherever PS packages are involved.

5.2.2.2 Referencing parts in *ProductStructureContexts* and other parts

Parts without a parent assembly are assigned directly to *ProductStructureContexts* (PScontexts for short). A Part instance may be used in one or more *Part_Assembly* instances or *PScontexts*. In the case of the latter, parts can be used in more than one project. This is relevant, for example, in contexts involving identical parts or standard purchased part parts. This means that a Part instance can be referenced not just by more than one *PScontext*, but also by assemblies outside its own *PScontext*.

Note: A *Part* instance with more than one reference (i.e. point of use) in the above sense only ever introduces its characteristics into these references in a single, consistent form. In other words, all attributes – including versions, degrees of maturity, and the structure from child components – exist just once. Any changes made affect all the points of use, so inspection plans also apply implicitly to all points of use.

Nonetheless, it is possible to prevent use of the option of embedding multiple structures in *PScontexts* by choosing appropriate I++ DMS clients. Steps can be taken to ensure compliance with these guidelines by including suitable inspection algorithms in implementations of I++ DMS services. If multiple referencing is used, all the references (points of use) are always named in the *PScontextID* attribute of the *ProductStructure* instance when the product structure is read.

5.2.3 *InspectionPlan*

Serves as a transport object for transferring an inspection plan. Contains the meta data that references the QA product structure; transports all direct and indirect content. An inspection plan is created by inspection planners and read by measurement programmers; it serves to guide the creation of measurement programs.

5.2.4 Inspection Plan Elements (IPE)

There follows an overview of the predefined inspection plan elements to illustrate their importance. A detailed description is provided in Annex A. It focuses in particular on the creation of the relevant data objects. The figure immediately below explains the symbols used in the subsequent figures.

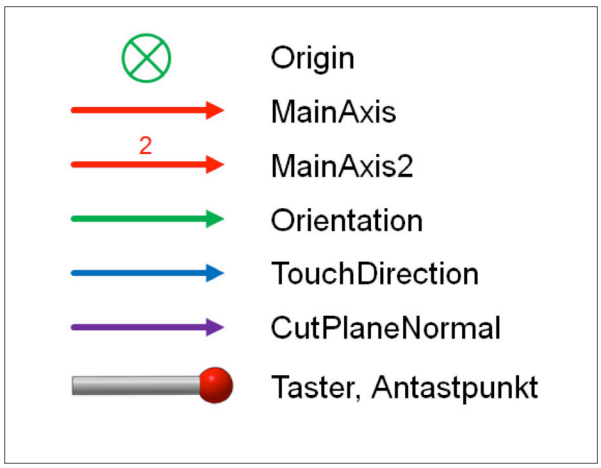


Figure 16: Caption for all IPE figures

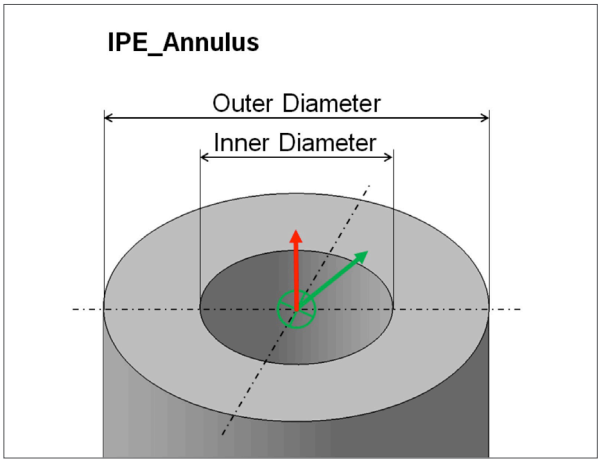


Figure Figure 17: IPE_Annulus

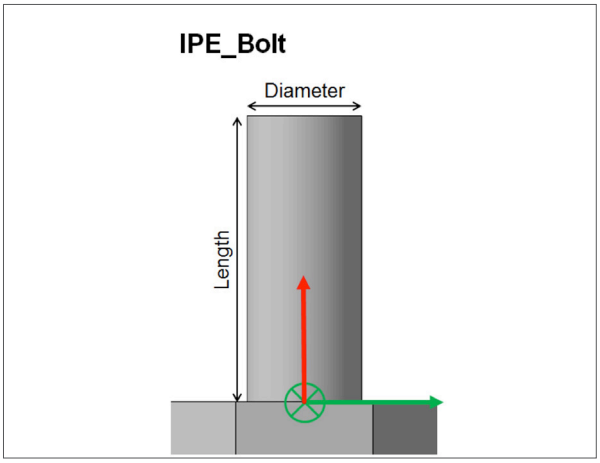


Figure 18: IPE_Bolt

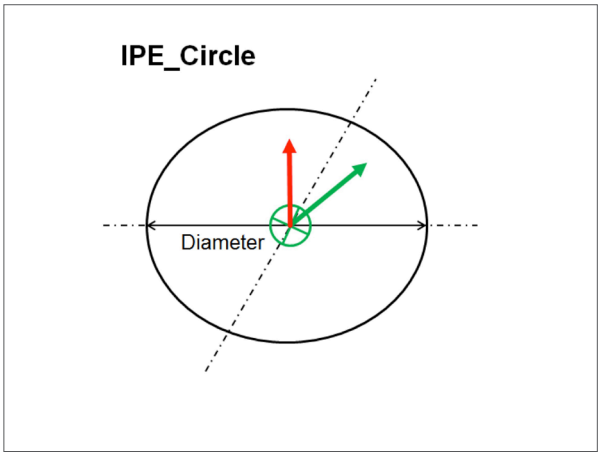


Figure 19: IPE_Circle

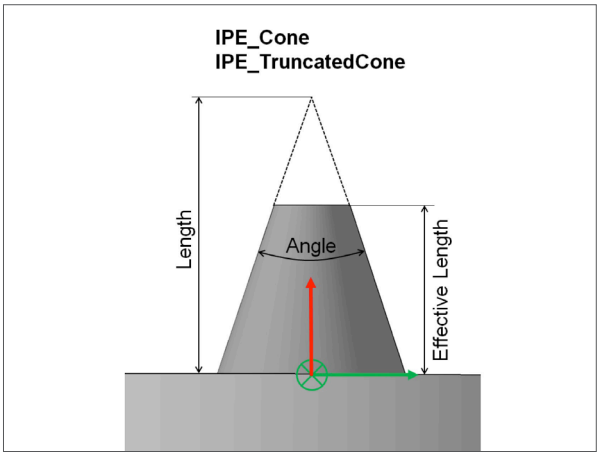


Figure 20: IPE_Cone and IPE_TruncatedCone

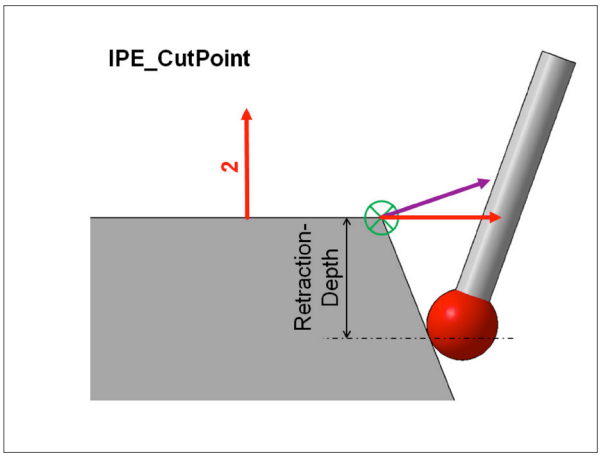


Figure 21: IPE_CutPoint

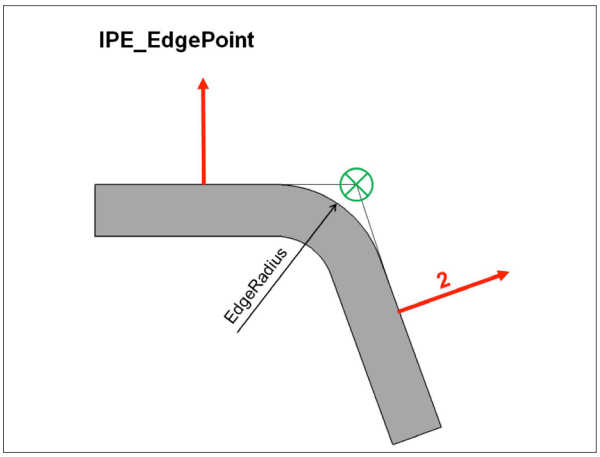


Figure 22: IPE_EdgePoint

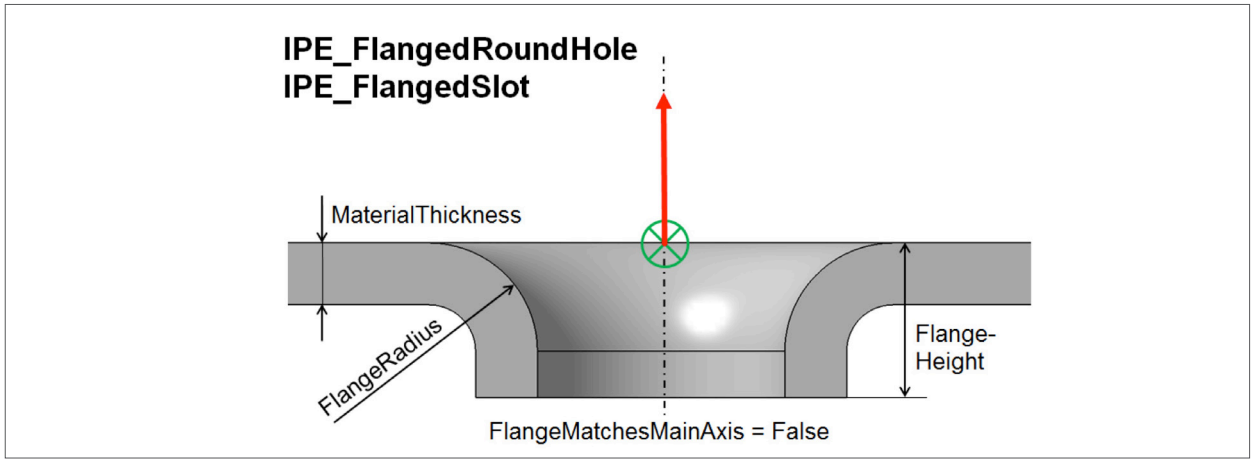


Figure 23:IPE_FlangedRoundHole and IPE_FlangedSlot

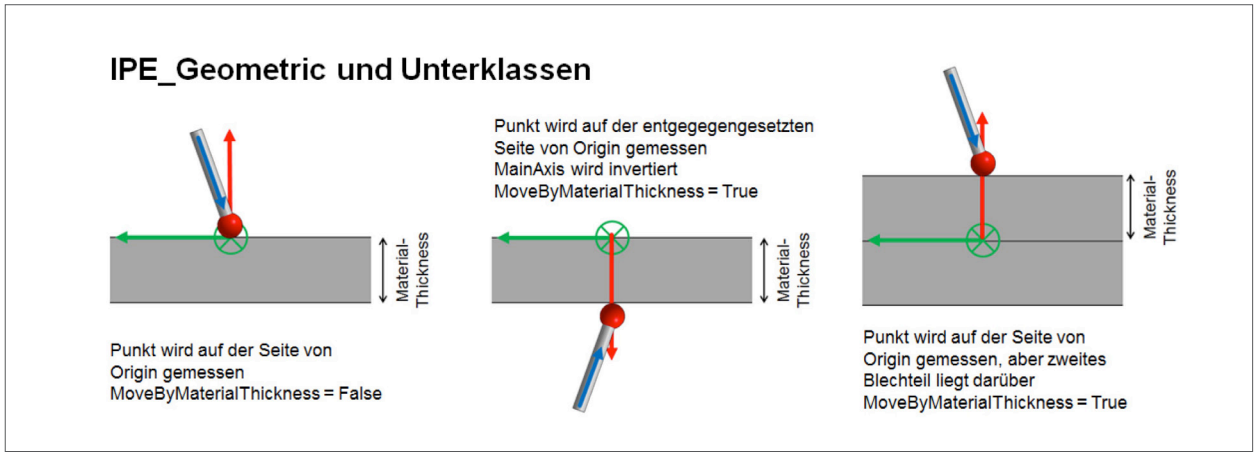


Figure 24: IPE_Geometric

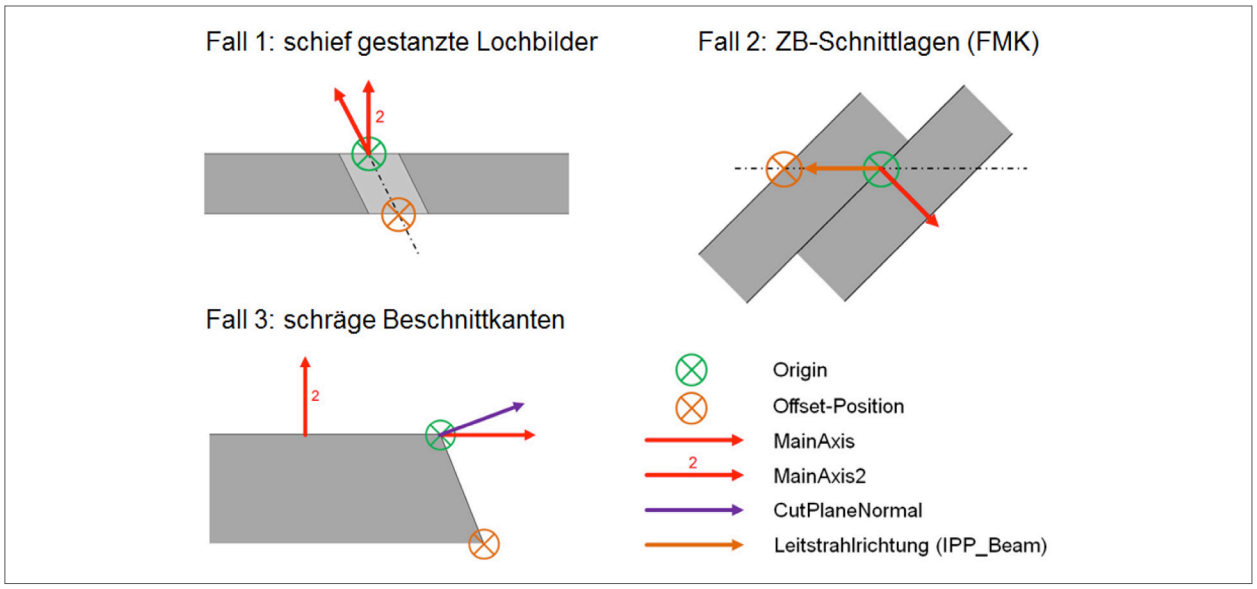


Figure 25: Offset calculation for IPEs

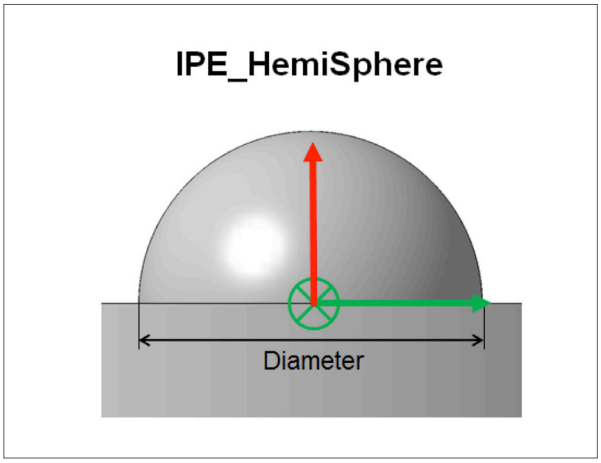


Figure 26: IPE_HemiSphere

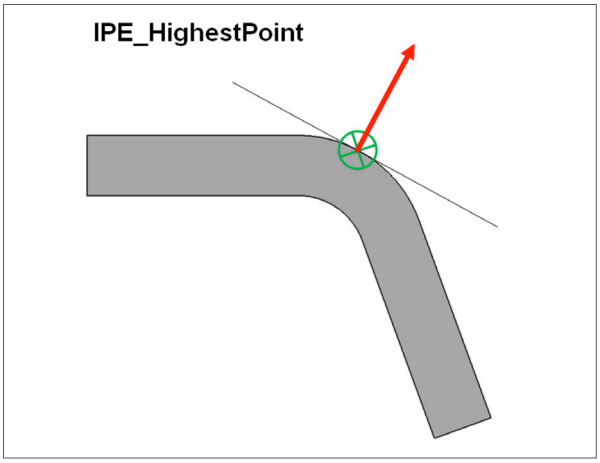


Figure 27: IPE_HighestPoint

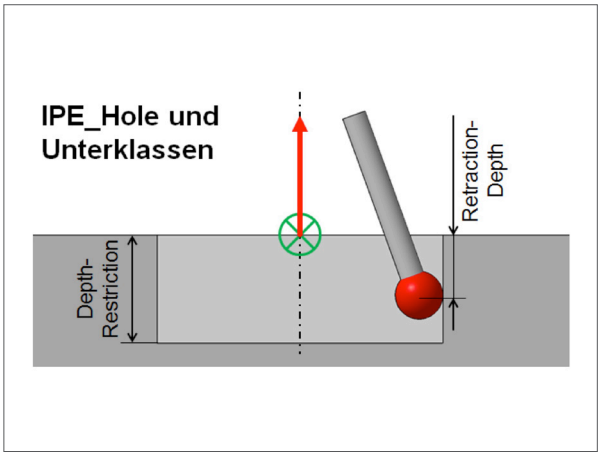


Figure 28: IPE_Hole and subclasses

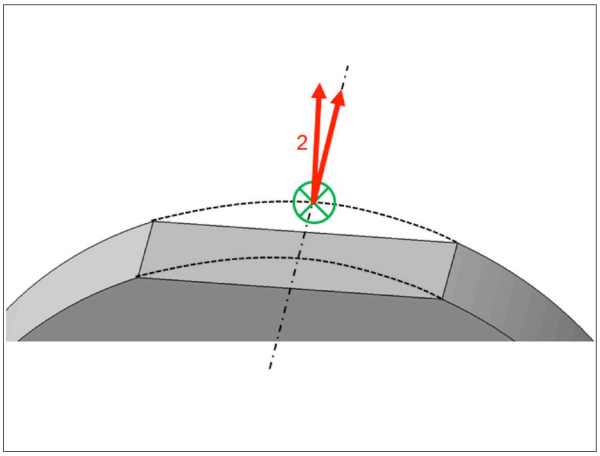


Figure 29: IPE_Hole and subclasses

The hole axis is not perpendicular to the surface. *Origin* is on the theoretical intersecting point of the hole axis and the surface. *MainAxis2* defines the surface normal as a substitute for the position of *Origin*.

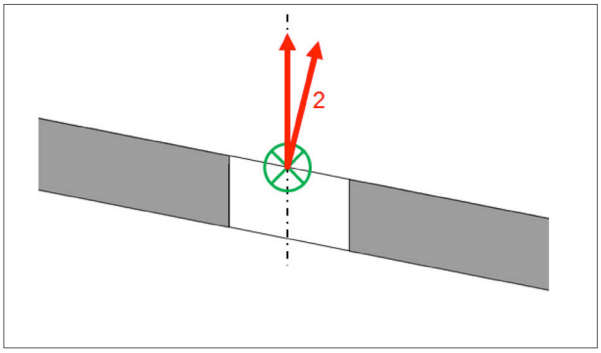


Figure 30: IPE_Hole and subclasses

MainAxis is the hole cutting direction. *MainAxis2* is the surface normal. *MainAxis2* is not required if the hole direction is normal.

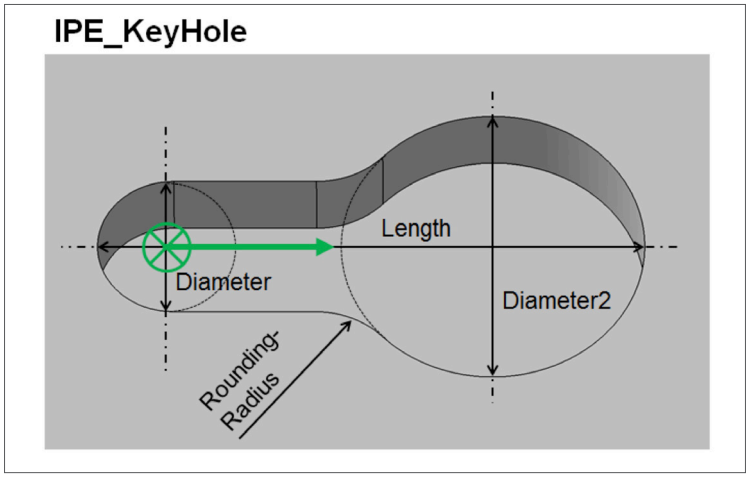


Figure 31: IPE_Keyhole

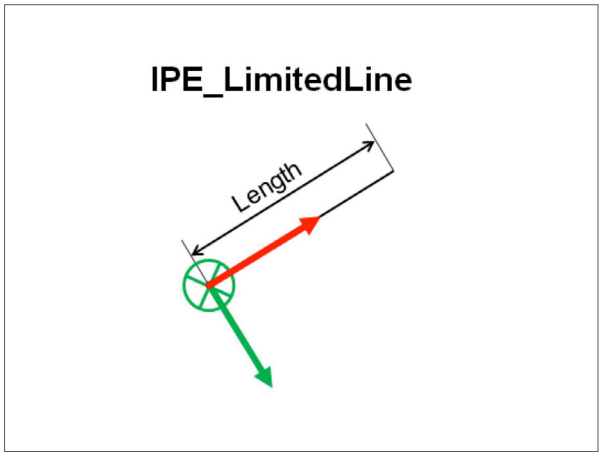


Figure 32: IPE_LimitedLine

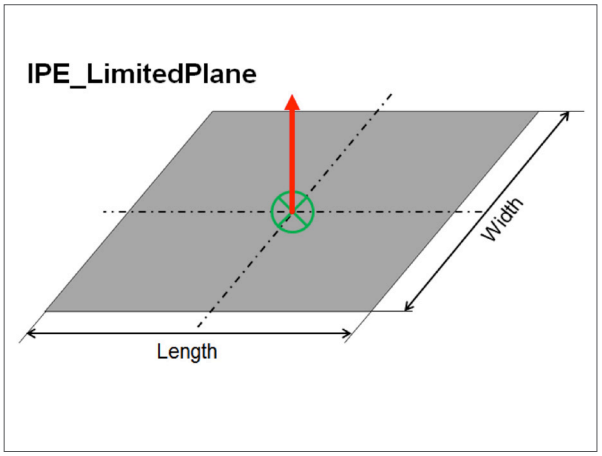


Figure 33: IPE_LimitedPlane

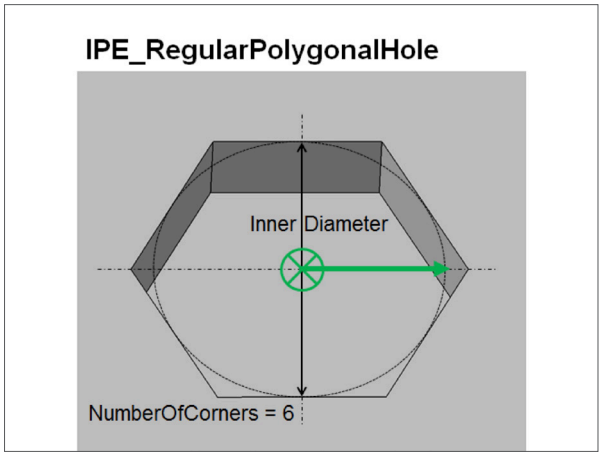


Figure 34: RectangularPolygonalHole

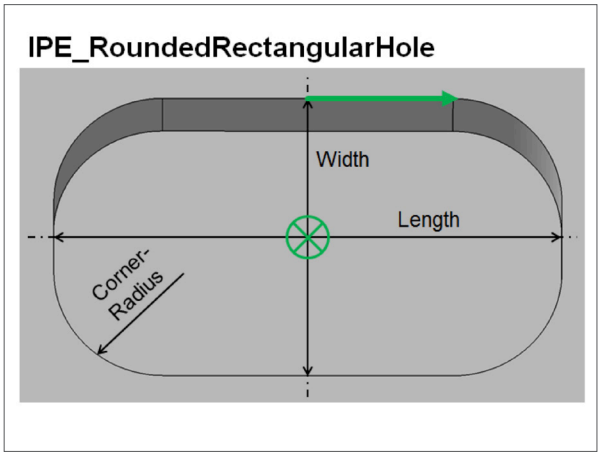


Figure 35. IPE_RoundedRectangularHole

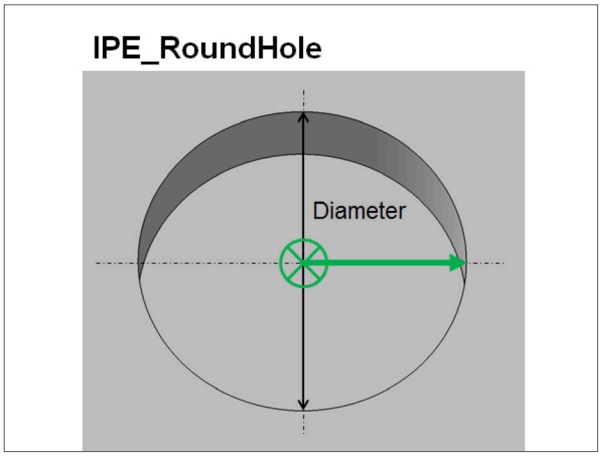


Figure 36: IPE_RoundHole

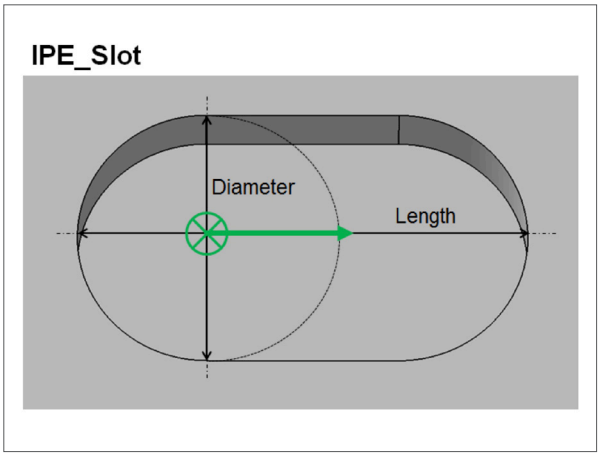


Figure 37: IPE_Slot

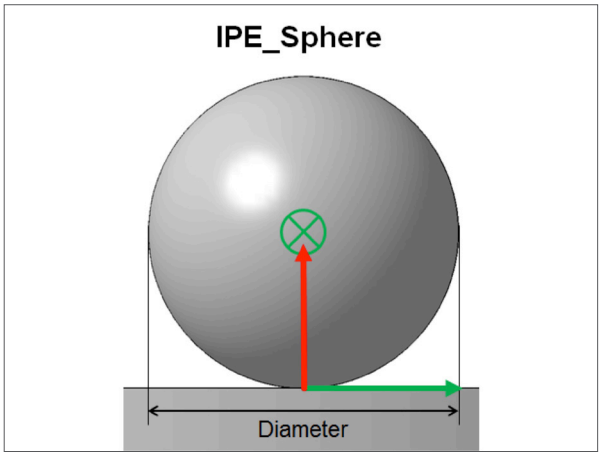


Figure 38: IPE_Sphere

5.2.5 *InspectionTasks*

The purpose of *InspectionTasks* is to provide users with a tool for maintaining an overview of (a) the *QAversions* in the QA product structures and (b) the versionmanaged inspection plans. The *InspectionTask* instance represents the reason for creating a particular *QAversion* or a particular *InspectionPlan*.

There is a separate transport object specifically for *InspectionTasks*. Ways in which clients can use such operations include requesting all *InspectionTasks*, offering them to users for selection, and then accessing and loading the product structure or inspection plan via the references contained in the *InspectionTask* instances. The *InspectionTasks* can also be displayed concurrently to provide more extensive information on the part nodes when a user is navigating through a QA product structure.

5.2.6 InspectionEffectivity

InspectionEffectivity belongs to the class *"CalculationsAndStrategies"* and also includes the *InspectionSeverity*. You can use *InspectionEffectivity* to control the validity of an *InspectionPlan*. In addition, *InspectionSeverity* can be used to transfer information as to whether the inspection is related to safety, documentation or functions.

5.2.7 InspectionProgram

Transfers the measurement program's meta data. It does not present the contents of measurement programs in full. Instead, it is simply used to manage a reference to an actual measurement program.

5.2.8 Quality criteria, tolerances and reference systems

A quality criterion describes a requirement applied to the geometry of a manufactured product. Essentially, it assigns a tolerance to a geometric point. The latter is represented as instances of subclasses of IPE, the former as instances of subclasses of Tolerance. In addition, one or more assembly stages (*Part_Assembly*) are assigned to the quality criterion to indicate where the tolerances apply. A given geometric point may therefore have different quality criteria if it is present at multiple assembly stages. The IPE instances contain the target values, with permitted deviations expressed as tolerances. The IPEs are therefore sometimes referred to as "target" features.

Some tolerance types require a reference system in order to be defined in full. In I++ DMS, a reference system is an instance of the class *ReferenceSystem*. An instance of this class is referenced directly by the relevant Tolerance instance. The reference may be omitted from an I++ DMS data record, but the reference system should be present in the database nonetheless. If this information is missing, the global coordinate system applies.

If necessary, reference systems may include information on how they originated or were created. This can be supplied by means of the *AlignmentStrategy*, Datum and *DatumTarget* classes. These basically specify an alignment strategy, the geometric objects (Datum) required in calibration, and the geometric objects' role in the alignment strategy.

A tolerance may reference a specific attribute of a geometric object. In this case, the tolerance applies to one or more attributes of the relevant IPE instances. These attributes are listed in *QC_Single*.

5.2.8.1 Shape and orientation tolerances

For shape and orientation tolerances, the *GenericParameters* defined in the basic class *Tol_GPS* can be used to save optional additional information that is required for a complete description of the tolerance. This includes, for example, support for material conditions and other modifiers.

To ensure compatibility between the exchanging systems, we suggest the following entries for the most common properties:

MODIFIER (KEY)	VALUES (VALUE)	MEANING
ToleranceZoneType	Planar	flat tolerance zone
	Circular	circular tolerance zone
	Spherical	spherical tolerance zone
MaterialRequirement	MMR	Maximum material requirement
	LMR	Minimum material requirement
DimensionElement	UUID	UUID for dimension of the material condition
TiltedToleranceZoneAngle		Angle to axis
TiltedToleranceZoneRotation		Rotation around axis
ProjectedToleranceZonePlanes	UUID	UUID for reference plane
ProjectedToleranceZoneDistance		Distance to reference plane
ProjectedToleranceZoneLength		Length of the projected tolerance zone
UnequallyDisposedToleranceZone		Displacement

Table 1: Predefined values

5.2.8.2 References for shape and orientation tolerances

The *GenericParameter* defined in the class Date can be used to save optional additional information, which is required for a complete description of references, for references that are used by shape and orientation tolerances.

To ensure compatibility between the exchanging systems, we suggest the following entries for the most common properties:

MODIFIER (KEY)	VALUES (VALUE)	MEANING
SituationfeaturePlane		The plane is needed as a situation element
SituationfeatureLine		The straight line is required as a situation element
SituationfeaturePoint		The point is needed as a situation element
MaterialRequirement	MMR	Maximum material requirement
	LMR	Minimum material requirement
DimensionElement	UUID	UUID for dimension of the material condition

Table 2: Predefined values

The entry of a situation element as a key in the *GenericParameter* determines the use of this property, in which case the actual value plays no role.

5.2.8.3 Quality criteria's geometry reference

See the description of the *QC* class.

Every quality criterion, even in connection with distance tolerance, references exactly one IPE to select the geometry (there is no provision for referencing a geometry directly in the data schema) and one tolerance. Unless necessary and/or expected, the IPE is associated directly with *GeometricObjects* via an aggregation relationship or via a *Calculation_NominalElement*. The IPE has all the requisite attributes to accommodate the relevant target values. An *IPE* may be assigned to more than one *QC*.

5.2.9 Calculations and strategies

5.2.9.1 Calculation_ActualElementDetection

Represents a calculation to be performed or strategy to be executed that uses existing *IPEs* as input.

Example application: measurement principles, also referred to as measurement strategies.

5.2.9.2 Calculation_NominalElement

Describes how an IPE is derived from the design geometry.

A composite aggregation between *IPE_Single* and *CalculationNominalElement* indicates that the *IPE_Single* instance is derived from other *IPEs*. The attribute *IPE_Single*. *CalculatedElement* is set here to true. At the same time, the *IPE_Single* can be linked with 1..n *QC* through the inherited aggregation between *IPE* and *QC*.

The inherited attribute *Operation* is defined as follows:

Standardized operations:

- *IPP_DistanceBetween IPE_Distance:* Distance between the Origins of two in-put IPEs. This can also transfer the projections of the distance to the spatial axes X, Y and Z and the corresponding directions A and B, in addition to the Euclidean distance value. The axes A and B are transferred as standardized direction vectors using additional *OperationParameter IPP_DistAxisA_X, _Y, _Z and IPP_DistAxisB_X, _Y, _Z*.
- *IPP_AngleBetween IPE_Angle*: Angle between two input *IPEs of the type IPE_Line*; the operation takes the parameter *AngleMode* with two fixed values, interior and exterior; these determine whether an enclosed angle between two directional lines is meant, or its supplementary angle to 180 degrees.
- *IPP_Symmetry SymmetricPoint* between the origins of two input *IPEs*.
- *IPP_LiesOn*: All input elements lie on the output element (e.g. *Surface, Curve, SectionPlane*).
- *IPP_IsEdgeCurveOf*: A curve as an input element represents the edge of a hole as an output element.
- *IPP_Projection*: The output element is the result of a projection of the input element with the role *IPP_IPE_From* onto the input element with the role *IPP_IPE_To*.
- *IPP_Beam*: The output element (*IPE_Surface_Point*) lies on the point at which the beam intersects with the geometry. The direction of the beam is described by *OperationParameter IPP_BeamAxisX, IPP_BeamAxisY* and *IPP_BeamAxisZ*.
- *IPP_Intersection*: The output element is the result of the intersection of two or three input elements. The intersections listed below are supported. The application must decide on the right approach to take in the event of an error:

- *IPE_Line (3D) + IPE_Line (3D) ➠ IPE_Point.*
- *IPE_Line + IPE_Plane ➠ IPE_Point*
- *IPE_Line + IPE_Face ➠ IPE_Point*
- *IPE_Plane + IPE_Plane ➠ IPE_Line*
- *IPE_Plane + IPE_Face ➠ IPE_Curve*
- *IPE_LimitedPlane + IPE_Face ➠ IPE_Curve*
- *IPE_Plane + IPE_Plane + IPE_Plane ➠ IPE_Point*

The operand type is determined by the *IPE* instances used or by the casting operations specified in *Operand.Role*.

- *IPP_Transformation*: The output element is created from the input element through a transformation using values in OperationParameter.
- *IPP_Mirror*: The output element is created by mirroring the input element at the y-axis.
- *IPP_Construct_IPE_Line*: An *IPE_Line* is created from two *IPE* instances that are interpreted as points. The sequence of the input elements determines the direction of the *IPE_Line*. The standardized *MainAxis* of the *IPE_Line* points from the first point to the second. The sequence of the points is determined from *Operand.Index*.
- *IPP_Construct_IPE_Plane*: An *IPE_Plane* is created from three *IPE* instances that are interpreted as points. The sequence of the input elements determines the direction of the *IPE_Plane*. The *MainAxis* of the *IPE_Plane* is defined as the cross product of the connecting lines from $Point_1$ to $Point_2$ and $Point_1$ to $Point_3$ and then standardized. The sequence of the points is determined from *Operand.Index*.

5.2.10 Grouping quality criteria and inspection plan elements (QCs and IPEs)

This section concerns the classes *QC* and *IPE* and their subclasses.

To enhance the clarity of databases and allow related elements to be processed together, quality criteria and *IPE* instances can be grouped together, based on selectable aspects. The subclasses of these classes are grouped together according to a unified method, described here:

The *GroupType* attribute of the base classes *QC* and *IPE* describes the group type and, thus, the grouping criterion or reason.

Function is an attribute orthogonally opposite to the *GroupType* attribute – in other words, an independent group attribute that provides additional information on the group.

Below is a list of predefined values. Besides these, other values can be chosen freely for both attributes. The formation of groups is optional.

GROUP TYPE	MEANING
Program	Grouping with regard to subsequent measurement program modules.
InspectionLocation	Elements of one and the same measuring point.
InspectionSublocation	Elements that belong to part of a measuring point.
QCcascading	Combines grouped *QC* instances with a logical AND operation to form a new *QC* represented by the *QC_Group* instance. If all the grouped *QCs* are met, the parent *QC* is met, too.
Hierarchy	Other, hierarchical grouping.
Classification	Grouping category not specified in detail; used to assign Function values. *Function* must therefore be set.

Table 2: Grouping categories

FUNCTION	MEANING. ELEMENTS RELEVANT FOR ...
Gap	Spalt
Flush	Flush, offset, transition
RPS	RPS
Alignment	Alignment
Assembly	Assembly
Carcass	Carcass
FunctionalMeasure	Point for which a functional dimension exists
ReportingMeasure	Point to be reported
Sampling	Sampling
ProductionRun	Production run
BodyInWhite	Shell
Inline	Inline
CPK	CPK

Table 3: Predefined Function values

5.2.11 Company-specific extensions of I++ DMS

The purpose of *Extensions* is to allow application-specific data packets to be transferred via I++ DMS. However, extensions should only be used in situations where the standardized I++ DMS model is not able to adequately represent the information in question.

Every I++ DMS client must be able to accept incoming instances of extended classes in read operations without issuing error messages, and to process their standard parts. The *Extension* contents may be dropped when data is written back.

If the I++ DMS services are adapted, this must be done in such a way that any standards-compliant I++ DMS client can operate with any *Extensions* without the need for special adaptation. There are plans to include extensions in the next version of the I++ DMS recommendation and to add them to the information model as necessary.

- **Extensions** to service **operations** using the ***Universal***: parameter: clients can ignore these. An application-specific data packet can be transferred via these extensions if necessary. The parameter should be used sparingly and only when necessary.
- **Extensions** to service **operations** using the ***Sender*** parameter: This parameter must be populated by every I++ DMS application. It has to be possible to configure the values in a given application so that they conform to the application-specific values. This makes it easier to classify the data to be written.
- Each ***IppDMSentity*** can be extended by means of an ***Extension*** attribute of the type ***Extension***. See the relevant description.

5.3 Transferring data via I++ DMS

Besides using the recommended web services method, data can also be transferred by means of files, or a hybrid combination of the two. The underlying data structure is always identical and can be found in the XML schema file derived from the information model (see Annex C).

5.3.1 Service-based data exchange

When data is exchanged via web services, a separate service request must be made for each transport object; the response to that service request is the desired transport object.

Annex B describes all operations supported by I++ DMS 3.0. The individual operations are covered there in detail. The data types of the operation parameters used to transfer user data are also referred to as transport objects.

All operations can be expanded for specific applications by means of a universal parameter, *Universal*, of the type *Extension*. They also support the parameter *Sender*, used to transfer information on the application or service to be called.

The lock mode parameter *Mode*, supported by a number of services, is declared as optional. This means that it does not need to be used or understood by every application.

As a rule, search, load, save and delete operations are available for the most important information objects. In search operations, a search filter is specified (see the description of the *SearchFilter* class). The filter consists of multiple logically *ORed* lists of key-value operators. The lists are interpreted in such a way that their values are logically *ANDed*. Each search operation has a list of attributes that can be used as search keys. The key notation takes the form <informationobject>.<attribute>. *Inspection.Name* means that the attribute Name of the information object Inspection is addressed. Note that searches may include wildcards (e.g. "%"). The size of the search result set can be limited using the *ResultSetLimit* attribute in the search filter.

When defining search filters, attributes of other information objects may also be included that should not be returned by a search. This means that the result set should only include objects that are linked to objects in the additional filter. For instance, if the filter "*Part.SystemId* = 12345" is set in a search for measurements, the search only returns measurements for the part with the ID 12345.

In searches, only part of the transport object is populated with information. As a rule, this information comprises the meta data of the information object specified in the name of the search operation. By contrast, load operations return the entire information object, along with any additional information objects linked via *contains* relationships.

With deletions, the behavior is difficult to define because it is hard to ascertain exactly which information objects can be deleted additionally, depending on which other actions are being performed on the measurement data. For example, if an inspection plan is deleted, can the descriptions of the measuring elements be deleted too? For this reason, all that is stipulated here is that only the information object itself should be deleted. Whether or not all the information associated with the information object is also deleted is a matter to be decided individually in any given implementation.

This means that, in some circumstances, referenced information objects may no longer exist. In such instances, I++ DMS will throw an error when it reloads the referenced object.

5.3.2 File-based data exchange

File-based data exchange involves creating an individual XML file for every transport object to be transferred. The transport object represents the root object.

6 Open issues list

- How is the data for *InspectionEffectivity* determined?
  - Inspection planning system
  - Criteria, "relevant to documentation", etc., e.g. from parts masters
- Release management for the *InspectionPlan* and other objects?
  and/or "maturity level" e.g. for QCs?
- Include measurement uncertainty / measuring equipment capability in the model
  - Inspection equipment number attribute
  - Minimum reference to inspection equipment number
- Equipment pool management (e.g. which measurement equipment are available if there is no DME)
- Virtual measuring room (e.g. "problematic geometry")
  - Reference to, for example, clamping geometry, travel paths
- Notation for the neutral description of measurement principles
  - Catalog of measurement principles
  - Currently only reference to a single selected principle is supported

7 References

This version of the Recommendation is based on prior versions 1.0, 1.1 and 2.0. These were adopted by OEM AG, which consists of seven European automakers: AUDI, BMW, Daimler, Opel, Porsche, Volvo and VW. On behalf of the working group, we would like to thank the authors.

Authors V1.0
Hans-Martin Biedenbach (Audi AG)
Johann Stoll (Audi AG)
Helmut Tiringer (BMW AG)
Wolfhart Umlauft (Daimler AG)
Kai Gläsner (Daimler AG)

Authors V1.1
Andreas Disch (Kronion GmbH)
Johann Stoll (Audi AG)
Dr. Johann U. Zimmermann (Dr. Zimmermann engineering office)

Authors V2.0
Helmut Tiringer (BMW AG)
Klaus-Peter Niemann (VW AG)
Nadine Heinrichs (VW AG)
Andreas Disch (Kronion GmbH)
Siegfried Heinz (Kronion GmbH)
Sascha Köhn (Kronion GmbH)
Florian Langhojer (Senacor Technologies)

[DMSC2015a] DMSC; Dimensional Metrology Standards Consortium, Inc. (DMSC) (Hrsg.): Quality Information Framework (QIF) – An Integrated Model for Manufacturing Quality Information - Part 1: Overview and Fun-damental Principles Version 2.1, 2015a

[DMSC2015b] DMSC; Dimensional Metrology Standards Consortium, Inc. (DMSC) (Hrsg.): Quality Information Framework (QIF) – An Integrated Model for Manufacturing Quality Information - Part 2: Quality Information Framework (QIF) Library – Information Model and XML Schema Files Version 2.1, 2015b

[DMSC2015c] DMSC; Dimensional Metrology Standards Consortium, Inc. (DMSC) (Hrsg.): Quality Information Framework (QIF) – An Integrated Model for Manufacturing Quality Information - Part 4: QIF Plans Infor-mation Model and XML Schema File Version 2.1, 2015c

[DMSC2015d] DMSC: Quality Information Framework website. URL http://qifstandards.org/. - consulted on 2017-03-17. – QIF Standard

[DMSC2015e] DMSC: Website of the Dimensional Metrology Standards Consortium.
URL http://www.dmsc-inc.org/. - consulted am 2017-02-22

[Gar11] Gartner Group: IT Definitions and Glossary.
http://www.gartner.com/technology/it-glossary/#22_0.
[Quote from: 19.07.2011]

[Gla2010] Gläsner, Kai Henri: Herstellerneutrale Schnittstellen I++ in der Koordinatenmesstechnik. Anwendungen, Möglichkeiten und Grenzen. In: Koordinatenmesstechnik 2010: Technologien für eine wirtschaftliche Produktion, VDI-Berichte. Volume 2120. Düsseldorf: VDI-Verlag, 2010 - ISBN 978-3-18-092120-4, p. 261-268

[IMTI06] IMTI; The Integrated Manufacturing Technology Initiative (IMTI, Inc) (Hrsg.): A roadmap for metrology interoperability (No. NIST IR 7381). Gaithersburg, MD: National Institute of Standards and Technology, 2006. – dx.doi.org/10.6028/nist.ir.7381

[Ipp08] Biedenbach, H.-M., Stoll, J., Tiringer, H., Umlauft, W., & Gläsner, K. (2008). I++ DMS - Data Management Services Version 1.0.

Disch, A., Stoll, J., & Zimmermann, D. J. (2009). I++ DMS - Data Management Services Version 1.1.

Tiringer, H., Niemann, K.-P., Heinrichs, N., Disch, A., Heinz, S., Köhn, S., et al. (2014). I++ DMS - Data Management Services Version 2.0.

[Mel08] Melzer, I.: Service-orientierte Architekturen mit Web Services: Kon-zepte - Standards - Praxis. Heidelberg Spektrum Akademischer Verlag, 2008.

[Mor2016] Morse, Edward; Heysiattalab, Saeed; Barnard-Feeney, Allison; Hed-berg Jr., Thomas: Interoperability: Linking Design and Tolerancing with Metrology. In: Procedia CIRP, 14th CIRP CAT 2016 - CIRP Conference on Computer Aided Tolerancing. Volume 43 (2016), p. 13–16

prostep IVIP

VDA | Verband der Automobilindustrie

3D MDM
measurement data management

prostep ivip association

Dolivostraße 11
64293 Darmstadt
Germany

Phone +49-6151-9287336
Fax +49-6151-9287326
psev@prostep.com
www.prostep.org

ISBN 978-3-9820795-4-7
Version 3.0, 2020-3
Price 109€