13. Annex I: QM Conceptual Model

El primer annex és la documentació del model conceptual de l’eina QM.

Ha estat escrit i dissenyat per en Xavier Franch, Gemma Grau, Carme Quer, Xavier Lopez-Pelegrín i Juan P. Carvallo.
Visualització de models de qualitat mitjançant una interfície web
Visualització de models de qualitat mitjançant una interfície web

DesCOTS: QM    Conceptual Model

Xavier Franch, Gemma Grau, Carme Quer, Xavier Lopez-Pelegrín, Juan P. Carvallo

Universitat Politècnica de Catalunya (UPC)
c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)

email: {ggrau, franch, cquer, carvallo}@lsi.upc.edu
1. A conceptual model driven presentation of QM

The conceptual model can be divided into nine different parts that will be detailed in the rest of the paper, from sections 2 to 10:

- Arranging quality models in a taxonomy of categories and domains
- Classifying general objects
- Defining and grouping of quality entities
- Defining hierarchies of quality entities
- Defining metrics
- Assignment of metrics to quality entities
- Establishing relationships among quality entities
- Defining requirement patterns
- Defining a glossary

2. Arranging quality models in a taxonomy of categories and domains

A taxonomy is used for the organization of Quality Models. This taxonomy is composed of Categories (communication infrastructure, collaboration software, etc.) and Domains (workflow systems, mail servers, anti-virus tools, etc.), generalized as Quality Scopes. Domains are grouped into categories and categories on their turn are grouped into other categories. One domain or category may be part of more than one category. We attach quality models to quality scopes in the taxonomy, supporting model reuse by inheriting them downwards the hierarchy. We allow the construction of quality models for each scope in the taxonomy. The taxonomy just has a root, that is the Generic node.
Fig. 1. Arrangement of quality models in a taxonomy.

The User that constructs a quality model corresponding to a quality scope, is an Expert User of the DesCOTS system. In case, the expert user changes we are interested in knowing the user that created it (creatorUser).

3. Classifying general objects

Since we want to maintain some common attributes for different elements that are managed by QM. We introduced the class Object in order to generalize this fact. The common attributes are two strings that correspond to a description and several comments, and the Sources (bibliographic source, commercial tool, tutorial, web page,...) from were the element has been seen that needs to be part of the QM repository. The elements can be elements specific of a quality model, or may be elements that are general for the construction of any quality model (for example we will have global metrics that will be shared in the construction of all quality models). For this reason we differentiate among Global Sources (that are sources of elements interesting for the construction of any quality model) and Domain Sources (that are sources of elements related with one quality model).
Fig. 2. Generalization of common attributes for QM elements.

General objects are Quality Pieces, Quality Entities, Metrics, Quality Entity Metrics Assignments, Requirement Patterns, Families of Patterns and Relation Scales.

4. Defining and grouping of quality entities

Quality models contain quality entities customized to a particular Quality Scope. More than one Quality Model Version of a quality model may be defined. An order is necessary among the versions of a quality model ({ordered}). In order to define a new version the previous version has to be validated (validationDate), and the lastVersionState of the quality model has to be VALIDATED.

Aside of quality models, QM also deals with other Quality Fragments. Quality entities are grouped into quality fragments. Quality entities are related to quality fragments in two different ways. Quality entities may be copied from one fragment to another. If this is done, they are replicated, and each replica is a different quality entity that belongs to a different quality fragment. Thus, quality entities are not shared among quality fragments. This decision makes changes on quality entities to be local, making easier quality model management. One quality model version may have more than one quality entity root, since all quality entities in the first level of the quality model (Characteristics) will be roots of the quality model.

The other type of quality fragment are Quality Patterns. These patterns are chunks of quality entities that appear in many quality models. QM promotes reusability by allowing the construction and maintenance of a catalog of quality patterns. This catalog may be accessed by all the expert users (during the construction of quality models), and thus these users can share quality patterns as a global element of QM.

Elements that are specific of a quality model are Quality Model Specific Metrics, Requirement Patterns, Families of Patterns and Domain Sources.
5. Defining hierarchies of quality entities

As the cornerstone of our quality framework, we have chosen the ISO/IEC 9126-1 standard. It is quite generic, presents a hierarchical structure and is widespread. So we have included in our conceptual model its three types of Quality Entities: Characteristics, Subcharacteristics and Attributes. The ISO/IEC 9126 standard is not precise enough in some points and, therefore, some decisions have been taken and reflected in the conceptual model. The two most important ones are:

- Hierarchies of subcharacteristics and attributes are allowed without any restriction about its number of levels.
- An attribute or subcharacteristic may be associated to several subcharacteristics, as the standard does not forbid overlapping of software entities.

We have also introduced an extra classification hierarchy for quality entities, which indicates if an entity is Specific Quality Entity or Generic Quality Entity, i.e. if it is fully defined or not. Thus, we also have specializations for each type of specific quality entity, obtaining: Specific
Characteristics, Specific Subcharacteristics and Specific Attributes. This specialization is introduced in order to model that just specific entities may be decomposed and that metrics can be only defined for specific subcharacteristics and attributes.

One textual restriction has to be defined in order to restrict the use of specific quality entities into quality models fragments; they cannot appear into quality patterns. On the contrary, generic quality entities facilitate reusability among quality models, because they hide those details that are bound to particular context. Since they are not bound to any particular quality model, they may be included and tailored to the specific needs of several quality models. Thus this type of entity may appear in any type of quality fragment. However, a quality model cannot be considered complete if it still contains generic quality entities.

Specific subcharacteristics are named Derived Subcharacteristics if they are decomposed in a new level of subcharacteristics. On the contrary, if they are decomposed into attributes, they are named Basic Subcharacteristics. Also, specific attributes, that are decomposed in a new level of attributes are named Derived Attributes. And attributes that are not plus decomposed are named Basic Attributes.
6. Defining metrics

QM manages two types of Metrics: the Global Metrics that are general to all quality model constructions, and the Quality Model Specific Metrics, that are metrics that correspond to a specific quality model version. Metrics of each of these types may be replicated to the other type. This means that a user may decide that a metrics defined as specific, may be useful as a global metrics, or the other way round. If this is done they are just copies and may be changed independently.

We may also classify metrics as Qualitative Metrics and Quantitative Metrics. A metrics is qualitative when it is not possible to establish a precise, non-ambiguous measurement procedure to get the value of the quality entity that it evaluates, but it is possible to give an appreciative value (subjective). Otherwise, the metric is quantitative. Quantitative metrics usually catch observable quality factors of software products (Number of Retransmission Retries before Failure,…). As a desirable property, the measurement procedure for objective metrics should be always repeatable and should give the same results.

Qualitative metrics always have a metrics of type string.

Quantitative metrics may be a Formula or a Basic Metric. A quantitative metrics is basic when its value must be assigned directly by a software quality expert. Otherwise, the metrics is a formula, defined by means of a formalDefinition. A formula will be based on the quality entities from which the value is calculated. Since a quality entity may be evaluated by different metrics depending on the quality entity that they are decomposing, it is necessary to relate the formula with the Quality Entity Metrics Assignment, that is the assignment of the metrics to the quality entity (the assignment corresponds to a quality entity used in the formula, the predecessor quality entity that it is decomposing and the metrics of the quality entity when it is decomposing the predecessor quality entity).

Basic Metrics may be Simple, Set, Tuple or Function. Simple metrics are Boolean, String (for them it is possible to establish a defaultValue), Integer (for them it is possible to establish a max value and a min value), Real (for them it is possible to establish a max value and a min value), or Domain (they may be Ordered Domains or not, depending on if it is possible to establish an order among the Domain Values). The Set Elements must have a simple metrics. The Tuple Elements may have a simple metrics (Simple Tuple Elements) or a set metrics (Set Tuple Elements). Tuple elements must have a name to identify them. The Function metrics may have more than one Input Parameter and just one Output Parameters that may have any type of simple metrics but function (see textual restriction).

Finally, in order to facilitate the definition of the quality entity metrics assignment (see the next section), we have needed to add one generalization of qualitative metrics and formulas in Derived Metrics, since they will be the type of metrics suitable for derived quality entities.
7. Assignment of metrics to quality entities

To allow the evaluation of products in a domain following a quality model, we need to assign metrics to quality entities. **Quality Entity Metrics Assignment** are elements of a quality model, and the assignment of a metrics to quality entities of a quality model version must be quality model specific metrics (see the previous section).

No all the types of metrics are suitable for each type of quality entity. For this reason we have specialized the quality entity metrics assignment depending on the type of quality entity to which the assignment corresponds.
Next, we describe the relationships that are involved in the assignment depending on the type of quality entity:

- Characteristics are used as a classification level which groups the different quality entities related with it. Most of times they are not evaluated, however we allow to assign them a qualitative metrics. Thus the **Characteristic Metrics Assignment** is an association class among a qualitative metrics and the characteristic to which the metrics is assigned.

![Characteristic Metrics Assignment](image)

**Fig. 7.** Characteristic metrics assignment

- Subcharacteristics can be measured by derived metrics. Taking into account that a subcharacteristic may decompose more than one subcharacteristic (having different metrics in each decomposition), the **Subcharacteristic Metrics Assignment** is an association class among a derived metrics, the subcharacteristic to which the metrics is assigned, and the quality entity that it is decomposing in this assignation.

![Subcharacteristic Metrics Assignment](image)

**Fig. 8.** Subcharacteristic metrics assignment.
Derived attributes can be measured by derived metrics. Taking into account that an attribute may decompose more than one subcharacteristic or derived attribute (having different metrics in each decomposition), the Derived Attribute Metrics Assignment is an association class among a derived metrics, the attribute to which the metrics is assigned, and the quality entity that it is decomposing in this assignment.

![Diagram of Derived Attribute Metrics Assignment](image)

**Fig. 9.** Derived attributes metrics assignment.

Basic attributes can be measured by basic metrics. Taking into account that an attribute may decompose more than one subcharacteristic or derived attribute (having different metrics in each decomposition), the Basic Attribute Metrics Assignment is an association class among a basic metrics, the attribute to which the metrics is assigned, and the quality entity that it is decomposing in this assignment.

![Diagram of Basic Attribute Metrics Assignment](image)

**Fig. 10.** Basic attributes metrics assignment.

8. Establishing relationships among quality entities
It is possible to establish relationships among *Quality Entities* of a quality model version. These relationships play a fundamental role when analysing software requirements expressed in terms of quality entities. For instance, consider the requirements “The system shall encrypt personal data” and “The system shall provide optimal response time when retrieving a personal data record”. Both of them involve quality factors, Encryption Strategy Used and Response Time, respectively. A complete quality model should state that these two quality attributes are conflicting, which also makes the requirements mutually conflicting.

The *Relation Scale* consist of the set of possible values (*Scale Elements*) of the scale. Two examples of relation scales (with 3 and 5 scale elements) are:

IQMC-scale = (depends, conflicts, collaborates), from [FC03].

NFR-scale = (make, help, unknown, hurt, break), from [CNYM00].

Thus, two quality entities may be related by an scale element in a relation scale, and it is also possible to have the intensity of the relationship by means of the *intensity* attribute of the association class *QER Degree* (quality entity relationship degree) included for the relationship. The intensity is not a mandatory value for the association since it has not sense for all the relation scales (it has sense in the *IQMC-scale* above but not in the *NFR-scale*).

Finally, taking into account that it is possible to have refinement of relationships, that is, that relationships among subcharacteristics, then are refined in relationships among the attributes in which these subcharacteristics are decomposed, there is a reflexive association among quality entity relationship degrees.

![Diagram](image)

*Fig. 11. Relationships among quality entities.*

**9. Defining requirement patterns**
A Requirement Pattern provides a template that can be tailored to specific contexts to generate software requirements. Requirement patterns are elements of quality model versions. Thus, they are always associated to a quality model version.

Requirement patterns are classified in Families of Patterns, although it is possible to have requirement patterns that are not assigned to any family. It is also possible to have a hierarchy of families.

The specific information managed for requirement patterns are a formalDefinition and also another narrative definition. In both of them may appear Parameters, that will be filled when the requirements will be used in a selection process. In order to know which values may take these parameters, the Quality Entity Specific Metrics.

The formal definition is a formula that when evaluated must have a boolean value. It is based on the quality entities from which the Boolean value is calculated. Since a quality entity may be evaluated by different metrics depending on the quality entity that they are decomposing, it is necessary to relate the formula with the Quality Entity Metrics Assignment, that is the assignment of the metrics to the quality entity (the assignment corresponds to a quality entity used in the formula, the predecessor quality entity that it is decomposing and the metrics of the quality entity when it is decomposing the predecessor quality entity).

It is possible to establish relationships among requirement patterns of a quality model version. The relationship follow the same Relation Scale than the quality entities (see the previous section). Thus, two requirement patterns may be related by an Scale Element in a
relation scale, and it is also possible to have the intensity of the relationship by means of the *intensity* attribute of the association class *PR Degree* (pattern relationship degree) included for the relationship. The intensity is not a mandatory value for the association since it has not sense for all the relation scales.

10. Defining a glossary

A glossary helps QM to have definitions of terms used in the construction of quality models. There is just one global glossary that manages common *Glossary Terms* shared by all *Users*.

**Fig. 13. Glossary**
Visualització de models de qualitat mitjançant una interfície web
14. Annex II: QM Database and XML

El segon annex és la documentació de la base de dades vella i de l’XML que descriu cada model de qualitat.

Ha estat escrit i dissenyat per la Gemma Grau, Carme Quer i Xavier López-Pelegrín.
Visualització de models de qualitat mitjançant una interfície web

DesCOTS: QM  Database and XML

Gemma Grau, Carme Quer, Xavier Lopez-Pelegrín

Universitat Politècnica de Catalunya (UPC)

c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)

email: {ggrau, cquer}@lsi.upc.edu
1. Implementation of QM

The database used in QM is a MySQL database.

Currently we have the quality models in QM stored in XML format in a field of a table in a database. Thus, when a quality model is open, there is a servlet that takes its XML from the database and gives that XML to the client program of QM. Changes in the quality models are not stored until the user choose the “save” option of the model. When this happens the new state of the quality model is stored in the database (qm_history).

In this document you can find the structure of the database and the description of the DTD of the XML that stores the quality models.

2. Current Database Structure

```sql
-- Base de dades: 'BD_QM'

--

-- This table maintains the relationships among quality models and scopes

-- CREATE TABLE `ambit_qm` (  `qm_code` int(11) NOT NULL default '0',  `id_ambit` int(11) NOT NULL default '0',  PRIMARY KEY (`id_ambit`),  UNIQUE KEY `qm_code` (`qm_code`) ) TYPE=MyISAM;

--

-- This table has a row for each quality model

-- CREATE TABLE `quality_model` (  `code` int(11) NOT NULL auto_increment,  `name` varchar(255) NOT NULL default '',  `user` varchar(255) NOT NULL default ''
```
'lastVersionState'
enum('UNDER_CONSTRUCTION','WAITING_VALIDATION','VALIDATED') NOT NULL default 'UNDER_CONSTRUCTION',
    PRIMARY KEY ('code'),
    UNIQUE KEY 'name' ('name')
) TYPE=MyISAM AUTO_INCREMENT=34;

-- This table has a row for each version of a quality model
--
CREATE TABLE `quality_model_versions` (  
    `qm_code` int(11) NOT NULL default '0',
    `code` int(11) NOT NULL default '0',
    `name` varchar(200) NOT NULL default '',
    `creation_date` varchar(15) NOT NULL default '',
    `validation_date` varchar(15) default NULL,
    `description` varchar(255) default NULL,
    `comments` varchar(255) default NULL,
    `xml_description` longtext,
    PRIMARY KEY (`qm_code`,`code`),
    UNIQUE KEY `qm_code` (`qm_code`, `name`)  
) TYPE=MyISAM;

-- In the database we maintain the history of the changes in each quality model version
-- The `xml` field is the one that stores the XML of a quality model that you
-- can find described in the next section of this document.
--
CREATE TABLE `qm_history` (  
    `qm_code` int(11) NOT NULL default '0',
    `qm_version_code` int(11) NOT NULL default '0',
    `datetime` timestamp(14) NOT NULL,
    `xml` longtext,
    PRIMARY KEY (`qm_code`,`qm_version_code`,`datetime`)  
) TYPE=MyISAM;

---
Visualització de models de qualitat mitjançant una interfície web

-- In this table there is a row for each global metrics
--

CREATE TABLE `metrics` (  
  `id` int(11) NOT NULL auto_increment,  
  `name` varchar(255) NOT NULL default ",",  
  `explanation` varchar(255) NOT NULL default ",",  
  `comments` varchar(255) NOT NULL default ",",  
  `user` varchar(255) NOT NULL default ",",  
  `kind`  
  enum('BOOLEAN','INTEGER','REAL','STRING','DOMAIN','SET','TUPLE','FUNCTION','FORMULA','QUALITATIVE') NOT NULL default 'BOOLEAN',  
  PRIMARY KEY (`id`),  
  UNIQUE KEY `name` (`name`)  
) TYPE=MyISAM AUTO_INCREMENT=67 ;

-- ---------------------------------------------------------------------------
--
-- In this table there is a row for each global metrics of type integer or real
--

CREATE TABLE `metrics_numeric` (  
  `metric_id` int(11) NOT NULL default '0',  
  `min_value` float default NULL,  
  `max_value` float default NULL,  
  PRIMARY KEY (`metric_id`)
) TYPE=MyISAM;

-- ---------------------------------------------------------------------------
--
-- In this table there is a row for each global metrics of type string
--

CREATE TABLE `metrics_string` (  
  `metric_id` int(11) NOT NULL default '0',  
  `special_value` varchar(255) NOT NULL default ",",  
  PRIMARY KEY (`metric_id`)
) TYPE=MyISAM;
Visualització de models de qualitat mitjançant una interfície web

-- In this table there is a row for each global metrics of type domain

CREATE TABLE `metrics_domain` (  
`metric_id` int(11) NOT NULL default '0',  
`is_ordered` tinyint(1) NOT NULL default '0',  
PRIMARY KEY (`metric_id`)  
) TYPE=MyISAM;

-- In this table there is a row for each value in a global metrics of type domain

CREATE TABLE `domain_metrics_values` (  
`metric_id` int(11) NOT NULL default '0',  
`num` int(11) NOT NULL default '0',  
`value` varchar(255) NOT NULL default '',  
PRIMARY KEY (`metric_id`,`num`)  
);

-- In this table there is a row for each global metrics of type set  
-- The field `element_metric` is the identifier of the global metrics of the  
-- elements in the set.

CREATE TABLE `metrics_set` (  
`metric_id` int(11) NOT NULL default '0',  
`element_metric` int(11) NOT NULL default '0',  
PRIMARY KEY (`metric_id`)  
) TYPE=MyISAM;

-- In this table there is a row for each global metrics of type tuple

CREATE TABLE `metrics_tuple` (  

'metric_id' int(11) NOT NULL default '0',
PRIMARY KEY ('metric_id')
) TYPE=MyISAM;

-- In this table there is a row for each tuple element in global metrics of type tuple

CREATE TABLE `tuple_metrics_elements` (
  'metric_id' int(11) NOT NULL default '0',
  'num' int(11) NOT NULL default '0',
  'element_name' varchar(255) NOT NULL default '',
  'element_metric' int(11) NOT NULL default '0',
  PRIMARY KEY ('metric_id', 'num')
) TYPE=MyISAM;

-- In this table there is a row for each global metrics of type function

CREATE TABLE `metrics_function` (
  'metric_id' int(11) NOT NULL default '0',
  'out_metrics' int(11) NOT NULL default '0',
  PRIMARY KEY ('metric_id')
) TYPE=MyISAM;

-- In this table there is a row for each parameter in a global metrics of type function

CREATE TABLE `function_metrics_parameters` (
  'metric_id' int(11) NOT NULL default '0',
  'num' int(11) NOT NULL default '0',
  'parameter_metrics' int(11) NOT NULL default '0',
  PRIMARY KEY ('metric_id', 'num')
) TYPE=MyISAM;
Visualització de models de qualitat mitjançant una interfície web

-- In this table there is a row for each quality patterns
-- quality patterns are also currently stored in XML
--

CREATE TABLE `quality_patterns` (
  `username` varchar(250) NOT NULL default ",",
  `xml` longtext,
  PRIMARY KEY (`username`)
) TYPE=MyISAM;

-- In this table there is a row for each term in the glossary
--

CREATE TABLE `glossary` (
  `term` varchar(245) NOT NULL default ",",
  `username` varchar(8) NOT NULL default ",",
  `last_date` varchar(15) NOT NULL default ",",
  `definition` longtext NOT NULL,
  PRIMARY KEY (`term`, `username`)
) TYPE=MyISAM;

-- In this table there is a row for each source from where the global metrics
-- have been obtained
--

CREATE TABLE `global sources` (
  `id` int(11) NOT NULL default '0',
  `name` varchar(255) NOT NULL default ",",
  `comments` varchar(255) NOT NULL default ",",
  PRIMARY KEY (`id`)
) TYPE=MyISAM;

-- This table maintains the relationships among global metrics and global sources.
Visualització de models de qualitat mitjançant una interfície web

CREATE TABLE `metrics_sources` (  `metric_id` int(11) NOT NULL default '0',  `source_id` int(11) NOT NULL default '0',  PRIMARY KEY (`metric_id`, `source_id`)  ) TYPE=MyISAM;

3. DTD corresponding to the XML for a Quality Model

Quality Model XML Documents' structure and correctness is defined by its DTD (Document Type Declaration). DTD validation ensures element content, attributes and structure are correct.

The DOCTYPE declaration (DTD) for the Quality Model contains a name and a list of ELEMENT declarations.

```xml
<!DOCTYPE xml_model [  <!-- ELEMENT DECLARATIONS -->  ]>
```

The root Element of the Document is named after the DOCTYPE name.

ELEMENT declarations define the content for the element. The content can be defined in terms of other Elements (children), or as the text inside the Element. The #PCDATA (Parseable Character Data) content type is commonly used to specify that the Element contains text that is not markup, therefore the Element can not have children.

3.1. Common Element Declarations

In the Quality Model DTD there are several Elements that are referenced as childs of other Elements. These elements are mostly single-valued data such as name or explanation, or multi-valued as in a source list.

The #PCDATA (Parseable Character Data) content type is used to specify that the Element's content is the String text inside the Element.
Visualització de models de qualitat mitjançant una interfície web

<!ELEMENT name (#PCDATA)>
<!ELEMENT explanation (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ATTLIST user id ID #REQUIRED>
<!ELEMENT sources (source*)>
<!ELEMENT source (#PCDATA)>
<!ATTLIST source id ID #REQUIRED>

Examples:

<name> Suitability </name>

<sources>
  <source id="1"> ISO9126-1 Standard </source>
  <source id="2"> Various Sources </source>
</sources>

3.2. Root Element Declaration

The root Element of the Document has three child nodes. The first two of them are detailed below.

<!ELEMENT xml_model(roots, key_classes, qmodel_classes)>

The roots Element contains the identifiers of the Quality Model Entities that represent the first level of decomposition of the Quality Model (characteristics).

<!ELEMENT roots (root_item*)>
<!ELEMENT root_item (#PCDATA)>
<!ATTLIST root_item id IDREF #REQUIRED>

The key classes Element contains the new identifiers to assign in the Quality Model and is only used for Quality Model construction purposes.

<!ELEMENT key_classes (source_counter, entity_counter, metric_counter, evaluation_counter, entity_req_counter, pattern_counter, relation_scale_counter,
Visualització de models de qualitat mitjançant una interfície web

```
relation_kind_counter, relation_counter>
</!ELEMENT source_counter (#PCDATA)>
</!ELEMENT entity_counter (#PCDATA)>
</!ELEMENT metric_counter (#PCDATA)>
</!ELEMENT evaluation_counter (#PCDATA)>
</!ELEMENT entity_req_counter (#PCDATA)>
</!ELEMENT pattern_counter (#PCDATA)>
</!ELEMENT relation_scale_counter (#PCDATA)>
</!ELEMENT relation_kind_counter (#PCDATA)>
</!ELEMENT relation_counter (#PCDATA)>
```

Example:

```
<xml_model>
   <roots>
      <root_item id="1">Functionality</root_item>
      <root_item id="2">Reliability</root_item>
      <root_item id="3">Usability</root_item>
      <root_item id="4">Efficiency</root_item>
      <root_item id="5">Maintainability</root_item>
      <root_item id="6">Portability</root_item>
   </roots>
   <key_classes>
      <source_counter>2</source_counter>
      <metric_counter>1</metric_counter>
      <evaluation_counter>1</evaluation_counter>
      <entity_req_counter>1</entity_req_counter>
      <pattern_counter>1</pattern_counter>
      <relation_scale_counter>1</relation_scale_counter>
      <relation_kind_counter>1</relation_kind_counter>
   </key_classes>
   <qmodel_classes>...
</qmodel_classes>
</xml_model>
```

### 3.3. Quality Model Elements

The `qmodel_classes` Element contains the Elements with the information about all aspects of the Quality Model. Each of these aspects will be detailed in separate sections.
Visualització de models de qualitat mitjançant una interfície web

relation_scale_list, requirement_list, 
requirement_pattern_roots, 
requirement_pattern_list>

The sources_list Element contains the information about the sources in the Quality Model.

Example:

```xml
<qmodel_classes>
  <sources_list>
    <source id="1">ISO/IEC 9126-1:2001</source>
  </sources_list>
  <quality_entity_list> ... </quality_entity_list>
  <evaluation_list> ... </evaluation_list>
  <metrics_list> ... </metrics_list>
  <relation_list> ... </relation_list>
  <relation_kind_list> ... </relation_kind_list>
  <relation_scale_list> ... </relation_scale_list>
  <requirement_list> ... </requirement_list>
  <requirement_pattern_roots> ... </requirement_pattern_roots>
  <requirement_pattern_list> ... </requirement_pattern_list/>
</qmodel_classes>
```

3.4. Quality Entity List Element

The quality_entity_list Element contains the Elements that represent all the Quality Entities existing in the Quality Model.

```
<!ELEMENT quality_entity_list (quality_entity*)>
```
Visualització de models de qualitat mitjançant una interfície web

The quality_entity Element has three attributes which specify Id, Entity kind and decomposition level. Please note that child Elements marked "?" are optional.

```
<!ELEMENT quality_entity (quality_entity_id,
 quality_entity_attributes,
 user, sources?, comments?,
 entity_successors?,entity_evaluations?,
 entity_requirements?,entity_relations?,
 entity_composition?)>

<!ATTLIST quality_entity id ID #REQUIRED>
<!ATTLIST quality_entity kind
 (characteristic|subcharacteristic|attribute
 |basic_attribute|derived_attribute) #REQUIRED>
<!ATTLIST quality_entity level (generic|specific)
 #REQUIRED>
```

The quality_entity_id Element contains the name of the Quality Entity.

```
<!ELEMENT quality_entity_id (#PCDATA)>
```

The quality_entity_attributes Element contains the name and explanation as the common Elements previously defined (user, sources and comments are also used). Please note the name is replicated, and always up to date with the name in the quality_entity_id Element.

```
<!ELEMENT quality_entity_attributes (name,explanation)>
```

The entity_successors, entity_evaluations, entity_relations and entity_requirements Elements contain lists of Elements, which have an Id attribute referencing another Element of the Model. They also have the name of the referenced Element in the #PCDATA content, and should be ignored to avoid inconsistency.

```
<!ELEMENT entity_successors (successor_id*)>
  <!ELEMENT successor_id (#PCDATA)>
  <!ATTLIST successor_id id IDREF #REQUIRED>
```
Visualització de models de qualitat mitjançant una interfície web

The entity_evaluations Element contains the id's of the Metrics Assignments of the Quality Entity.

```xml
<!ELEMENT entity_evaluations (evaluation_ref*)>
<!ELEMENT evaluation_ref (#PCDATA)>
<!ATTLIST evaluation_ref id IDREF #REQUIRED>
<!ELEMENT entity_requirements (requirement_ref*)>
<!ELEMENT requirement_ref (#PCDATA)>
<!ATTLIST requirement_ref id IDREF #REQUIRED>
<!ELEMENT entity_relations (relation_ref*)>
<!ELEMENT relation_ref (#PCDATA)>
<!ATTLIST relation_ref id IDREF #REQUIRED>
```

The entity_composition Element contains name and description Elements for the composition of the Entity.

```xml
<!ELEMENT entity_composition (name,description)>
```

**Example:**

```xml
<quality_entity id="1" kind="characteristic" level="specific">
    <quality_entity_id>Functionality</quality_entity_id>
    <quality_entity_attributes>
        <name>Functionality</name>
        <explanation>The capability of the software ...</explanation>
    </quality_entity_attributes>
    <user id="1">GESSI</user>
    <sources>
        <source id="1">ISO/IEC 9126-1:2001</source>
    </sources>
    <comments/>
    <entity_successors>
        <successor_id id="7">Suitability</successor_id>
        <successor_id id="8">Accuracy</successor_id>
        <successor_id id="9">Interoperability</successor_id>
        <successor_id id="10">Security</successor_id>
        <successor_id id="11">Functionality Compliance</successor_id>
    </entity_successors>
<entity_evaluations>
    <evaluation_ref id="14">Metrics Assignment</evaluation_ref>
</entity_evaluations>
```
3.5. Evaluation List Element (Metrics Assignments)

Quality Entities can have one or more metrics assignment, each one of which has one identifier and some additional information. The evaluation list Element contains all this information related to Metrics Assignments of one Quality Entity.

Metric Assignments are the relationship between a Quality Entity (with a given parent Entity) and a Metrics.

"<!ELEMENT evaluation_list (evaluation*)>"+

A Metrics Assignment Element is composed of four childs, which are detailed below, and one Attribute for the id.

The previous sources Element definition is reused for the last child.

<!ELEMENT evaluation (evaluation_id, evaluation_context, evaluation_attributes, sources?)>
<!ATTLIST evaluation id ID #REQUIRED>

The evaluation_id Element contains the name of the Metrics Assignment. This name is generated automatically, and does not report any information because it can be derived from the rest of the information.

<!ELEMENT evaluation_id (#PCDATA)>

The evaluation_context Element contains the references for the Metrics Assignment. As usual, each child Element contains an Id Attribute referencing the desired Element, and #PCDATA with the Name, which should be ignored to avoid inconsistency.
Visualització de models de qualitat mitjançant una interfície web

<!ELEMENT evaluation_context
(eval_entity, eval_predecessor, eval_metric)>
<!ELEMENT eval_entity (#PCDATA)>
<!ATTLIST eval_entity id IDREF #REQUIRED>
<!ELEMENT eval_predecessor (#PCDATA)>
<!ATTLIST eval_predecessor id IDREF #REQUIRED>
<!ELEMENT eval_metric (#PCDATA)>
<!ATTLIST eval_metric id IDREF #REQUIRED>

The evaluation_attributes Element contains an explanation for the Metrics Assignment, and a measuring protocol in the universal_property Element.

<!ELEMENT evaluation_attributes
(explanation, universal_property)>
<!ELEMENT universal_property (#PCDATA)>

Example:

<evaluation_list>
  <evaluation_id="1">
    <evaluation_id> Metrics Assignment </evaluation_id>
    <evaluation_context>
      <eval_entity id="1">Attribute 1</eval_entity>
      <eval_predecessor id="11">Compliance</eval_predecessor>
      <eval_metric id="2">Operative Systems</eval_metric>
    </evaluation_context>
    <evaluation_attributes>
      <explanation/>
      <universal_property/>
    </evaluation_attributes>
    <comments/>
    <sources/>
  </evaluation>
  <evaluation_id="2">
    <evaluation_id> Metrics Assignment </evaluation_id>
    <evaluation_context>
      <eval_entity id="2">Attribute 5</eval_entity>
      <eval_predecessor id="16">Suitability</eval_predecessor>
      <eval_metric id="2">Languages</eval_metric>
    </evaluation_context>
    <evaluation_attributes>
      <explanation/>
      <universal_property/>
    </evaluation_attributes>
    <comments/>
    <sources/>
  </evaluation>
</evaluation_list>
3.6. Metrics List Element

The metrics_list Element contains a list of all the Metrics Elements representing the Metrics existing in the Quality Model.

```xml
<!ELEMENT metrics_list (metric*)>
```

The metric Element contains all the information of one Metrics, and has two Attributes: one for the Id of the Metrics and another for the Metrics kind.

```xml
<!ELEMENT metric (metric_id,kind)>  
<!ATTLIST metric id ID #REQUIRED>  
<!ATTLIST metric kind (qualitative | formula | integer | real | boolean | string | domain | set | tuple | function) #REQUIRED>
```

The metric_id Element contains the name of the Metrics.

```xml
<!ELEMENT metric_id (#PCDATA)>
```

The kind Element contains one of the Metrics kind specific Elements, which have all necessary information related to the kind they represent. This Element is optional although it is not recommended to be missing in order to avoid incompleteness.

```xml
<!ELEMENT kind ( (qualitative | formula | integer | real | boolean | string | domain | set | tuple | function) ?)>
```

Example:

```xml
<metric id="3" kind="domain">
  <metric_id>Operative Systems</metric_id>
  <kind>
    <domain>
      ...
    </domain>
  </kind>
</metric>
```
Visualització de models de qualitat mitjançant una interfície web
DesCOTS: QM  Conceptual Model

Xavier Franch, Gemma Grau, Carme Quer, Xavier Lopez-Pelegrín, Juan P. Carvallo

Universitat Politècnica de Catalunya (UPC)

c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)

email: {ggau, franch, cquer, carvallo}@lsi.upc.edu
1. **A conceptual model driven presentation of QM**

   The conceptual model can be divided into nine different parts that will be detailed in the rest of the paper, from sections 2 to 10:
   - Arranging quality models in a taxonomy of categories and domains
   - Classifying general objects
   - Defining and grouping of quality entities
   - Defining hierarchies of quality entities
   - Defining metrics
   - Assignment of metrics to quality entities
   - Establishing relationships among quality entities
   - Defining requirement patterns
   - Defining a glossary

2. **Arranging quality models in a taxonomy of categories and domains**

   A taxonomy is used for the organization of *Quality Models*. This taxonomy is composed of *Categories* (communication infrastructure, collaboration software, etc.) and *Domains* (workflow systems, mail servers, anti-virus tools, etc.), generalized as *Quality Scopes*. Domains are grouped into categories and categories on their turn are grouped into other categories. One domain or category may be part of more than one category. We attach quality models to quality scopes in the taxonomy, supporting model reuse by inheriting them downwards the hierarchy. We allow the construction of quality models for each scope in the taxonomy. The taxonomy just has a root, that is the *Generic* node.
The User that constructs a quality model corresponding to a quality scope, is an Expert User of the DesCOTS system. In case, the expert user changes we are interested in knowing the user that created it (creatorUser).

3. Classifying general objects

Since we want to maintain some common attributes for different elements that are managed by QM. We introduced the class Object in order to generalize this fact. The common attributes are two strings that correspond to a description and several comments, and the Sources (bibliographic source, commercial tool, tutorial, web page,...) from were the element has been seen that needs to be part of the QM repository. The elements can be elements specific of a quality model, or may be elements that are general for the construction of any quality model (for example we will have global metrics that will be shared in the construction of all quality models). For this reason we differentiate among Global Sources (that are sources of elements interesting for the construction of any quality model) and Domain Sources (that are sources of elements related with one quality model).
Visualització de models de qualitat mitjançant una interfície web

![Diagram](image)

**Fig. 2.** Generalization of common attributes for QM elements.

General objects are Quality Pieces, Quality Entities, Metrics, Quality Entity Metrics Assignments, Requirement Patterns, Families of Patterns and Relation Scales.

4. **Defining and grouping of quality entities**

*Quality models* contain quality entities customized to a particular *Quality Scope*. More than one *Quality Model Version* of a quality model may be defined. An order is necessary among the versions of a quality model ({*ordered*}). In order to define a new version the previous version has to be validated (*validationDate*), and the *lastVersionState* of the quality model has to be *VALIDATED*.

Aside of quality models, QM also deals with other *Quality Fragments*. *Quality entities* are grouped into quality fragments. Quality entities are related to quality fragments in two different ways. Quality entities may be copied from one fragment to another. If this is done, they are replicated, and each replica is a different quality entity that belongs to a different quality fragment. Thus, quality entities are not shared among quality fragments. This decision makes changes on quality entities to be local, making easier quality model management. One quality model version may have more than one quality entity *root*, since all quality entities in the first level of the quality model (*Characteristics*) will be roots of the quality model.

The other type of quality fragment are *Quality Patterns*. These patterns are chunks of quality entities that appear in many quality models. QM promotes reusability by allowing the construction and maintenance of a catalog of quality patterns. This catalog may be accessed by all the expert users (during the construction of quality models), and thus these users can share quality patterns as a global element of QM.

Elements that are specific of a quality model are *Quality Model Specific Metrics, Requirement Patterns, Families of Patterns* and *Domain Sources*.
5. Defining hierarchies of quality entities

As the cornerstone of our quality framework, we have chosen the ISO/IEC 9126-1 standard. It is quite generic, presents a hierarchical structure and is widespread. So we have included in our conceptual model its three types of Quality Entities: Characteristics, Subcharacteristics and Attributes. The ISO/IEC 9126 standard is not precise enough in some points and, therefore, some decisions have been taken and reflected in the conceptual model. The two most important ones are:

- Hierarchies of subcharacteristics and attributes are allowed without any restriction about its number of levels.
- An attribute or subcharacteristic may be associated to several subcharacteristics, as the standard does not forbid overlapping of software entities.

We have also introduced an extra classification hierarchy for quality entities, which indicates if an entity is Specific Quality Entity or Generic Quality Entity, i.e. if it is fully defined or not. Thus, we also have specializations for each type of specific quality entity, obtaining: Specific
Characteristics, Specific Subcharacteristics and Specific Attributes. This specialization is introduced in order to model that just specific entities may be decomposed and that metrics can be only defined for specific subcharacteristics and attributes.

One textual restriction has to be defined in order to restrict the use of specific quality entities into quality models fragments; they cannot appear into quality patterns. On the contrary, generic quality entities facilitate reusability among quality models, because they hide those details that are bound to particular context. Since they are not bound to any particular quality model, they may be included and tailored to the specific needs of several quality models. Thus this type of entity may appear in any type of quality fragment. However, a quality model cannot be considered complete if it still contains generic quality entities.

Specific subcharacteristics are named Derived Subcharacteristics if they are decomposed in a new level of subcharacteristics. On the contrary, if they are decomposed into attributes, they are named Basic Subcharacteristics. Also, specific attributes, that are decomposed in a new level of attributes are named Derived Attributes. And attributes that are not plus decomposed are named Basic Attributes.
6. Defining metrics

QM manages two types of Metrics: the Global Metrics that are general to all quality model constructions, and the Quality Model Specific Metrics, that are metrics that correspond to an specific quality model version. Metrics of each of these types may be replicated to the other type. This means that a user may decide that a metrics defined as specific, may be useful as a global metrics, or the other way round. If this is done they are just copies and may be changed independently.

We may also classify metrics as Qualitative Metrics and Quantitative Metrics. A metrics is qualitative when it is not possible to establish a precise, non-ambiguous measurement procedure to get the value of the quality entity that it evaluates, but it is possible to give an appreciative value (subjective). Otherwise, the metric is quantitative. Quantitative metrics usually catch observable quality factors of software products (Number of Retransmission Retries before Failure, ...). As a desirable property, the measurement procedure for objective metrics should be always repeatable and should give the same results.

Qualitative metrics always have a metrics of type string.

Quantitative metrics may be a Formula or a Basic Metric. A quantitative metrics is basic when its value must be assigned directly by a software quality expert. Otherwise, the metrics is a formula, defined by means of a formalDefinition. A formula will be based on the quality entities from which the value is calculated. Since a quality entity may be evaluated by different metrics depending on the quality entity that they are decomposing, it is necessary to relate the formula with the Quality Entity Metrics Assignement, that is the assignment of the metrics to the quality entity (the assignment corresponds to a quality entity used in the formula, the predecessor quality entity that it is decomposing and the metrics of the quality entity when it is decomposing the predecessor quality entity).

Basic Metrics may be Simple, Set, Tuple or Function. Simple metrics are Boolean, String (for them it is possible to establish a defaultValue), Integer (for them it is possible to establish a max value and a min value), Real (for them it is possible to establish a max value and a min value), or Domain (they may be Ordered Domains or not, depending on if it is possible to establish an order among the Domain Values). The Set Elements must have a simple metrics. The Tuple Elements may have a simple metrics (Simple Tuple Elements) or a set metrics (Set Tuple Elements). Tuple elements must have a name to identify them. The Function metrics may have more than one Input Parameter and just one Output Parameters that may have any type of simple metrics but function (see textual restriction).

Finally, in order to facilitate the definition of the quality entity metrics assignment (see the next section), we have needed to add one generalization of qualitative metrics and formulas in Derived Metrics, since they will be the type of metrics suitable for derived quality entities.
7. Assignment of metrics to quality entities

To allow the evaluation of products in a domain following a quality model, we need to assign metrics to quality entities. *Quality Entity Metrics Assignment* are elements of a quality model, and the assignment of a metric to quality entities of a quality model version must be quality model specific metrics (see the previous section).

No all the types of metrics are suitable for each type of quality entity. For this reason we have specialized the quality entity metrics assignment depending on the type of quality entity to which the assignment corresponds.

![Fig. 6. Characteristic metrics assignment](image-url)
Next, we describe the relationships that are involved in the assignment depending on the type of quality entity:

- Characteristics are used as a classification level which groups the different quality entities related with it. Most of the times they are not evaluated, however we allow to assign them a qualitative metrics. Thus the Characteristic Metrics Assignment is an association class among a qualitative metrics and the characteristic to which the metrics is assigned.

![Characteristic Metrics Assignment Diagram]

**Fig. 7.** Characteristic metrics assignment

- Subcharacteristics can be measured by derived metrics. Taking into account that a subcharacteristic may decompose more than one subcharacteristic (having different metrics in each decomposition), the Subcharacteristic Metrics Assignment is an association class among a derived metrics, the subcharacteristic to which the metrics is assigned, and the quality entity that it is decomposing in this assignment.

![Subcharacteristic Metrics Assignment Diagram]

**Fig. 8.** Subcharacteristic metrics assignment.
Visualització de models de qualitat mitjançant una interfície web

- Derived attributes can be measured by derived metrics. Taking into account that an attribute may decompose more than one subcharacteristic or derived attribute (having different metrics in each decomposition), the Derived Attribute Metrics Assignment is an association class among a derived metrics, the attribute to which the metrics is assigned, and the quality entity that it is decomposing in this assignment.

![Diagram of Derived Attribute Metrics Assignment]

**Fig. 9.** Derived attributes metrics assignment.

- Basic attributes can be measured by basic metrics. Taking into account that an attribute may decompose more than one subcharacteristic or derived attribute (having different metrics in each decomposition), the Basic Attribute Metrics Assignment is an association class among a basic metrics, the attribute to which the metrics is assigned, and the quality entity that it is decomposing in this assignment.

![Diagram of Basic Attribute Metrics Assignment]

**Fig. 10.** Basic attributes metrics assignment.

8. Establishing relationships among quality entities
It is possible to establish relationships among Quality Entities of a quality model version. These relationships play a fundamental role when analysing software requirements expressed in terms of quality entities. For instance, consider the requirements “The system shall encrypt personal data” and “The system shall provide optimal response time when retrieving a personal data record”. Both of them involve quality factors, Encryption Strategy Used and Response Time, respectively. A complete quality model should state that these two quality attributes are conflicting, which also makes the requirements mutually conflicting.

The Relation Scale consist of the set of possible values (Scale Elements) of the scale. Two examples of relation scales (with 3 and 5 scale elements) are:

IQMC-scale = (depends, conflicts, collaborates), from [FC03].

NFR-scale = (make, help, unknown, hurt, break), from [CNYM00].

Thus, two quality entities may be related by an scale element in a relation scale, and it is also possible to have the intensity of the relationship by means of the intensity attribute of the association class QER Degree (quality entity relationship degree) included for the relationship. The intensity is not a mandatory value for the association since it has not sense for all the relation scales (it has sense in the IQMC-scale above but not in the NFR-scale).

Finally, taking into account that it is possible to have refinement of relationships, that is, that relationships among subcharacteristics, then are refined in relationships among the attributes in which these subcharacteristics are decomposed, there is a reflexive association among quality entity relationship degrees.

![Diagram](image-url)

**Fig. 11.** Relationships among quality entities.

9. **Defining requirement patterns**
A Requirement Pattern provides a template that can be tailored to specific contexts to generate software requirements. Requirement patterns are elements of quality model versions. Thus, they are always associated to a quality model version.

Requirement patterns are classified in Families of Patterns, although it is possible to have requirement patterns that are not assigned to any family. It is also possible to have a hierarchy of families.

![Diagram of Requirement Patterns](image)

Fig. 12. Requirement patterns.

The specific information managed for requirement patterns are a formalDefinition and also another narrative definition. In both of them may appear Parameters, that will be filled when the requirements will be used in a selection process. In order to know which values may take these parameters, the Quality Entity Specific Metrics.

The formal definition is a formula that when evaluated must have a boolean value. It is based on the quality entities from which the Boolean value is calculated. Since a quality entity may be evaluated by different metrics depending on the quality entity that they are decomposing, it is necessary to relate the formula with the Quality Entity Metrics Assignment, that is the assignment of the metrics to the quality entity (the assignment corresponds to a quality entity used in the formula, the predecessor quality entity that it is decomposing and the metrics of the quality entity when it is decomposing the predecessor quality entity).

It is possible to establish relationships among requirement patterns of a quality model version. The relationship follow the same Relation Scale than the quality entities (see the previous section). Thus, two requirement patterns may be related by an Scale Element in a
relation scale, and it is also possible to have the intensity of the relationship by means of the intensity attribute of the association class PR Degree (pattern relationship degree) included for the relationship. The intensity is not a mandatory value for the association since it has not sense for all the relation scales.

10. Defining a glossary

A glossary helps QM to have definitions of terms used in the construction of quality models. There is just one global glossary that manages common Glossary Terms shared by all Users.

Fig. 13. Glossary
Visualització de models de qualitat mitjançant una interfície web
14. Annex II: QM Database and XML

El segon annex és la documentació de la base de dades vella i de l'XML que descriu cada model de qualitat.

Ha estat escrit i dissenyat per la Gemma Grau, Carme Quer i Xavier López-Pelegrín.
Visualització de models de qualitat mitjançant una interfície web

DesCOTS: QM Database and XML

Gemma Grau, Carme Quer, Xavier Lopez-Pelegrín

Universitat Politècnica de Catalunya (UPC)

c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)

email: {ggrau, cquer}@lsi.upc.edu
1. Implementation of QM

The database used in QM is a MySQL database.

Currently we have the quality models in QM stored in XML format in a field of a table in a database. Thus, when a quality model is open, there is a servlet that takes its XML from the database and gives that XML to the client program of QM. Changes in the quality models are not stored until the user chooses the “save” option of the model. When this happens the new state of the quality model is stored in the database (qm_history).

In this document you can find the structure of the database and the description of the DTD of the XML that stores the quality models.

2. Current Database Structure

```sql
-- Base de dades: 'BD_QM'
```

```sql
-- ---------------------------------------------
-- This table maintains the relationships among quality models and scopes
--
CREATE TABLE `ambit_qm` (  
    `qm_code` int(11) NOT NULL default '0',  
    `id_ambit` int(11) NOT NULL default '0',  
    PRIMARY KEY (`id_ambit`),  
    UNIQUE KEY `qm_code` (`qm_code`)  
) TYPE=MyISAM;
```

```sql
-- ---------------------------------------------
-- This table has a row for each quality model
--
CREATE TABLE `quality_model` (  
    `code` int(11) NOT NULL auto_increment,  
    `name` varchar(255) NOT NULL default '',  
    `user` varchar(255) NOT NULL default ''  
) TYPE=MyISAM;
```
Visualització de models de qualitat mitjançant una interfície web

`lastVersionState`
```
enum('UNDER_CONSTRUCTION','WAITING_VALIDATION','VALIDATED') NOT NULL default 'UNDER_CONSTRUCTION',
   PRIMARY KEY (`code`),
   UNIQUE KEY `name` (`name`)
) TYPE=MyISAM AUTO_INCREMENT=34 ;
```

-- This table has a row for each version of a quality model

CREATE TABLE `quality_model_versions` (
   `qm_code` int(11) NOT NULL default '0',
   `code` int(11) NOT NULL default '0',
   `name` varchar(200) NOT NULL default '',
   `creation_date` varchar(15) NOT NULL default '',
   `validation_date` varchar(15) default NULL,
   `description` varchar(255) default NULL,
   `comments` varchar(255) default NULL,
   `xml_description` longtext,
   PRIMARY KEY (`qm_code`,'code'),
   UNIQUE KEY `qm_code` (`qm_code`,'name')
) TYPE=MyISAM;

-- In the database we maintain the history of the changes in each quality model version
-- The `xml` field is the one that stores the XML of a quality model that you
-- can find described in the next section of this document.

CREATE TABLE `qm_history` (
   `qm_code` int(11) NOT NULL default '0',
   `qm_version_code` int(11) NOT NULL default '0',
   `datetime` timestamp(14) NOT NULL,
   `xml` longtext,
   PRIMARY KEY (`qm_code`,'qm_version_code','datetime')
) TYPE=MyISAM;
-- In this table there is a row for each global metrics
--

CREATE TABLE `metrics` (  
    `id` int(11) NOT NULL auto_increment,  
    `name` varchar(255) NOT NULL default ",
    `explanation` varchar(255) NOT NULL default ",
    `comments` varchar(255) NOT NULL default ",
    `user` varchar(255) NOT NULL default ",
    `kind`  
    enum('BOOLEAN','INTEGER','REAL','STRING','DOMAIN','SET','TUPLE','FUNCTION','FORMULA','QUALITATIVE') NOT NULL default 'BOOLEAN',
    PRIMARY KEY (`id`),
    UNIQUE KEY `name` (`name`)  
) TYPE=MyISAM AUTO_INCREMENT=67 ;

-- -----------------------------------------------

-- In this table there is a row for each global metrics of type integer or real
--

CREATE TABLE `metrics_numeric` (  
    `metric_id` int(11) NOT NULL default '0',  
    `min_value` float default NULL,
    `max_value` float default NULL,
    PRIMARY KEY (`metric_id`)  
) TYPE=MyISAM;

-- -----------------------------------------------

-- In this table there is a row for each global metrics of type string
--

CREATE TABLE `metrics_string` (  
    `metric_id` int(11) NOT NULL default '0',  
    `special_value` varchar(255) NOT NULL default ",
    PRIMARY KEY (`metric_id`)  
) TYPE=MyISAM;

-- -----------------------------------------------
CREATE TABLE `metrics_domain` (  `metric_id` int(11) NOT NULL default '0',  `is_ordered` tinyint(1) NOT NULL default '0',  PRIMARY KEY (`metric_id`) ) TYPE=MyISAM;

CREATE TABLE `domain_metrics_values` (  `metric_id` int(11) NOT NULL default '0',  `num` int(11) NOT NULL default '0',  `value` varchar(255) NOT NULL default '',  PRIMARY KEY (`metric_id`, `num`) );

CREATE TABLE `metrics_set` (  `metric_id` int(11) NOT NULL default '0',  `element_metric` int(11) NOT NULL default '0',  PRIMARY KEY (`metric_id`) ) TYPE=MyISAM;

CREATE TABLE `metrics_tuple` (
CREATE TABLE `tuple_metrics_elements` (  `metric_id` int(11) NOT NULL default '0',  `num` int(11) NOT NULL default '0',  `element_name` varchar(255) NOT NULL default '',  PRIMARY KEY (`metric_id`, `num`)  ) TYPE=MyISAM;

CREATE TABLE `metrics_function` (  `metric_id` int(11) NOT NULL default '0',  `out_metrics` int(11) NOT NULL default '0',  PRIMARY KEY (`metric_id`)  ) TYPE=MyISAM;

CREATE TABLE `function_metrics_parameters` (  `metric_id` int(11) NOT NULL default '0',  `num` int(11) NOT NULL default '0',  `parameter_metrics` int(11) NOT NULL default '0',  PRIMARY KEY (`metric_id`, `num`)  ) TYPE=MyISAM;
CREATE TABLE `quality_patterns` (  `username` varchar(250) NOT NULL default ",",  `xml` longtext,  PRIMARY KEY (`username`) ) TYPE=MyISAM;

CREATE TABLE `glossary` (  `term` varchar(245) NOT NULL default ",",  `username` varchar(8) NOT NULL default ",",  `last_date` varchar(15) NOT NULL default ",",  `definition` longtext NOT NULL,  PRIMARY KEY (`term`),`username`) ) TYPE=MyISAM;

CREATE TABLE `global sources` (  `id` int(11) NOT NULL default 0,  `name` varchar(255) NOT NULL default ",",  `comments` varchar(255) NOT NULL default ",",  PRIMARY KEY (`id`) ) TYPE=MyISAM;

This table maintains the relationships among global metrics and global sources.
CREATE TABLE `metrics_sources` (
  `metric_id` int(11) NOT NULL default '0',
  `source_id` int(11) NOT NULL default '0',
  PRIMARY KEY (`metric_id`, `source_id`)
) TYPE=MyISAM;

3. DTD corresponding to the XML for a Quality Model

Quality Model XML Documents' structure and correctness is defined by its DTD (Document Type Declaration). DTD validation ensures element content, attributes and structure are correct.

The DOCTYPE declaration (DTD) for the Quality Model contains a name and a list of ELEMENT declarations.

```xml
<!DOCTYPE xml_model [ 
  <!-- ELEMENT DECLARATIONS -->
]>
```

The root Element of the Document is named after the DOCTYPE name.

ELEMENT declarations define the content for the element. The content can be defined in terms of other Elements (children), or as the text inside the Element. The `#PCDATA` (Parseable Character Data) content type is commonly used to specify that the Element contains text that is not markup, therefore the Element can not have children.

3.1. Common Element Declarations

In the Quality Model DTD there are several Elements that are referenced as childs of other Elements. These elements are mostly single-valued data such as name or explanation, or multi-valued as in a source list.

The `#PCDATA` (Parseable Character Data) content type is used to specify that the Element's content is the String text inside the Element.
Visualització de models de qualitat mitjançant una interfície web

<!ELEMENT name (#PCDATA)>
<!ELEMENT explanation (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT user (#PCDATA)>
   <!ATTLIST user id ID #REQUIRED>
<!ELEMENT sources (source*)>
   <!ELEMENT source (#PCDATA)>
      <!ATTLIST source id ID #REQUIRED>

Examples:

  <name> Suitability </name>

  <sources>
      <source id="1"> ISO9126-1 Standard </source>
      <source id="2"> Various Sources </source>
  </sources>

3.2. Root Element Declaration

The root Element of the Document has three child nodes. The first two of them are detailed below.

   <!ELEMENT xml_model (roots, key_classes, qmodel_classes)>

The roots Element contains the identifiers of the Quality Model Entities that represent the first level of decomposition of the Quality Model (characteristics).

   <!ELEMENT roots (root_item*)>
   <!ELEMENT root_item (#PCDATA)>
      <!ATTLIST root_item id IDREF #REQUIRED>

The key classes Element contains the new identifiers to assign in the Quality Model and is only used for Quality Model construction purposes.

   <!ELEMENT key_classes (source_counter, entity_counter,
        metric_counter, evaluation_counter, entity_req_counter,
        pattern_counter, relation_scale_counter,