Fraunhofer Einrichtung
Experimentelles
Software Engineering

REFSENO: A Representation Formalism for Software Engineering Ontologies

Report

Authors:
Carsten Tautz
Christiane Gresse von Wangenheim

IESE-Report No. 015.98/E
Version 1.1
October 20, 1998

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Table of Contents

1 Purpose of Report

The purpose of this report is to define the current state of the *r*epresentation *f*ormalism for *s*oftware *en*gineering *o*ntologies (REFSENO). The document begins with an introduction and overview of the state of the art (Chapter 2). Based on the state of the art, the notation (Chapter 3) and semantics (Chapter 4) of the basic constructs of REFSENO are defined. Experience gained so far with the application of REFSENO is documented (Chapter 5). The report ends with a summary and outlook (Chapter 6). The appendix illustrates REFSENO with an exemplary ontology in the domain of software measurement planning based on the GQM approach.

The definition of REFSENO and the example in the appendix are subject to change as more experience in the application of REFSENO (in particular for GQM planning) is gained. These changes will result in new versions of this report.

2 Introduction

»Reuse practice appears to exhibit considerable [improvement] potential, far more than other ongoing activities.« [ZS95, p. 167] The benefits of software reuse are manifold. Among them are improved productivity, improved reliability, better estimates, and faster time-to-market [SPDM94]. Traditionally, the emphasis has been on reusing code. However, reuse does not have to stop there. In order to continuously improve the software quality and productivity and transfer innovative software technologies into practice, several kinds of software-related knowledge can be reused [GB97, GRA+98, Hen97]. This includes

- products created by a software project (e.g., design document, code),
- processes (e.g., requirement analysis, inspection),
- quality and resource models (e.g., effort and reliability prediction models), or
- any lessons learned regarding the software process or products (e.g., using scenario-based inspections more faults are found than using checklist-based inspections in the organization Y).

As software engineering is a fairly young discipline, its technologies are not as mature as those of other engineering disciplines. On the other hand, requirements to software systems are steadily growing concerning their complexity, performance, etc. Therefore, the capturing and reuse of explicit software development know-how is essential for continuous improvement. What represents relevant software know-how differs among companies regarding their environmental context and their specific goals. Therefore, organization-specific software know-how which comprises the core of a mature software organization, has continuously to evolve based on experiences gathered in software projects. In that endeavor, software organizations require support in collecting experiences from their projects, packaging those experiences (e.g., build models from empirical data, formalize informal knowledge), and in validating and reusing experiences in future projects. The support of these tasks requires comprehensive learning information systems which are based on explicit knowledge representations [MBY97].

For the continuous build-up of software knowledge in an organization, the *experience factory* approach [BCR94] has been proven to be a successful solution [Rom96]. The experience factory organization complements the project organization by enabling the continuous learning on software development from examples of individual software projects and communication of software knowledge across the organization.

In order to enable the reuse of software engineering knowledge and the operationalization of the experience factory in practice, the domain of the software

engineering knowledge has to be modeled explicitly defining a structure of an experience base where the experiences are stored. Such a model must:

- define all artifact types to be stored in the experience base as well as the kinds of relationships between the artifacts,
- guide software engineers to specify the knowledge to be retrieved,
- facilitate the maintenance of an experience base,
- allow similarity-based retrieval.

This report presents a representation formalism to adequately model the structure of an experience base. It can be used to:

- model software engineering knowledge explicitly in a precise, consistent, and complete manner using alternate representations,
- conceptualize software engineering knowledge explicitly for various application domains and contexts resulting in conceptual models,
- validate conceptual models of software engineering knowledge,
- communicate conceptual models of software engineering knowledge,
- operationalize an experience base based on the conceptualization of software engineering knowledge.

In the following some basic terminology underlying the definition of the notation is introduced.

2.1 Basic Terms

Knowledge

In the context of this report

> *knowledge* is the set of all statements about the represented world that are believed to be true by the knowledge source and are really true. [Rei91]

This means that knowledge is always defined with respect to a knowledge source. According to the definition above, knowledge does *not* include statements about the represented world which:

- are not believed to be true by a knowledge source
- are believed to be true by a knowledge source, but are not really true

The latter case requires hypotheses and opinions to be marked as such. For example, the general statement »perspective-based reading is better than ad-hoc reading« alone is (usually) not considered as knowledge whereas the statements »perspective-based reading can be better than ad-hoc reading« and »perspective-based reading has been more effective than ad-hoc reading in projects X, Y, and Z« are considered as knowledge, because there is empirical data proving the statements to be correct. This means that the latter two state-

ments are really true and not just believed to be true by a single knowledge source.

Representation

A *representation* is a set of representation constructs together with the *interpretation* how the constructs map onto the characteristics of the represented world. [Rei91]

A representation stands as a substitute for a set of facts – called represented world. Therefore, it is a model of the world. A representation fulfils the following properties [Rei91]:

1 A representation captures some characteristics (= regularities of a general nature as well as properties and relationships between objects) of the represented world.

2 A representation does not necessarily capture (usually never) all characteristics of the represented world.

3 Not all characteristics of a representation stand necessarily for some characteristics of the represented world.

Representation formalism

A *representation formalism* is a notation for specifying representations plus a definition of the meaning of the notation (cf. »knowledge representation model« [Rei91]).

Usually, a representation formalism is defined by primitives with a defined meaning from which representations can be assembled. In this sense, all programming languages can be interpreted as a representation formalism. The actual program code is then the representation of the program (i.e., a representation of a process to be performed by the operating system of a computer).

Conceptualization

A conceptualization is a special type of representation:

A *conceptualization* is a set of concepts, instances, and other entities that are assumed to exist in some area of interest and the relationships that hold among them. [Gru93]

In contrast to the definition of the term »representation«, this definition assumes the existence of primitives such as concepts[1], instances, and relationships.

1 Concepts can be seen in analogy to classes in object-oriented modeling.

Ontology

An *ontology* is the explicit specification of a conceptualization [Gru93].

All programs are based on a conceptualization of a portion of a world they are developed for. For example, a tool for requirements elicitation may provide means to capture different types of requirements such as functional and non-functional requirements as well as predefined design decisions. This can be done without explicitly specifying how these terms are related and what properties they have. Only if such an implicit conceptualization is made explicit, it is called an ontology.

The term ontology can be seen in analogy to the term data model used in the area of database management systems. The (up to now) differing focuses in the areas of databases and knowledge representation imply, however, different representation constructs and types of operations defined upon the structures. [Rei91, p. 13]. In the area of databases, simple but efficiently implemented constructs prevail, while more complex constructs are typical for the area of knowledge representation. Another difference is that in the database world change operations are strongly emphasized, while inference processes on static structures are emphasized by the knowledge representation community. This is a knowingly polarized characterization since the two areas have been approaching each other in recent years.

Epistemistic primitives

An *epistemistic primitive* stands for a class of alike characteristics. Its level of generality does not limit its occurrence to a single area of discourse (or class of discourse areas). [Rei91]

As can be deduced from the definition, an epistemistic primitive is domain-independent. Typical examples for epistemistic primitives are the elements of a conceptualization (concepts, instances, …).

Standard vocabulary

An element of a *standard vocabulary* can be used to represent characteristics which are

- specific enough not to occur in every discourse area and
- general enough to describe several more complex characteristics

Thus, a standard vocabulary is domain-specific.

2.2 Modeling Levels

Knowledge level

Symbol level

Knowledge can be represented on different levels of abstraction. In this report we differentiate between *knowledge levels* and the *symbol level*. On a knowledge level, knowledge contents are visible, but not the internal structures in which the contents manifest themselves. These internal structures are visible on the symbol level. Thus, the symbol level is defined by the chosen implementa-

tion while the knowledge levels are implementation-independent. In other words, a knowledge level describes »what« to represent while the symbol level describes »how« to represent it.

For the purpose of this report, we distinguish three knowledge levels: the epistemological level, the conceptual level, and the linguistic level. [Rei91]

The basic characteristic of all knowledge levels is that descriptions on a particular knowledge level can be specified using the constructs of the next lower level. In the following, the three knowledge levels will be described in detail (Figure 1).

Epistemological level

1 The *epistemological level* defines the epistemistic primitives such as concepts, attributes, relationships, etc. Thus, the epistemological level is domain-independent.

Conceptual level

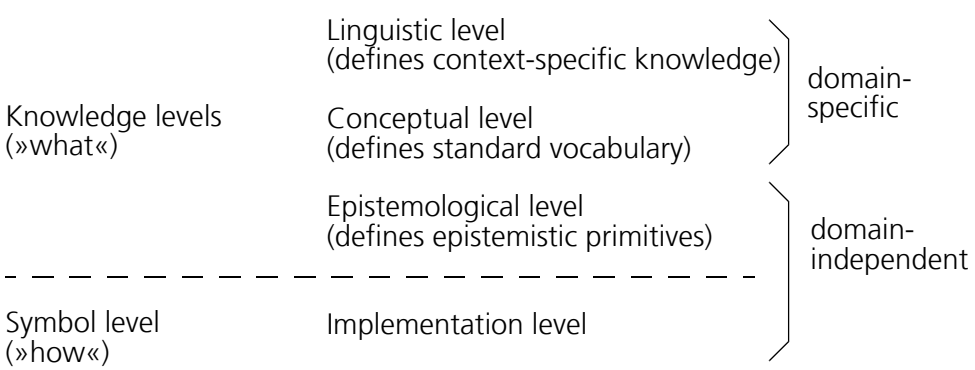
2 The *conceptual level* defines the standard vocabulary. It is domain-specific. Exemplary constructs of this level (for the software engineering domain) are process models, measurement plans, code modules, lessons learned, etc.

As an explicit specification of a conceptualization, an ontology is always defined on this level. Thus, an ontology can be defined using epistemistic primitives.

Linguistic level

3 Finally, the *linguistic level* defines concrete instances of the constructs defined on the conceptual level. It is domain- and context-specific. An exemplary construct on this level (for a particular software development organization) is a concrete measurement plan for measuring the effort of project X at company Y.

Figure 1: Different representation levels



2.3 The Representation Formalism REFSENO

In this paper, epistemistic primitives are defined in the form of a *r*epresentation *f*ormalism for *s*oftware *en*gineering *o*ntologies (REFSENO). Ontologies defined using REFSENO can be easily tailored to company-specific needs as they are not hard-wired into an implementation (only the epistemistic primitives are) and fulfil all requirements of a structure model listed on page 3.

Thus, software engineering knowledge is modeled by defining a software engineering ontology using our representation formalism REFSENO (domain knowledge) and by instances of the concepts defined in the ontology (context-specific knowledge). The next chapter will present the definition of the epistemistic primitives of REFSENO. The primitives draw from ideas from several areas such as database mechanisms (e.g., relationships between concepts and implied consistency rules) [Che76], case-based reasoning mechanisms (e.g., similarity-based retrieval with incomplete information) [Alt97], and knowledge-based mechanisms (e.g., inference rules) [Rei91]. In addition, the representation formalism is object-centered similar to the meta modeling in UML [Cor97].

It should be emphasized that none of the approaches alone would be sufficient for implementing a software engineering experience base. Except for case-based reasoning, none of the approaches consider similarity-based retrieval in a detail necessary. On the other hand, case-based reasoning lacks the ability to cope with relationships between concepts to the degree needed. However, in our approach we integrated representation constructs from all approaches to form a new consistent representation formalism for software engineering ontologies. The result is based on several years of experience in structuring experience bases [Gäß95, Stu95, Tau93, TA97].

It is important to realize that the ontologies defined using REFSENO serve the purpose of software knowledge management and not as the basis for the implementation of intelligent assistants in the sense of Mylopoulos et al. [MBY97]. Depending on the purpose, the employed knowledge representations and bases will differ (see Table 1). This means that intelligent assistants, which are used to edit software engineering artifacts, should use different knowledge representations and knowledge bases than the experience base.

Table 1: Software knowledge management vs. intelligent assistants (according to [MBY97])

Criterion	Software knowledge management	Intelligent assistant
Type of representation used	mix of formal, declarative and informal	formal, often procedural
Type of knowledge captured	descriptive information about artifact and information about environment of artifact	environment is less important; domain-independent heuristics
Coverage of software knowledge base	Quite broad	Narrow, specific to the task the assistant is intended to perform

Criterion	Software knowledge management	Intelligent assistant
Completeness of software knowledge base	Useful even if it is incomplete	Sufficiently complete to support inferences required for performance of task
Criteria for success of knowledge capture activity	1. time saved in chasing for information 2. accuracy and completeness of knowledge	how well the knowledge-based system performs intended task

In the literature there exist various representation formalisms for the purpose of software knowledge management. However, they either provide no means for similarity-based retrieval[1] (e.g., RLF [SWT89], LaSSIE [DBSB91], and ESTAME [OB92]) or they are restricted to one type of artifact (e.g., faceted classification [PDF87] and the Reusable Software Library [BAB+87]). Generic approaches like information retrieval can in principle be applied to all kinds of artifacts but are not adequate, because it is difficult to determine the relationships between the various types of artifacts [SM83]. Even worse, not all information necessary for retrieving software engineering artifacts can be found in the artifacts themselves [BR91]. Thus, indexing of software engineering knowledge cannot be totally automated, which takes away one of the biggest advantages information retrieval has to offer.

One of the few formalisms that allow both, the storage of various types of artifacts and similarity-based retrieval, is the Extensible Description Formalism (EDF) [Ost92]. Yet, REFSENO uses a more rigid type system, allowing more user guidance for retrieval and maintenance of software engineering knowledge. Moreover, EDF does not provide a clear distinction between the conceptual and linguistic level as REFSENO does. The clear distinction between the two levels has the significant advantage that experts can provide their (linguistic) knowledge in form of example cases guided by conceptual knowledge pre-defined by knowledge engineers. On the other hand, knowledge engineers only have to focus on the elicitation of the ontology, but not of context-specific knowledge as necessary for conventional knowledge-based systems [FG90].

As linguistic knowledge tends to be volatile, this is important because it reduces considerably the effort for acquiring and storing such knowledge. In contrast, knowledge on the conceptual level is typically much more stable, but requires more effort to be captured. However, we aim at providing *basic ontologies* that can be tailored to organization-specific needs with a comparably low amount of effort. When all basic ontologies are integrated, they model the complete struc-

1 As no two software development projects are alike, it is unlikely that a candidate matching exactly the requirements is available in the experience base. Thus, similarity-based retrieval is vital for the success of an experience factory [BR91].

ture of an experience base. One such basic ontology is the GQM planning ontology, which we will use to illustrate the representation formalism.

2.4 Structure of this Report

The next chapter defines the epistemistic primitives needed for describing software engineering ontologies. The primitives are exemplified using excerpts from an ontology for GQM planning artifacts. Chapter 4 completes this representation formalism on the epistemological level by defining the operations that are allowed on ontologies defined using the epistemistic primitives. Chapter 5 gives first lessons learned from applying the representation formalism for software engineering ontologies (short REFSENO). A summary and outlook can be found in Chapter 6. Finally, Appendix lists the complete ontology for GQM planning artifacts using the representation formalism defined in this report.

3 Notation

This chapter defines the notation used for the epistemistic primitives needed to define a software engineering ontology. Each epistemistic primitive of REFSENO is defined by:

- **Synonyms.** Under synonyms terms are listed which are used with other modeling approaches or representation formalisms. The listed synonyms have a similar meaning to the described epistemistic primitive of REFSENO.
- **Definition.** This section gives a terse definition of the epistemistic primitive by listing its components. Components may be epistemistic primitives themselves. The definition specifies the *abstract syntax*, a term known from programming languages [ASU86]. In addition, some identifiers, functions, and predicates are defined which are used in subsequent descriptions.
- **Description.** This section explains the components using narrative text, that is, it specifies the semantics of the abstract syntax.
- **Representation.** This section defines how the epistemistic primitive is represented in REFSENO (REFSENO uses a tabular representation mixed with formulas). Thus, the representation specifies the *concrete syntax* [ASU86].
- **Example.** In order to illustrate the representation of the epistemistic primitive, this section gives an example using excerpts from the ontology for GQM Planning Artifacts (see Appendix).
- **Alternate representation.** For reasons of comprehensibility, it is sometimes useful to represent the same issues in different ways (e.g., using a tabular or graphical representation). This section presents alternate ways of representing epistemistic primitives.

3.1 Concept

Concepts model software engineering entities (e.g., a GQM plan, a process model, or a development product), or are needed for modeling purposes.

3.1.1 Synonyms

Artifact type, class (UML), case model (CBR), entity, frame

3.1.2 Definition

A concept is a 10-tuple (*name*, *extension*, *intension*, sim_{artif}, $sim_{I/F}$, sim_{ctxt}, *assertion, precondition, description*, *purpose*, *intended users*).

An intension is a 3-tuple (*artif*, *I/F*, *ctxt*). Each component consists of terminal and nonterminal concept attributes.

The functions extension(concept), assert(concept), and precond(concept) return the *extension*, *assertion*, and *precondition* of *concept* respectively. The functions $\text{intension}_{\text{artif}}(\text{concept})$, $\text{intension}_{\text{I/F}}(\text{concept})$, and $\text{intension}_{\text{ctxt}}(\text{concept})$ return the respective components of the intension. The function intension(concept) is defined as:

$$\text{intension}(c) = \text{intension}_{\text{artif}}(c) \cup \text{intension}_{\text{I/F}}(c) \cup \text{intension}_{\text{ctxt}}(c)$$

$\text{sim}_{\text{artif}}(c)$, $\text{sim}_{\text{I/F}}(c)$, and $\text{sim}_{\text{ctxt}}(c)$ denote the similarity functions sim_{artif}, $sim_{I/F}$, sim_{ctxt} associated with the concept *c*:

$$\text{sim}_{\text{artif}}(c), \text{sim}_{\text{I/F}}(c), \text{sim}_{\text{ctxt}}(c) : \text{extension}(c) \times \text{extension}(c) \rightarrow [0;1]$$

3.1.3 Description

*Name*The name is used for reference purposes. All concepts used in an ontology have unique names.

Extension The set of all instances belonging to the concept. The extension of the ontology is restricted to those instances specified as part of the ontology (Section 3.6). Since context-specific knowledge is also represented as instances (Chapter 4), the notion of *extension* is extended to include the context-specific knowledge as well. It should be clear, that the ontology's *extension* (extension in the restricted sense) does not change with the insertion or deletion of instances. In the rest of the report, the term »extension« will be used with its extended notion. By the same token, the function extension(concept) will return the union of the ontology's instances and the context-specific knowledge.

Intension The set of all attributes an instance which belongs to the concept must exhibit (i.e., all instances of the *extension* are characterized using the same attributes). There are two kinds of attributes: terminal (Section 3.2) and nonterminal attributes (Section 3.4). Furthermore, each attribute belongs to one of three layers: artifact, interface, or context [BR91]. Attributes of artifact layer characterize the instance itself (e.g., author and programming language for code modules). Attributes of the interface layer characterize how a particular instance can be integrated into the surrounding system (e.g., parameters and global variables for code modules). Finally, attributes of the context layer characterize the environment, in which the instance has been applied (e.g., application domain for code modules). The context layer also contains attributes describing the quality of the instance in the specified environment (e.g., reliability in context of the specified application domain for code modules). The three components of the *intension* mirror these layers.

sim_{artif}
$sim_{I/F}$
sim_{ctxt}

The similarity functions [Alt97] associated with the concept. They compute the similarity between two instances of the *extension* based on the *intension* of the concept. More precisely $sim_{artif}(c)$, $sim_{I/F}(c)$, and $sim_{ctxt}(c)$ are based on $intension_{artif}(c)$, $intension_{I/F}(c)$, and $intension_{ctxt}(c)$ respectively. The values of the similarity functions[1] are combined to a single similarity value as follows:

$$sim(c)(i, i') = w_{artif} \cdot sim_{artif}(c)(i, i') + w_{I/F} \cdot sim_{I/F}(c)(i, i') + w_{ctxt} \cdot sim_{ctxt}(c)(i, i')$$

where w_{artif}, $w_{I/F}$, and w_{ctxt} are weights with which the similarity function can be adjusted to the needs and/or skills of the user (Chapter 4). The sum of the weights is always 1.

All similarity values are in the range between 0 (denoting total dissimilarity) and 1 (denoting total similarity, i.e., equivalence).

Assertion

A condition, expressed as a formula (Section 3.7), which all instances of the *extension* must fulfil.

Precondition

A condition, expressed as a formula (Section 3.7), which must be fulfilled before instances are inserted or changed.

Description

A narrative text defining the software engineering entity.

Purpose

In general, only a portion of the real world is represented by an ontology. Since every representation serves some purpose, this purpose shall be stated explicitly. For a concept, its purpose may be viewed as a justification for the existence of the concept in the ontology. There are two types of concepts:

1 Concepts resembling a software engineering entity. For this type of concepts, usage scenario(s) shall be stated showing for what type of »real-world queries« the concept is used.

2 Concepts introduced for modeling reasons. For instance, sometimes the intension of two concepts overlap. In such a case, a third concept can be introduced capturing the intersection of both intensions even though there might not be a software engineering entity corresponding to this new concept.

1 The concept's similarity functions are of a global nature because they are based on the local similarity functions of the concept's attributes.

Intended users As mentioned for *purpose*, a query is started for a particular purpose. Different users have different tasks. For each task, different information needs exist. While *purpose* describes the tasks for which a concept is necessary, the *intended users* describe who is expected to perform the task. *Intended users* are described by their roles.

3.1.4 Representation

Concepts, their intensions, and their extensions are represented using separate tables.

3.1.4.1 Concept Glossary

The concept glossary lists alphabetically all concepts of an ontology. One row of the concept glossary corresponds to one concept. The columns are labeled »Name«, »Description«, »Purpose«, and »Intended users« denoting the respective components of the concept definition.

3.1.4.2 Concept Attribute Table

The *intension* of a concept is represented using a concept attribute table. The representation is explained in the sections about terminal concept attributes (Section 3.2.4) and nonterminal concept attributes (Section 3.4.4). The similarity function is presented by a mathematical formula (Section 3.7.4) or by the term »standard«. The standard similarity functions[1] for a concept *c* are defined as:

$$\forall j \in \{\text{artif,IVF,ctxt}\}: \text{sim}_j(c)(i,q) = \left(\sum_{at \in \text{intension}_j(c)} \begin{cases} 0 \Leftrightarrow i.at \sim q.at = \text{undefined} \\ p(at) \quad \text{otherwise} \end{cases} \right) = 0 \,?\, 1 :$$

$$\left(\sum_{at \in \text{intension}_j(c)} \begin{cases} 0 \Leftrightarrow i.at \sim q.at = \text{undefined} \\ p(at) \quad \text{otherwise} \end{cases} \right)^{-1}$$

$$\times \sum_{at \in \text{intension}_j(c)} \begin{cases} 0 \Leftrightarrow i.at \sim q.at = \text{undefined} \\ p(at) \times (i.at \sim q.at) \quad \text{otherwise} \end{cases}$$

1 Informally, the standard similarity functions compute the weighted sum of the local similarities (footnote on page 12) of all attributes (Section 3.2 and Section 3.4) whose values are defined in both instances (Section 3.6). An attribute value is defined if it is not a special value (»n/a«, »undefined«, or »unknown«). If none of the attributes have defined values in both instances, the functions return 1 (first line). Otherwise the similarity is computed (line 3) and normalized (line 2).

Since the similarity functions are typically based on the similarity of the concept's attributes, they are represented with the table of concept attributes. By the same token assertions and preconditions are represented with the table of concept attributes because their definition is based on the concept's *intension*, too (Table 3).

3.1.4.3 Instance Table

Instances of the concept's *extension* are represented using instance tables. The representation is explained in the section about instances (Section 3.6.4).

3.1.5 Example

Table 2: Excerpt of a concept glossary

Name	Description	Purpose	Intended user(s)
Context Characterization	describes the context of a GQM measurement program concerning its organizational, project-specific and measurement-specific environment	Modeling: explicitly states the context from which the knowledge originates	Experience engineer
GQM Measure	A GQM measure is an operational definition of an attribute. The data collected according to the measures are used by a model to answer the question in the GQM plan.	Usage scenario: support development of GQM plan by supplying adequate measures for a model	Quality assurance personnel
GQM Plan	A GQM plan contains information necessary to motivate and define measures and interpret measurement data. Elementary components are GQM goal, questions, models, and measures.	Modeling: structures GQM products	Quality assurance personnel
…	…	…	…

3.1.6 Alternate Representation

none

3.2 Terminal Concept Attributes

The *intension* of a concept is a set of terminal and nonterminal concept attributes. Here, a terminal concept attribute is viewed as an epistemistic primitive.

Terminal concept attributes model how software engineering entities are specified for storage and retrieval.

3.2.1 Synonyms

Dimension, data element, feature, property, terminal slot

3.2.2 Definition

A terminal concept attribute is a 9-tuple (*name*, *description*, *cardinality*, *type*, *default value*, *mandatory*, *value inference*, *inferred attributes*, *standard weight*).

The predicate mand(attr) is true iff *attr* is mandatory where *attr* is the *name* of the attribute. t_{attr} denotes the *type* of a terminal concept attribute *attr*.

3.2.3 Description

Name The name is used for reference purposes. All concept attributes (both terminal and nonterminal) of one concept's *intension* have unique names.

Description A narrative text defining the meaning of the attribute.

Cardinality A range specifying the minimal and maximal number of values the attribute may have. If the cardinality is unequal 1, the attribute values are specified as a list where a cardinality of 0 denotes an empty list.[1]

Type Each terminal concept attribute is typed. The type is an epistemistic primitive described in Section 3.3.

Default value The *default value* concerns the insertion of new instances (Section 3.6 and Section 4.2). If the user entering the new instance does not specify a value for this attribute, the default value is used instead.

Mandatory This component also concerns the insertion of new instances. It specifies whether an attribute value of an instance has to be specified (i.e., the attribute value may only be undefined if the attribute is not mandatory).

Value inference This component defines how to calculate the attribute value automatically (if possible) based on other attributes' values (of an instance of this concept and of semantically related instances of this instance).

1 Some functions (see formulas in Section 3.7) interpret lists as sets. However, if no *value inference* is defined for a particular attribute, the sequence of the elements of its value will be the sequence in which they are specified (i.e., the sequence remains unchanged).

Inferred attributes This component lists all attributes whose value is inferred using a value of this attribute.[1]

Standard weight The *standard weight* may be used by the similarity functions of the concept this attribute belongs to. It is a non-negative real number.[2] A standard weight of 0 denotes an attribute whose value will not be used for querying (i.e., this attribute cannot be specified for a query).

3.2.4 Representation

Concept attribute table A terminal concept attribute is represented using the concept attribute table. The concept attribute table is concept-specific and contains one row for every attribute. The columns are labeled as follows: »Layer«, »Name«, »Description«, »Cardinality«, »Type«, »Default value«, »Mandatory«, »Value inference«, »To infer«, and »Standard weight«. Attributes are sorted according to the layers they belong to. Possible values for the first column are »artif«, »I/F«, and »ctxt«. The rest of the columns represent the respective components of a terminal attribute where »To infer« contains the *inferred attributes*.

The cardinality is specified using the UML notation, that is, a range is denoted as *low*..*high* where *low* stands for the lower bound (≥ 0) and *high* ($\geq$ *low*) for the upper bound. Instead of a number, an asterisk (»*«) can be used which stands for infinity. If *low* = *high*, the range may be abbreviated to a single number (e.g., »1« stands for »1..1«).

A *default value* is a value from the value range of the type (Section 3.3.4). An undefined entry for a *default value* is denoted with »–«.

If the attribute value is mandatory, the entry in the mandatory column is »yes«, otherwise »no« meaning that the attribute value may be undefined.

Value inferences are represented using a formula (Section 3.7.4). A »–« denotes that the value cannot be computed.

Inferred attributes are represented by pairs, separated by commas, of the form [*concept*]:[*attribute*] where *concept* and *attribute* stand for the names of the respective constructs.

1 There is a mutual dependence between *value inferences* and *inferred attributes*. For every attribute used in a *value inference*, the *inferred attributes* of the used attribute must include the attribute whose value is inferred. Thus, *inferred attributes* can be derived from the *value inferences* automatically.

2 For the standard similarity functions of concepts, this weight defines the importance of this attribute regarding the global similarity.

In addition, the similarity functions, the assertion, the precondition, and the super concept (Section 3.5) are defined near the concept attribute table. Attributes of the super concept are inherited by the concept which is specified through the concept attribute table. Thus, the *intension* of the concept does not only include the attributes specified in this concept attribute table, but also those specified in the attribute tables of its super concept(s). The root concept is denoted by »CONCEPT«.

3.2.5 Example

Table 3: Terminal concept attributes of the concept »Measurement Characterization«

Concept: Measurement Characterization Super concept: CONCEPT									
Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
I/F	constraints	on the measurement program (e.g., fixed amount of effort assigned to the measurement program)	0..*	Text	-	no	-	-	0

Concept: Measurement Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt	measurement integrated	measurement programs regularly established accompanying software development and maintenance	1	Boolean	false	no	-	-	1
	experiences with measurement	specifies if no experiences are available or either positive or negative experiences have been made with measurement in the past	1	Measurement-Knowledge	"not available"	no	-	-	1
	core measures	specifies if a set of core measures is collected in each project in case measurement programs are performed regularly	1	Boolean	-	no	-	-	1
	attitude	of management and project personnel concerning software quality improvement in general	1	Attitude	-	no	-	-	1
	effort	on the planning and execution of the measurement program in person-months	1	EffortPM	-	no	-	-	1
	duration	of the measurement program in calendar months	1	DurationM	-	no	-	-	1
	duration of data collection	period in calendar months	1	DurationM	-	no	-	-	1

Concept: Measurement Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt	frequency of feedback sessions	during the execution phase per calendar month	1	FreqM	-	no	-	-	1
	training	describes training(s) of the participants regarding the GQM approach and its application which took place during the planning phase	1	Text	-	no	-	-	0
	number of goals	size of the measurement program in terms of number of GQM goals	1	Cardinal	-	no	card(union(filter([GQM Product Experience], [context].[measurement experience]).[gqm plan].[gqm goal]))	-	1
	number of questions	size of the measurement program in terms of number of questions in the GQM plans	1	Cardinal	-	no	card(union(union(filter([GQM Product Experience], [context].[measurement experience]).[gqm plan]).[gqm question]))	-	1
	number of measures	size of the measurement program in terms of number of measures in the GQM plans	1	Cardinal	-	no	card(union(union(filter([GQM Product Experience], [context].[measurement experience]).[gqm plan]).[gqm measure]))	-	1

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

An example for a more complex precondition can be found in the concept attribute table for »tTaxonomyNode« (Appendix):

precond: pos(1, [RootLevel].[VRange].[Symbol], [Symbol].[Symbol]) = (

An example for a more complex assertion can be found in the concept attribute table for »TCardinal« (Appendix):

assertion: [LowerBound] ≤ [UpperBound]

3.2.6 Alternate Representation

The *value inference* can also be represented graphically as a tree structure. The root is the attribute of the value inferred. The leafs are the attributes whose value is used for the inference. In contrast to the formula in the table, the attribute names are complemented by the names of the concepts they belong to. All other nodes denote the operators used. Figure 2 shows an example.

Interpretation

In order to understand the interpretation of the figure, the reader should have read the sections on nonterminal attributes (Section 3.4) and formulas (Section 3.7). It may be advantageous to come back here after having read the rest of this chapter.

The figure has to be read from bottom to top. In Table 3, »[context]« in the »value inference« column refers to the nonterminal attribute »context« of the concept the value inference is part of (i.e., »Measurement Characterization«). This is shown in lower left corner of the figure. The nonterminal attribute »context« has the destination concept »Context Characterization« and the cardinality 1 (Table 7 on page 31). Thus, »[context]« returns exactly one instance of »Context Characterization« and ».[measurement experience]« refers to the attribute »measurement experience« of the concept »Context Characterization«. This is denoted in the figure by »[Context Characterization]: [measurement experience]«. The destination concept of the nonterminal attribute »measurement experience« is »Measurement Experience«. Since the cardinality of »measurement experience« is »0..*« a list of instances of »Measurement Experience« is returned (Figure 5 on page 32). However, to compute the number of questions in the GQM plan, we need to access the attribute »gqm plan« of the concept »GQM Product Experience« which is a specialization of »Measurement Experience«. Therefore, the list of instances is filtered (function »filter«). All instances which are not instances of »GQM Product Experience« are removed from the list. The result of the function is a list of instances of »GQM Product Experience«. The nonterminal »gqm plan« of »GQM Product Experience« returns the set of GQM plans which belong to an instance of the »GQM Product Experience«. Thus, the result of »filter(…).[gqm plan]« is a list of a lists of instances of a GQM plan. The function »union« takes this list of lists and merges the GQM plans into a single list of GQM plans by removing all duplicates. For each element of the list, the set of GQM questions is accessed through the attribute »gqm question« (denoted by ».[gqm question]« in the formula and »[GQM Plan]: [gqm question]« in the figure). The result is a list of lists of GQM questions. This is again merged into a single list by the function »union«. Finally, the function »card« counts the number of elements of the resulting list.

3.3 Types of Terminal Concept Attributes

All terminal concept attributes are typed. Here, a type is viewed as an epistemistic primitive.

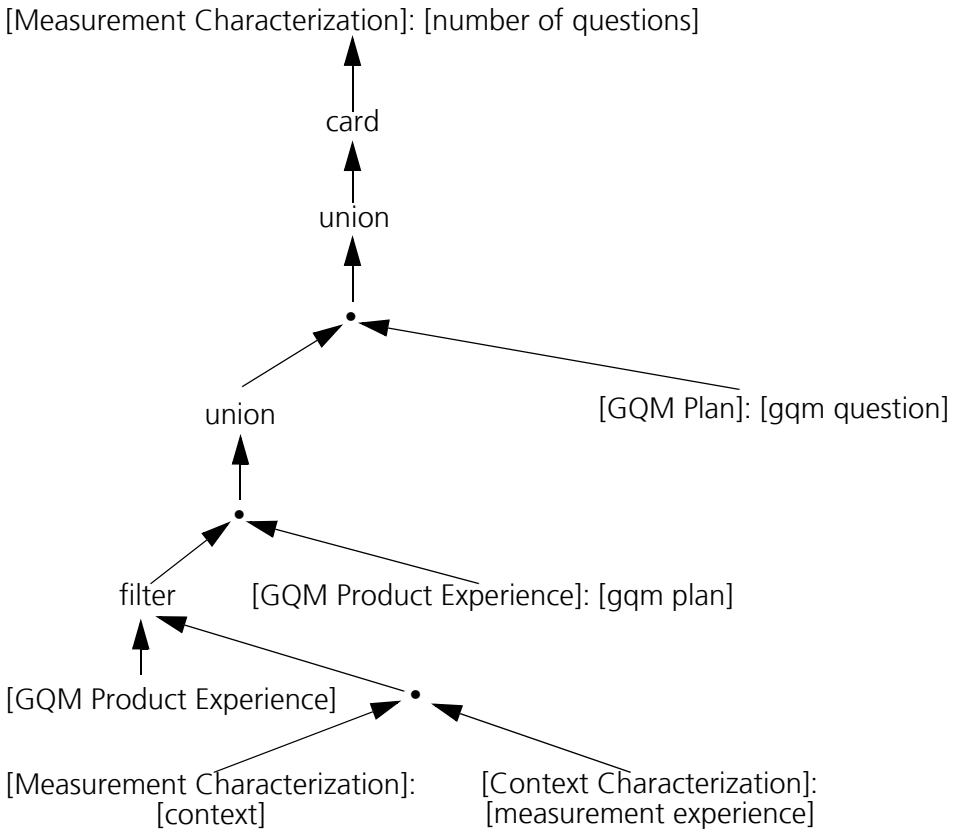


Figure 2: Example for the graphical representation of a value inference

Types model qualities of software engineering entities such as lines of code and efficiency, or they are used to specify entities further, e.g., a type may specify possible programming languages. Programming languages may be a terminal concept attribute for concepts such as a code module (specifying in which language the module is written) or a project (specifying what languages were used in the project).

3.3.1 Synonyms

none

3.3.2 Definition

A type is a 5-tuple (*name*, *supertype*, *value range*, *unit of measure*, *sim*).

The function range(type) returns the value range of *type*.

sim_{type} denotes the similarity function *sim* associated with *type*:

$$sim_{type}: range(type) \times range(type) \rightarrow [0;1]$$

$super_{type}$ denotes the supertype of *type*.

3.3.3 Description

Name The name is used for reference purposes. All types of an ontology have unique names.

Supertype This component specifies the type's *supertype*. Types differ from their supertype in one or more of the following ways:

- **Value range.** For symbol types (see predefined types below) the value range of the type includes the value range of its supertype:

  $$range(type) \supseteq range(super_{type})$$

  The range of the predefined types »text« and »identifier« cannot be changed. For all other types the range of the supertype includes value range of the type:

  $$range(type) \subseteq range(super_{type})$$

- **Unit of measure.**
- **Sim.**

Value range The value range specifies the possible values for all attributes of this type. In addition to the values specified the following special values are allowed: »undefined«, »unknown«, »n/a« (not applicable). »undefined« means that the value is currently not specified, but will be specified later; »unknown« means that the value is not known and will not be specified later; »n/a« means that the attribute is not applicable for the instance.

Unit of measure This component specifies the measurement unit for numerical types (»integer« or »real«). Otherwise this component is not applicable.

Sim The similarity function associated with the type. It computes the similarity between two possible values of this type. The similarity function returns a real number between 0 (denoting total dissimilarity) and 1 (denoting total similarity, i.e., equivalence).

The similarity function is extended to terminal concept attributes as follows. Let *i* and *q* be two attribute values[1]. Attribute values can be interpreted as sets (the

minimal and maximal number of elements is defined by the cardinality). Then the similarity value is computed in the following way:

$$sim_{attr}(i, q)$$

$$= \begin{cases} \sum_{e_2 \in q} \frac{\max\{sim_{t_{attr}}(e_1, e_2) \mid e_1 \in i\}}{card(q)} & \Leftrightarrow card(q) > 0 \wedge card(i) > 0 \\ 0 & \Leftrightarrow card(i) = 0 \\ 1 & \text{otherwise} \end{cases}$$

This allows the computation of the similarity between two attribute values. Informally, the similarity function considers for each element in the second argument the best match in the first argument and sums up the similarity values of the pairs. Finally the similarity value for the attribute values is normalized resulting in a similarity value in the range [0; 1].

3.3.3.1 Predefined Types

Ordered types

The following types are predefined. From each of these types, subtypes may be derived (for an explanation of the representation see the next section). The following types are called *ordered types*: Integer, Real, Text (alphabetically ordered according to ASCII), Identifier (alphabetically ordered according to ASCII), Date (ordered according to year, month, and day), Time (ordered according to hour, minute, and second), Timestamp (ordered according to year, month, day, hour, minute, second), and OrderedSymbol. For these types, a total ordering is defined.

Table 4: Predefined types

Name	Super-type	Value range	Unit of Measure	Similarity
Boolean	TYPE	true, false	n/a	$sim(i, q) = \begin{cases} 1 \Leftrightarrow i = q \\ 0 \quad \text{otherwise} \end{cases}$

1 Generally, the similarity is defined between two attribute values. However, similarity functions need not to be symmetrical (i.e., sim(i, q) ≠ sim(q, i)). In practice, similarity functions are used for the similarity-based retrieval (cf. Section 4.1). Here, a similarity value is computed between an instance stored in the experience base and a query which is also specified in form of an instance. In the similarity functions, the arguments *i* and *q* refer to the instance and the query respectively (or their attributes in case of type similarity functions).

Name	Super-type	Value range	Unit of Measure	Similarity
Integer	TYPE	*..*	n/a	Let *minvalue()* and *maxvalue()* the lower and upper bound of the value range respectively. Then, the similarity is computed as follows: $$sim(i, q) = 1 - \frac{\lvert i - q \rvert}{maxvalue() - minvalue()}$$
Real	TYPE	*..*	n/a	Let *minvalue()* and *maxvalue()* the lower and upper bound of the value range respectively. Then, the similarity is computed as follows: $$sim(i, q) = 1 - \frac{\lvert i - q \rvert}{maxvalue() - minvalue()}$$
Text	TYPE	Any text	n/a	$$sim(i, q) = \begin{cases} 1 \Leftrightarrow i = q \\ 0 \quad otherwise \end{cases}$$
Identifier	Text	Any text consisting of letters (»a«..»z«, »A«..»Z«), digits (»0«..»9«), »-«, and »_«.	n/a	$$sim(i, q) = \begin{cases} 1 \Leftrightarrow i = q \\ 0 \quad otherwise \end{cases}$$
Date	TYPE	(1..31; 1..12; 0..*) [only valid dates]	n/a	Let *daydiff(d,d')* be the number of days the dates *d* and *d'* are apart and *date(1,1,1900)* stand for Jan 1, 1900. Then, the similarity is computed as follows: $$sim(i, q) = 1 - \frac{\lvert daydiff(i, q) \rvert}{daydiff(i > q\,?\ i : q, date(1, 1, 1900))}$$ where $i > date(1,1,1900)$ and $q > date(1, 1, 1900)$
Time	TYPE	(0..*; 0..59; 0..59)	n/a	Let *timediff(t,t')* be the number of seconds the times *t* and *t'* are apart and *time(0,0,0)* stand for 00:00:00. Then, the similarity is computed as follows: $$sim(i, q) = \left(i = q\,?\ 1 : 1 - \frac{\lvert timediff(i, q) \rvert}{timediff(i > q\,?\ i : q, time(0, 0, 0))}\right)$$

Name	Super-type	Value range	Unit of Measure	Similarity
Timestamp	TYPE	(1..31; 1..12; 0..*; 0..*; 0..59; 0..59) [the first three components constitute a valid date]	n/a	Let *tsdiff(t,t')* be the number of seconds the timestamps *t* and *t'* are apart and *ts(1,1,1900,0,0,0)* stand for Jan 1, 1900, 0:00:00. Then, the similarity is computed as follows: $sim(i, q) = \left(i = q ? 1 : 1 - \frac{\lvert tsdiff(i, q)\rvert}{tsdiff(i > q ? i : q, ts(1, 1, 1900, 0, 0, 0))}\right)$ where $i > ts(1, 1, 1900, 0, 0, 0)$ and $q > ts(1, 1, 1900, 0, 0, 0)$
Symbol	TYPE	{}	n/a	$sim(i, q) = \begin{cases} 1 \Leftrightarrow i = q \\ 0 \quad \text{otherwise} \end{cases}$
OrderedSymbol	Symbol	{}	n/a	Each symbol in the ordered set is assigned a position. Let *pos(symbol)* be an integer value representing the position of *symbol* and *rangesize()* the number of symbols in the range. Then, the similarity is computed as follows: $sim(i, q) = 1 - \frac{\lvert pos(i) - pos(q)\rvert}{rangesize()}$
TaxonomySymbol	Symbol	"ROOT"	n/a	A taxonomy is represented as a tree. Let the function d(*n)* be the depth. of node *n* where d("ROOT") = 1. Moreover, let cnode(n_1, n_2) be the deepest common father of the nodes n_1 and n_2. Then the similarity is computed as follows: $sim(i, q) = \frac{d(cnode(i, q))}{d(i) < d(q) ? d(i) : d(q)}$
Interval[a]	Integer, Real, Date, Time, Timestamp, OrderedSymbol	(low;high) where *low* and *high* are the value ranges of the underlying type (Integer, Real, Date, Time, Timestamp, OrderedSymbol, and their subtypes)	n/a	The similarity is defined in a footnote[b] for space reasons.

a. In contrast to all other types, the interval type is actually a »type constructor«. Interval types can be constructed from all ordered types. Values of an interval type are pairs (l; h) where $l \le h$.

b. The similarity for the interval types is computed based on the similarity of their underlying types. Let *low(i)* and *high(i)* be the lower and upper bound of the interval *i* respectively. Then the similarity is computed as follows (the formulas can be simplified if expressed for a particular underlying type):

$$sim(i, q) = \begin{cases} \dfrac{low(i) \sim low(q) + low(i) \sim high(q)}{2} & \Leftrightarrow high(q) < low(i) \\ \dfrac{low(i) \sim low(q)}{2} & \Leftrightarrow low(q) < low(i) \le high(q) \le high(i) \\ 1 & \Leftrightarrow low(i) \le low(q) \wedge high(q) \le high(i) \\ \dfrac{high(q) \sim high(i)}{2} & \Leftrightarrow low(i) \le low(q) \le high(i) < high(q) \\ \dfrac{low(q) \sim high(i) + high(q) \sim high(i)}{2} & \Leftrightarrow high(i) < low(q) \\ \dfrac{low(i) \sim low(q) + high(q) \sim high(i)}{2} & \text{otherwise} \end{cases}$$

The motivation/derivation of this formula is beyond the scope of this report.

3.3.4 Representation

Types are represented using a type table. The type table contains one row for each type. The rows are sorted alphabetically according to the names of the types. The columns of the table are labeled »Name«, »Supertype«, »Value range«, »Unit of measure«, and »Similarity« corresponding to the components of the type.

A *supertype* is represented by the name of the supertype. The root type is denoted by »TYPE«.

The *value range* is represented by one of the following alternatives:

- **set of ranges** (separated by commas). Each range is represented in the form *low..high*. If an asterisk is used for *low*, it stands for -∞. If an asterisk is used for *high*, it stands for +∞. For each range *low* ≤ *high* must hold. If a set of ranges is specified, the ranges may not overlap.
- **enumeration of the possible values.** For numeric types, the values are written in ascending order. For symbol types, the values are quoted and arranged as follows:
  - **Type »Symbol«.** The symbols are ordered alphabetically.
  - **Type »Ordered Symbol«.** The symbols are ordered from lowest to highest.
  - **Type »Taxonomy Symbol«.** The symbols are arranged either graphically as a tree or textually where the hierarchy is indicated through indentation.
- **graphical representation.** The possible values are enumerated implicitly and completely by the nodes of a graph which is used for specifying the similarity function (see below).

- **tabular representation.** The possible values are enumerated implicitly and completely by the columns/rows of a table which is used for specifying the similarity function (see below).

Symbol glossary

For symbol types (see predefined types above) the representation also includes a narrative text for each symbol defining the symbol. A symbol glossary is used to capture the symbol definitions. The symbol glossary is represented as a table consisting of the columns »Type«, »Symbol«, and »Description«.

In practice, the range of symbol types (including taxonomy symbol types) may not be known completely at the time the ontology is being developed (i.e., more possible values may be identified while the experience base is already in operation). Since it is expected that this situation will occur quite frequently, it must be possible to extend symbol types at any time. Those symbol types which can be extended during operation are marked with »DYNAMIC« in the range field.

The *unit of measure* is represented by text. If the type is not used for measurement attributes, the entry in the respective column is »n/a«.

The *similarity function* can be represented using one of the following alternatives:

- **mathematical formula.** See Section 3.7.
- **graphically.** This is only possible if the *value range* is finite. All possible values are represented by a node. Edges between the nodes are labeled with the similarity value. If there is no edge between two possible values, the similarity of the respective values is 0.
- **tabular.** This is only possible if the *value range* is finite. For each possible value, the table contains a row and a column. The first argument of the similarity function corresponds to the row, the second argument to the column.
- **term »Standard«.** For the predefined types (see above), a standard similarity is defined. The application of the standard similarity function is specified using the keyword »Standard«.
- **term »Inherited«.** The similarity function is inherited from the supertype.

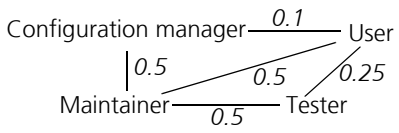
Notes may be used for definitions which do not fit in the type table. In these cases the corresponding note is specified by the name of the note.

3.3.5 Example

Table 5: Example of a type table

Name	Supertype	Value range	Unit of measure	Similarity
Attitude	OrderedSymbol	"rejecting", "disinterested", "motivated"	n/a	Inherited
DurationM	Real	0..*	calendar months	Standard
Cardinal	Integer	0..*	n/a	$sim(i,q) = \begin{cases} 1 & \Leftrightarrow i = q = 0 \\ 1 - \frac{\lvert i-q \rvert}{i > q\,?\; i : q} & \text{otherwise} \end{cases}$
Role	Symbol	Graph-1, DYNAMIC	n/a	Graph-1

Figure 3: Note »Graph-1«: graphical representation of a value range and a similarity function



In Figure 3 a symmetric similarity function is specified. However, a nonsymmetric similarity function can be defined as well using directed edges.

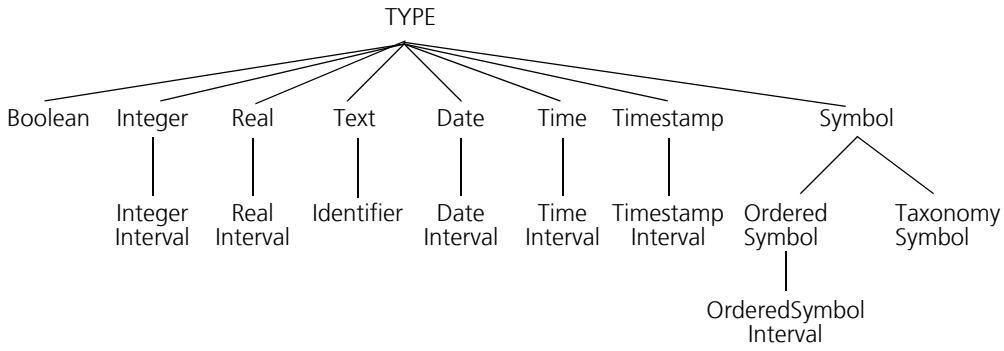
Table 6: Example of a Symbol Glossary

Type	Symbol	Description
Attitude	rejecting	refusing to accept and support measurement
	disinterested	without any interest wrt. measurement
	motivated	interested in and agreeing on the application of measurement
Role	Configuration Manager	integrates updates into the system, coordinates the production and release of versions of the system, and provides tracking of change requests.
	Maintainer	analyze changes, make recommendations, perform changes, perform unit and change validation testing after linking the modified units to the existing system, perform validation and regression testing after the system is recompiled by the Configuration Manager.
	Testers	present acceptance test plans, perform acceptance test and provide change request to the maintainers when necessary.
	Users	suggest, control and approve performed changes.

3.3.6 Alternate Representation

The predefined types can be arranged in a taxonomy and be represented by a tree.

Figure 4: Type hierarchy (taxonomy) of the predefined types



3.4 Nonterminal Concept Attributes

Besides terminal concept attributes, nonterminal attributes may be part of a concept's intension. Here, a nonterminal concept attribute is viewed as an epistemistic primitive.

Nonterminal concept attributes model how a particular software engineering entity is related to other software engineering entities. For example, a GQM goal is related to a GQM plan.

3.4.1 Synonyms

Association, pointer, nonterminal slot, reference

3.4.2 Definition

A nonterminal concept attribute is a 11-tuple (*name*, *kind*, *destination concept*, *reverse attribute, description*, *cardinality*, *default value*, *mandatory*, *value inference*, *inferred attributes*, *standard weight*)

The function dest(attr) and reverse(attr) return the *destination concept* and the *reverse attribute* of *attr* respectively. The function kind(attr) returns the *kind* of *attr*. The predicate mand(attr) is true iff *attr* is mandatory.

For each nonterminal concept attribute a reverse nonterminal attribute exists. Let *c* be the concept of which *n* is a nonterminal attribute. Further, let R(*k*) be

the reverse kind of kind *k* (see Section 3.5.2). Then there exists the nonterminal attribute *n'*=reverse(*n*) where reverse(*n*) ∈intension(dest(*n*)), kind(*n'*)=R(kind(*n*)) and dest(*n'*)=*c* and reverse(*n'*)=*n*.

3.4.3 Description

Name — The name is used for reference purposes. All concept attributes (both terminal and nonterminal) of one concept's *intension* have unique names.

Kind — Each nonterminal concept attribute is of a particular kind (e.g., »is-a« or »has-parts«). The kind is an epistemistic primitive described in Section 3.5.

Destination concept — This component specifies the concept associated with the concept, the nonterminal concept attribute belongs to. The values of a nonterminal concept attribute is a set of instances of the *destination concept*.

Reverse attribute — This component specifies the *reverse attribute* (see above).

Description, cardinality, default value, mandatory, value inference, inferred attributes, standard weight — see Section 3.2.3

(Default) values of nonterminal attributes are specified using the instance names of the referenced instances (see Section 3.6.3).

3.4.3.1 Predefined Similarity Functions

There are two predefined similarity functions for instance names. Let *i, q*, and *k* be instance names of the same concept *c* and *instance(k)* the instance *k* refers to. Then the similarity functions are defined as follows:

$$sim_1(i, q) = sim_{Identifier}(i, q) = (i = q)? 1 : 0$$

$$sim_2(i, q) = sim_c(instance(i), instance(q))$$

Informally, the first similarity function is based on the comparison of the names whereas the second similarity function is based on the comparison of the instances *i* and *q* refer to. Note that sim_2 cannot be used exclusively, because it has to be taken care that no circular computations are performed. Circular computations are possible since nonterminal concept attributes allow instances to

reference each other. One way to avoid circular computations is to mark all visited instances. If *i* or *q* refer to a marked instance then sim_1 is used instead of sim_2. Implementations may restrict the usage of sim_2 even further for performance reasons (see Section 4.1).

3.4.4 Representation

Nonterminal concept attributes are represented using the same concept attribute table as used for the terminal concept attributes (Section 3.2.4).

The entry for the »Type« column is a string of the form »<name of kind> [<name of destination concept>].[<name of reverse attribute>]« or »<reverse name of kind> [<name of destination concept].[name of reverse attribute>]«. The default value is represented using instance names.

Nonterminal concept attributes of the predefined kinds (Section 3.5.3) »is-a« and »has-instances« are not represented in the table.[1] The »is-a« relationship is expressed through the specification of the super concept (for an example, see Table 3), whereas the »has-instances« relationships are represented by their reverse kind »instance-of« as part of the instance tables (Section 3.6.4). Alternatively these relationships can be represented graphically (Section 3.4.6 and Section 3.5.6).

3.4.5 Example

Table 7: Nonterminal attribute of the concept »Measurement Characterization«

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt	context	references the respective context characterization	1	part-of [Context Characterization].[measurement context]	-	yes	-	-	10

1 The reason that nonterminal concept attributes of these kinds are not part of the concept attribute table is that they are not instantiated for the instances of a concept.

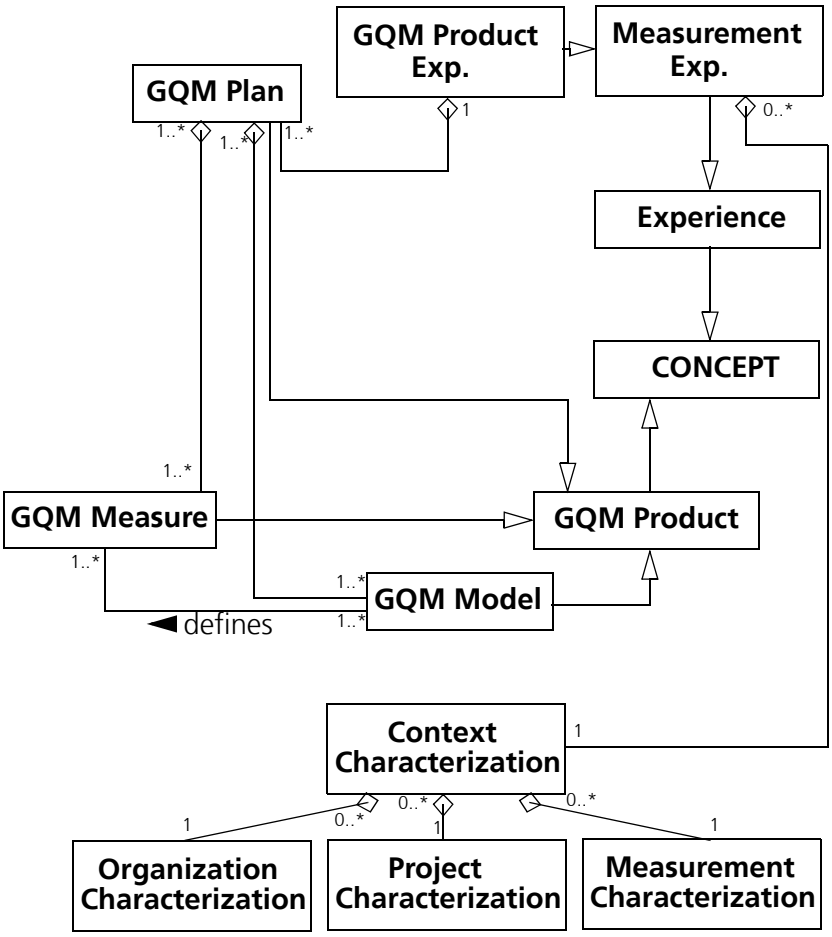
3.4.6 Alternate Representation

Concept diagram

Nonterminal concept attributes can be represented graphically in a concept diagram. Concept diagrams use a notation similar to the class diagrams of UML [Cor97]. Concepts are represented by boxes with their names written inside. A nonterminal attribute is represented by an edge between two concepts. At the side of the *destination concept* the *cardinality* is written. In addition, the name of the attribute is written at that side, too (optionally). Since there always exists a reverse nonterminal attribute, there will be a cardinality at both ends of the edge. The edge is labeled with the *kind* of the nonterminal attribute (only one direction is labeled) and a filled triangle pointing in the read direction.

Predefined kinds use a special notation (Section 3.5.6).

Figure 5: Graphical representation of nonterminal attributes



It is possible to:

- color each *kind* of nonterminal attribute differently
- use a separate figure for each *kind* of nonterminal attribute (for an example see Appendix).

3.5 Kinds of Nonterminal Concept Attributes

Each nonterminal concept attribute is of a particular kind. Here a kind is viewed as an epistemistic primitive.

Kinds of nonterminal concept attributes are analog to types of terminal concept attributes. They model the semantic relationships between software engineering entities. For example, the relationship that exists between a GQM Plan and a GQM Measure is of the kind »has-parts«.

3.5.1 Synonyms

Type

3.5.2 Definition

A kind of a nonterminal concept attribute is a 5-tuple (*name*, *reverse name, purpose*, *structure*, *properties*).

R(kind) denotes the *reverse name* of *kind*.

3.5.3 Description

Name The name is used for reference purposes. All kinds of nonterminal attributes have unique names.

*Reverse name*As stated in Section 3.4.2, nonterminal attributes come pairwise. This is illustrated by Figure 5. The relationship can be read in one direction (the direction shown by the filled triangle). However, it can also be read in the other direction using the reverse name.

Purpose Just as there is a reason for concepts to be part of an ontology, there is a reason for defining a kind of nonterminal attributes. The purpose is a set of typical usage scenarios which show how this kind of nonterminal attributes can be used effectively.

Structure Instances (Section 3.6) are related through kinds of nonterminal attributes. If instances are interpreted as nodes and the values of the nonterminal attributes as edges from a source to a destination instance[1], a structure is defined which has certain properties. The structure can be a set of trees, DAGs (directed acyclic graphs), or graphs (without any restrictions).

Properties Besides the structure property, additional properties may hold true. Examples of additional properties are: symmetry, transitivity, etc.

3.5.3.1 Predefined Kinds

There are four predefined kinds.

Table 8: Predefined kinds

Kind	Reverse name	Description	Structure	Properties
is-a	has-specialization	Denotes a specialization of a concept.	tree (single inheritance)	Transitivity Every concept listed in the concept glossary is a specialization of exactly one concept (single-inheritance). There is one predefined concept »CONCEPT« which is the most general concept. It has only one terminal attribute with the name »Id« of the type »Identifier«. *Id* corresponds to the name of the instance. Let c_1 and c_2 be concepts. If c_2 is-a c_1 then the following properties (inheritance) hold: $intension(c_1) \subseteq intension(c_2)$ $extension(c_2) \subseteq extension(c_1)$ $assert(c_2) \Rightarrow assert(c_1)$ $precond(c_2) \Rightarrow precond(c_1)$

1 The source instance is an instance of the concept to which the nonterminal concept attribute belongs to. The destination instance is an instance of the destination concept as defined by the nonterminal attribute.

Kind	Reverse name	Description	Structure	Properties
instance-of	has-instances	Denotes a special is-a relation. An instance is an element of the extension of a concept.	tree with no intermediary nodes	Let *i* be an instance of the concept c. Then the following properties hold: $intension(i) = intension(c)$ $i \in extension(c)$
has-parts	part-of	Denotes a decomposition. Subparts may be shared among concepts.	DAG	Transitivity
has-decomposition	decomposition-of	Denotes a decomposition where the subparts exist only if the surrounding part (aggregate) exists.	tree	Transitivity A concept may have at most one nonterminal attribute of the kind »decomposition-of«. The cardinality of a nonterminal attribute of the kind »decomposition-of« is always 1. If a concept has a nonterminal attribute of the kind »decomposition-of«, it may not have any nonterminal attributes of kind »part-of«.

3.5.4 Representation

Kind table

Kinds are represented using a kind table. The kinds are listed alphabetically by name, one in each row of the table. The columns are labeled »Kind«, »Reverse name«, »Purpose«, »Structure«, and »Properties« corresponding to the components of a kind. Each row defines implicitly the reverse kind as an additional kind which can be used for nonterminal attributes.

The »is-a« and »instances-of« relationships are represented outside the concept attribute table (Section 3.6.4). If a concept c_1 is-a concept c_2 then the concept attribute table of c_2 is also relevant for c_1 (i.e., the concept attributes of c_2 are not duplicated).

3.5.5 Example

Table 9: Example representing kinds of nonterminal attributes

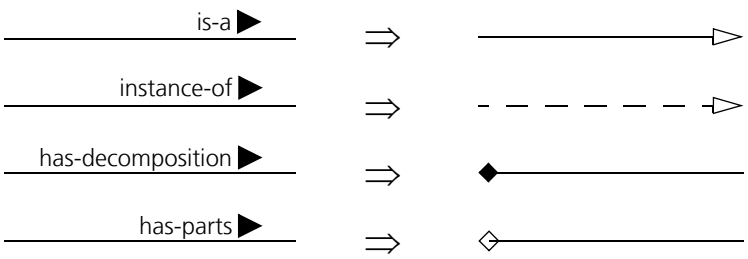
Kind	Reverse name	Purpose	Structure	Properties
defines	defined-by	Documents the dependency of instances. If an instance is changed, it needs to be checked for all instances defined by the changed instance whether they need to be changed, too.	DAG	transitivity

Kind	Reverse name	Purpose	Structure	Properties
depends-on	has-dependents	A special kind used for data collection procedures. The collection of data may depend on the collection of other data. For example, the finish date needs only be collected if the start date has also been collected in order to compute the duration of a process step. In this case, the collection of the finish date depends on the collection of the start date. If a measure is deactivated (i.e., temporarily no data for this measure is collected), all of its dependents should also be deactivated.	tree	transitivity

3.5.6 Alternate Representation

In the graphical representation the predefined kinds are represented different from »user-defined« kinds.

Figure 6: Graphical representation of predefined kinds



Note: In contrast to the concept attribute table, »instance-of« relationships can be part of the graphical representation.

3.6 Instances

The extension of a concept is a set of instances. Here, an instance is viewed as an epistemistic primitive. Usually, only a few instances (if any) are part of a software engineering ontology because most instances belong to the linguistic level. However, there is knowledge which is of importance to the modeled domain as a whole and which can be easily represented as instances. For example, instances of »GQM Measure« can be specified which are regularly used in a given domain (called core measures).

3.6.1 Synonyms

Case, characterization, object

3.6.2 Definition

An instance is a 3-tuple (*name*, *concept*, *values*).

There is an »instance-of« relationship between the instance and *concept* as well as all super concepts of *concept*, that is, if c_1 is a direct or indirect super concept of c_2 then:

$$\forall i \in \text{extension}(c_2): i \in \text{extension}(c_1)$$

The function val(attr) returns the value of the instance's attribute *attr*.

The component *values* is a set of pairs (*attr*, *value*) where *value* is the attribute value of the attribute denoted by *attr* (i.e., *value* = val(*attr*)). The following properties hold for all concept attributes *attr* of an instance:

$$\text{val}(\text{attr}) \subseteq \text{range}(t_{\text{attr}}) \text{ for terminal concept attributes}$$

$$\text{val}(\text{attr}) \subseteq \text{extension}(\text{dest}(\text{attr})) \text{ for nonterminal concept attributes}$$

$$\text{mand}(\text{attr}) \Rightarrow \text{val}(\text{attr}) \neq \text{undefined}$$

3.6.3 Description

Name — The name is used for reference purposes. All instances have unique names.

Concept — This component specifies from which concept the instance is instantiated from, i.e., the intension of the instance is defined by the intension of the specified concept.

Values — The attribute values of the instance. If a value list is specified for an attribute (cardinality > 1) all values of the list have to be unique (e.g., »{2, 3, 4, 2,6, 3}« is an invalid value because »2« and »3« are not unique)..

3.6.4 Representation

Instance table — An instance *i* is represented by an instance table. The instance table has three columns labeled »Layer«, »Attribute«, and »Value«. The first column corresponds to the layers of the attributes of *concept*, while the latter two columns correspond to the components of *value*. In addition, the most special concept (i.e., *concept*) is documented near the table.

3.6.5 Example

Table 10: Example instance

Concept: GQM Measure		
Layer	Attribute	Value
artif	id	failure_count_1
	definition	count of failure reports turned in before delivery
	scale	"ratio"
	unit	"n/a"
	range	decomposition [Type]
	comments	-
I/F	assumption	for each failure detected a failure report is filled out
	model	"unknown"
	data collections procedure	"unknown"
	questionnaire question	"unknown"
ctxt	gqm plan	"unknown"

3.6.6 Alternate Representation

none

3.7 Formulas

Formulas are used for similarity functions, assertions, preconditions, and value inferences. As such they model:

- the similarity between software engineering artifacts
- the similarity between software engineering quality values
- dependencies between software engineering quality values

3.7.1 Synonyms

Rules

3.7.2 Definition

A formula for a value inference is a mathematical term of type *t* where the variables are concept attributes and *t* is the type of the inferred attribute.

A formula for an assertion or a precondition is a mathematical term of type Boolean where the variables are concept attributes.

A formula for a concept similarity function is of the form

$$sim_I(i, q) = \text{<mathematical term>}$$

where $I \in$ {»artif«, »I/F«, »ctxt«}, *i* and *q* are instances of the concept the similarity function belongs to, and <mathematical term> returns a real number in the range [0; 1]. The similarity is calculated from *i* to *q*.

A formula for a type similarity function is of the form

$$sim(i, q) = \text{<mathematical term>}$$

where *i* and *q* are values within the range of the type, and <mathematical term> returns a real number in the range [0; 1] or the special value »undefined«. The function computes the similarity from *i* to *q*.

A mathematical function may use the following operators and functions:

- **Operators:** (), ||, ., - (unary), NOT, ~, ×, /, +, - (binary), AND, OR, ?:, WHERE, =, ≠, <, >, ≤, ≥.
- **Functions:** avg, card, cnode, d, date, day, daydiff, del, father, filter, high, hour, ins, intersect, low, max, maxvalue, min, minute, minvalue, month, p, pos, rangesize, secdiff, second, sum, sysdate, systime, systimestamp, time, timestamp, undef, union, year.

3.7.3 Description

Operators

The operators have the following signature and meaning (»int« is short for »integer«):

Table 11: Signature and meaning of operators

Operator	Signature(s)	Description
()	(any) -> any	Changes sequence of computation
||	|int| -> int, |real| -> real	Computes the absolute value
.	name.attr -> any	Computes an attribute value (see below)
- (unary)	-int -> int, -real -> real	Negates a numeric value
NOT	NOT boolean -> boolean	Negates a boolean expression
~	any ~ any -> real	Computes the similarity between two attribute values. Both values must be of the same type
×	int × int -> int, int × real -> real, real × int -> real, real × real -> real	Multiplies two numeric values
/	int / int -> real, int / real -> real, real / int -> real, real / real -> real	Divides two numeric values

Operator	Signature(s)	Description
+	int + int -> int, int + real -> real, real + int -> real, real + real -> real	Adds two numeric values
- (binary)	int - int -> int, int - real -> real, real - int -> real, real - real -> real	Subtracts two numeric values
AND	boolean AND boolean -> boolean	Logically ands two boolean expressions. A short-circuit computation (i.e., the computation of the second argument is only performed if the first argument computes to false) is not used because the second argument could compute to a special value in which case the whole expression would compute to the special value (see page 44).
OR	boolean OR boolean -> boolean	Logically ors two boolean expressions. Short-circuit computation is not used.
?:	boolean? any : any -> any	Returns the second argument if the boolean expression is true. Otherwise it returns the third argument. A short- circuit computation is used, i.e., either the second or the third argument is computed but not both.
WHERE	any WHERE boolean	Returns »undefined« if the boolean expression is false. Otherwise it returns the value of the first argument. A short-circuit computation is used (i.e., the first argument is not computed if the boolean expression computes to false).
=, ≠, <, >, ≤, ≥	num OP num OP num OP …-> boolean, orderedSymb OP orderedSymb OP orderedSymb OP … -> boolean, date OP date OP date OP … -> boolean where OP ∈ {=, ≠, <, >, ≤, ≥} symbol OP symbol OP symbol OP …-> boolean, boolean OP boolean OP boolean OP … -> boolean, text OP text OP text OP … -> boolean where OP ∈ {=, ≠}, interval OP interval OP … -> boolean where OP ∈ {=, ≠}	Compares values. Value comparison expressions may be specified as usual in mathematics, e.g., 10 < x < 100 or x < y < z are legal expressions.

The priority and associativity of the operators are defined as follows:

Table 12: Priority and associativity of operators

Priority level	Operators	Associativity
1	()	left to right
2	- (unary), NOT	right to left
3	~, ×, /	left to right
4	+, -	left to right
5	AND	left to right
6	OR	left to right
7	?:	right to left
8	WHERE	right to left
9	=, ≠, <, >, ≤, ≥	left to right

Constants

Formulas may use the following constants:

Table 13: Predefined constants

Constant	Type
true, false	boolean
…, -1, 0, 1, 2, …	int
..	real
undefined	*none (special value)*
n/a	*none (special value)*
unknown	*none (special value)*

Table 14: User defined constants

Notation	Type	Example
"<any text>" (quotes are represented by »""«)	text	"Hugo walks to his ""VW Golf"""
<type> ("<symbol name>")	symbol type *type*	Attitude("disinterested")

Constants of type date, time, and timestamp may be generated using the *date*, *time*, and *timestamp* functions respectively.

Attribute values

The specification of values depends upon for what type the mathematical formula is used:

Table 15: Notation for attribute values

Type of formula	Notation	Meaning	Example
Type similarity function	i or q	*i* refers to the first argument, *q* to the second of the similarity function. The similarity is computed from *i* to *q*.[a]	i
Concept similarity function	i.[<attr-name>] or q.[<attr-name>]	*i* refers to the first argument, *q* to the second argument of the similarity function. The similarity is computed from *i* to *q*.	i.[abstraction sheet]
Value inference, assertion, precondition	[<attr-name>]		[abstraction sheet]

a. The notion of »from … to« is used here in analogy to distances (= dissimilarities) that are always computed from one instance to another.

Attributes with a cardinality of exactly 1 return a single value while attributes with a cardinality range or a cardinality other than 1 return a list of values.

In case of the concept similarity function, the assertion, the precondition, and the value inference, the ».« operator may also be used for navigational expressions. For instance, the expression [gqm plan].[gqm questions] as part of a value inference formula for the concept »Measurement Characterization« returns a list of GQM questions lists.

Functions

The functions have the following signature and meaning (»int« is short for »integer«):

Table 16: Signatures and meaning of functions

Function	Signature(s)	Description
avg	avg(list of <elem>) -> <elem>	Computes the average. <elem> must be a numerical type (int or real)
card	card(list) -> int	Computes the number of elements in the list.
cnode	cnode(taxonomySymbol, taxonomySymbol) -> taxonomySymbol	Computes the deepest common father of two symbols of a taxonomy.
d	d(taxonomySymbol) -> int	Returns the depth of a symbol where d("ROOT") = 1.
date	date(int_1, int_2, int_3) -> date	Assembles a date value (int_1: day, int_2: month, int_3: year)
day	day(date) -> int, day(timestamp) -> int	Returns the day of a date or timestamp
daydiff	daydiff($date_1$, $date_2$) -> int	Returns the difference in days between two dates. If $date_1 < date_2$, daydiff returns a negative number.
del	del(list, int) -> list	Removes an element from a list given by the first argument. The element to be deleted is specified by the second argument (position in list). The resulting list is returned.
father	father(taxonomySymbol, taxonomySymbol) -> boolean	Returns true iff the first argument is a direct or indirect father of the second argument.
filter	filter(concept name, list of instances) -> list of instances	Returns all instances of a list which belong to the concept identified by *concept name*. At the same time it coerces the instances. This function is typically used in conjunction with »is-a« relationships. The value inferences in Section 3.2.5 show an exemplary application.
high	high(interval of elem) -> elem	Returns the upper bound of an interval value
hour	hour(time) -> int, hour(timestamp) -> int	Returns the hour of a time or timestamp
ins	ins(list, pos, elem) -> list	Inserts an element at a given position in a list. The resulting list is returned.
intersect	intersect(list of list) -> list	Computes the intersection of the sublists
low	low(interval of elem) -> elem	Returns the lower bound of an interval value
max	max(list of elem) -> elem	Computes the maximum of a list of numeric values, dates, or ordered symbols
maxvalue	maxvalue(), maxvalue(attr-name) -> elem	Returns the highest value of the range of an attribute type. In case of »*« (positive infinity), an implementation-dependent maximal value is returned. The function is applicable only for ordered types. The first signature is used for type similarity functions, the second signature for concept similarity functions, assertions, preconditions, and value inferences.
min	min(list of elem) -> elem	Computes the minimum of a list of numeric values, dates, or ordered symbols
minute	minute(time) -> int, minute(timestamp) -> int	Returns the minute of a time or timestamp

Function	Signature(s)	Description
minvalue	minvalue(), minvalue(attr-name) -> elem	Returns the lowest value of the range of an attribute type. In case of »*« (negative infinity), an implementation-dependent minimal value is returned. The function is applicable only for ordered types. The first signature is used for type similarity functions, the second signature for concept similarity functions, assertions, preconditions, and value inferences.
month	month(date) -> int, month(timestamp) -> int	Returns the month of a date or timestamp
p	p(attr-name) -> real	Returns the parameter of an attribute (see Chapter 4 for details)
pos	pos(orderedSymbol) -> int	Returns the position of a value of an ordered symbol type. The return value > 0.
pos	pos(list, int, elem) -> int	Returns the first occurrence of an element in a list starting at the position of the second argument. If the element is not found, 0 is returned, otherwise the position of the first occurrence (> 0).
rangesize	rangesize(), rangesize(attr-name) -> int	Returns the number of elements in the range of an attribute type. It is applicable only for types with finite ranges. The first signature is used for type similarity functions, the second signature for concept similarity functions, assertions, preconditions, and value inferences.
secdiff	secdiff($time_1$, $time_2$) -> int, secdiff($timestamp_1$, $timestamp_2$) -> int	Returns the difference in seconds between two times or timestamps. If $time_1 < time_2$ or $timestamp_1 < timestamp_2$, secdiff returns a negative number.
second	second(time) -> int, second(timestamp) -> int	Returns the second of a time or timestamp
sum	sum(list of elem) -> elem	Sums up all elements of a list. Elements must be of type int or real.
sysdate	sysdate() -> date	Returns the current date
systime	systime() -> time	Returns the current time
systime-stamp	systimestamp() -> timestamp	Returns the current timestamp
time	time(int_1, int_2, int_3) -> time	Assembles a time (int_1: hour, int_2: minute, int_3: second)
timestamp	timestamp(int_{123456}) -> timestamp	Assembles a timestamp (int_1: day, int_2: month, int_3: year, int_4: hour, int_5: minute, int_6: second)
union	union(list of list) -> list	Computes the union of the sublists
year	year(date) -> int, year(timestamp) -> int	Returns the year of a date or timestamp

Special values

The special values are propagated depending on the type of the mathematical formula.

A type similarity function returns the special value »undefined« if any subcomputation returns a special value. A concept similarity function may not return a special value, that is, it has to take care that all special values are »converted«

into a numeric value (for an example, see the definition of the standard concept similarity function).

For a value inference, assertion, or precondition, a subcomputation may return any of the special values depending on the values of its arguments. The special values are prioritized as follows: »n/a«, »unknown«, »undefined«, regular value. Examples:

- n/a + 5 = n/a
- 6 - unknown = unknown
- 5 + 6 = 11
- p(undefined) = undefined
- n/a AND unknown = n/a
- unknown / undefined = unknown

but (see description of type similarity function):

- n/a ~ 5 undefined
- 7 ~ unknown = undefined
- n/a ~ n/a = undefined

If an inferred value is not in the value range of the inferred attribute's type, the attribute's value is set to »undefined«.

If an assertion or precondition computes to a special value, it is interpreted as false.

3.7.4 Representation

Concept similarity functions, assertions, and preconditions are represented in the form as shown in the definition section together with its concept attribute table. Type similarity functions are represented in the form as shown in the definition section in the »Sim« column of the type table. Finally, value inferences are represented in the form as defined in the definition section in the »Value inference« column of the concept attribute table.

3.7.5 Example

See Section 3.2.5, Section 3.3.3, and Section 3.3.5.

3.7.6 Alternate Representation

Instead of »/« a fraction line may be used, i.e.,

$$x/y \quad \rightarrow \quad \frac{x}{y}$$

Instead of »?:« curly brackets may be used, i.e.,

$$\text{cond? } \text{val}_1 : \text{val}_2 \quad \rightarrow \quad \left\{ \begin{array}{ll} \text{val}_1 & \Leftrightarrow \text{cond} \\ \text{val}_2 & \text{otherwise} \end{array} \right\}$$

The right curly bracket may be left out if nothing follows.

4 Semantics

The retrieval and maintenance of context-specific knowledge can be guided if the model of the underlying structure is described explicitly. For this purpose, the structure model description must be formal so it can be interpreted by a computer system. The notation defined in the previous chapter was conceived with this objective in mind.

In the approach presented, the context-specific knowledge is represented on the linguistic knowledge level (see Section 2.2) and can be described using constructs of the conceptual level, i.e., using the standard vocabulary an ontology defines. Technically, context-specific knowledge is represented as instances of concepts.

This chapter describes how context-specific knowledge can be retrieved (Section 4.1) and maintained (Section 4.2 through Section 4.4) based on REFSENO. For each of the two areas (retrieval and maintenance) it is shown what operations are necessary. Operations are defined by their inputs, outputs, and side-effects on the representation of the context-specific knowledge. Furthermore, the operations are related to the tasks for the maturing of an experience base (cf. Figure 7).

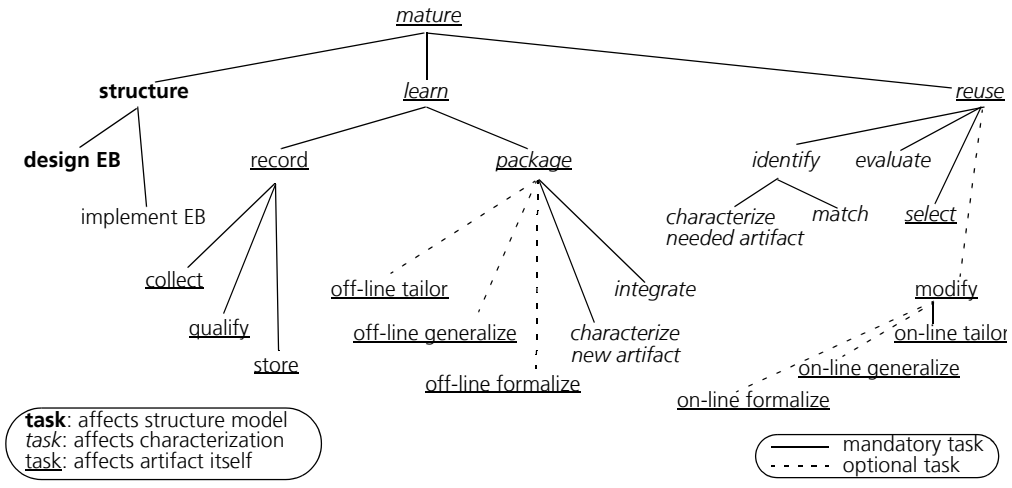


Figure 7: Maturing of an Experience Base (based on [BR91])

The performance of the tasks is more complex than described in the following sections, because the sections only address the changes in the representation of context-specific knowledge on the linguistic level for single instances. Thus, only elementary operations are described. These must be combined to provide »logical operations« (e.g., »remove GQM plan with all knowledge related to it«).

Such logical operations require, however, semantic knowledge which is not specified by REFSENO.

In order to show the effects of the various operations, the same contents of the experience base will be used. Figure 8 gives an overview of the contents. Table 17 through Table 29 show the instances while the intension is taken from the example in Appendix .

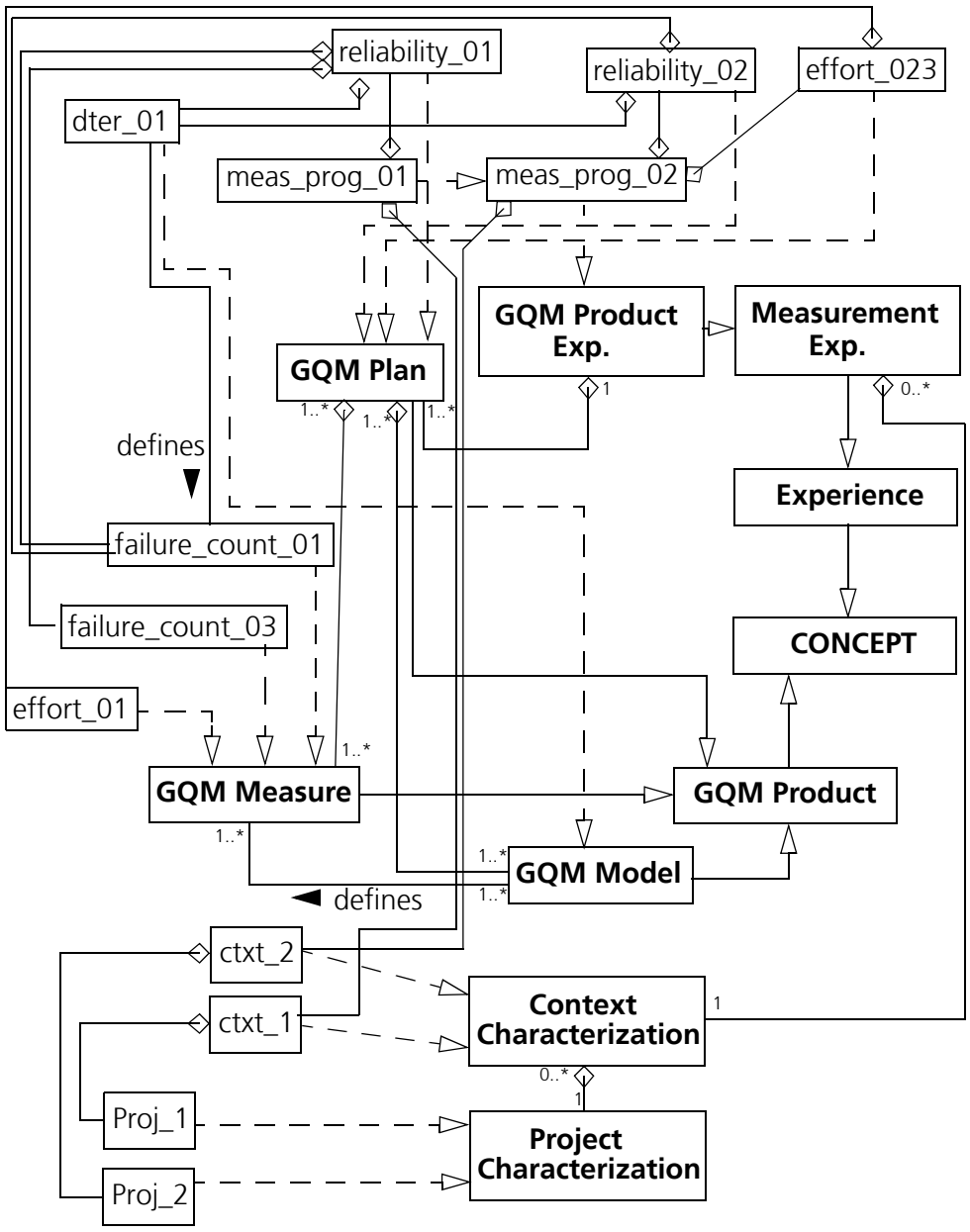


Figure 8: Overview of the contents of the experience base used in the examples for the descriptions of the operations

Table 17: Instance »proj_1«

Concept: Project Characterization		
Layer	Attribute	Value
artif	id	"proj_1"
	project name	C: Z
	project start	6-1-1997
	project end	undefined
	duration	18
	team size	2-20
	effort	unknown
	application of standard software process	false
	life-cycle model used	"waterfall"
	tools	unknown
	programming languages	unknown
	estimated product size	unknown
	type of software	{"embedded/real-time systems"}
	number of installations	1
	memory constraints	"normal"
	performance constraints	"normal"
	target platforms	{"embedded processors", "workstations"}
	newness to state of art	"initial delivery"
	functionality	"important"
	reliability	"desirable"
	usability	"desirable"
	efficiency	"important"
	maintainability	"important"
	portability	"unimportant"
ctxt	project goals	{"Complete development in time and budget"}
	context	{"ctxt_1"}
	comments	undefined

Table 18: Instance »proj_2«

Concept: Project Characterization		
Layer	Attribute	Value
artif	id	"proj_2"
	project name	C: X
	project start	2-1-1997
	project end	9-30-1998
	duration	19
	team size	3-10

Concept: Project Characterization		
Layer	Attribute	Value
artif	effort	unknown
	application of standard software process	false
	life-cycle model used	"waterfall"
	tools	unknown
	programming languages	{"Ada"}
	estimated product size	unknown
	type of software	{"embedded/real-time systems"}
	number of installations	1
	memory constraints	"normal"
	performance constraints	"normal"
	target platforms	{"embedded processors", "workstations"}
	newness to state of art	{"initial delivery"}
	functionality	"important"
	reliability	"desirable"
	usability	"desirable"
	efficiency	"important"
	maintainability	"important"
	portability	"unimportant"
ctxt	project goals	{"Complete development in time and budget"}
	context	{"ctxt_2"}
	comments	undefined

Table 19: Instance »ctxt_1«

Concept: Context Characterization		
Layer	Attribute	Value
artif	id	"ctxt_1"
	organization context	…
	project context	"proj_1"
	measurement context	…
ctxt	measurement experience	"meas_prog_01"
	Comments	undefined

Table 20: Instance »ctxt_2«

Concept: Context Characterization		
Layer	Attribute	Value
artif	id	"ctxt_2"

Concept: Context Characterization		
Layer	Attribute	Value
artif	organization context	…
	project context	"proj_2"
	measurement context	…
ctxt	measurement experience	"meas_prog_02"
	Comments	undefined

Table 21: Instance »meas_prog_01«

Concept: GQM Product Experience		
Layer	Attribute	Value
artif	id	"meas_prog_01"
	viewpoint	"Maintainer"
	representation form	"structured text"
	owner	"Goofy"
	status	"incomplete"
	version	0.02
	last change	3-2-1997
	readers	"C: Z"
	gqm plan	"reliability_01"
	measurement plan	…
I/F	preconditions for reuse	undefined
ctxt	acquisition technique	"interview"
	expected adaptations	undefined
	expected cost of reuse	undefined
	dates of reuse	undefined
	guidelines of reuse	undefined
	comments	undefined
	context	"ctxt_1"

Table 22: Instance »meas_prog_02«

Concept: GQM Product Experience		
Layer	Attribute	Value
artif	id	"meas_prog_02"
	viewpoint	"Maintainer"
	representation form	"structured text"
	owner	"Goofy"
	status	"incomplete"
	version	0.11
	last change	1-2-1997
	readers	"C: Z"
	gqm plan	"reliability_01"

Concept: GQM Product Experience		
Layer	Attribute	Value
artif	measurement plan	…
I/F	preconditions for reuse	undefined
ctxt	acquisition technique	"interview"
	expected adaptations	undefined
	expected cost of reuse	undefined
	dates of reuse	undefined
	guidelines of reuse	undefined
	comments	undefined
	context	"ctxt_1"

Table 23: Instance »reliability_01«

Concept: GQM Plan		
Layer	Attribute	Value
artif	id	"reliability_01"
	comments	undefined
	gqm goal	…
	abstraction sheet	…
	gqm question	{…}
	gqm model	{"dter_01", …}
	gqm measure	{"failure_count_1", "fault_count_03", …}
ctxt	gqm product experience	"meas_prog_01"

Table 24: Instance »reliability_02«

Concept: GQM Plan		
Layer	Attribute	Value
artif	id	"reliability_02"
	comments	undefined
	gqm goal	…
	abstraction sheet	…
	gqm question	{…}
	gqm model	{"dter_01", …}
	gqm measure	{"failure_count_1", …}
ctxt	gqm product experience	"meas_prog_02"

Table 25: Instance »effort_023«

Concept: GQM Plan		
Layer	Attribute	Value
artif	id	"effort_023"
	comments	undefined

Concept: GQM Plan		
Layer	Attribute	Value
artif	gqm goal	…
	abstraction sheet	…
	gqm question	{…}
	gqm model	{…}
	gqm measure	{“effort_01”, …}
ctxt	gqm product experience	“meas_prog_02”

Table 26: Instance þ»failure_count_1«

Concept: GQM Measure		
Layer	Attribute	Value
artif	id	“failure_count_1”
	comments	undefined
	definition	“count of failure reports turned in before delivery”
	scale	“ratio”
	unit	n/a
	range	…
I/F	assumption	{“for each failure detected a failure report is filled out”}
	model	{“dter_01”, …}
	data collections procedure	…
	questionnaire question	…
ctxt	gqm plan	{“reliability_01”, “reliability_02”}

Table 27: Instance »fault_count_03«

Concept: GQM Measure		
Layer	Attribute	Value
artif	id	“fault_count_03”
	comments	undefined
	definition	“count of fault per life cycle phase where the fault was introduced”
	scale	“nominal”
	unit	n/a
	range	…
I/F	assumption	{“the sw process includes the phases REQ, HLD, LLD/IMP”}
	model	{…}
	data collections procedure	…
	questionnaire question	…

Concept: GQM Measure		
Layer	Attribute	Value
ctxt	gqm plan	{"reliability_01"}

Table 28: Instance »effort_01«

Concept: GQM Measure		
Layer	Attribute	Value
artif	id	"effort_01"
	comments	undefined
	definition	"measures the effort spent on a project"
	scale	"ratio"
	unit	"person-month"
	range	…
I/F	assumption	{"the sw process includes the phases REQ, HLD, LLD/IMP"}
	model	{…}
	data collections procedure	…
	questionnaire question	…
ctxt	gqm plan	{"effort_023"}

Table 29: Instance »dter_01«

Concept: GQM Model		
Layer	Attribute	Value
artif	id	"dter_01"
	comments	undefined
	type	"quality model"
	category	"descriptive"
	definition	"distribution testing effectiveness per role"
I/F	assumption	{"testing is done by maintainers, testers, and users"}
	data source	…
	gqm measure	{"failure_count_1"}
	question	{…}
ctxt	gqm plan	{"reliability_01", "reliability_02"}

4.1 Retrieval of Context-Specific Knowledge

Context-specific knowledge is retrieved using a *query specification*:

Input A query specification in the form of one »main« and optionally several related instances. The »main« instance is an instance of any concept that is justified with a usage scenario.

Output Either an error message telling which attribute value specified is out of range or a list of instances similar to the »main« instance specified together with the similarity value (from the returned instance to the specified instance[1]). The concept similarity function is used for computing the similarity value. The first argument to the similarity function is the instance in the experience base while the second argument is always the specified instance.

The list of instances returned is limited to the extension of the concept of the »main instance«[2].

Side effects *none*

4.1.1 Relation to Software Engineering

The retrieval supports the identification, selection, and partially the evaluation tasks:

1 **Characterize needed artifact**[3]**.** Characterization is performed by defining the query specification.

2 **Match.** The query specification contains one distinguished instance, the »main« instance. Only instances which have the same intension as the »main« instance will be returned. This restricts the search space.

3 **Select.** For all instances returned a similarity value is computed using the concept similarity function. By sorting the potential instances (from the match mechanism) according to their similarity, a selection can be made by cutting off the list at a reasonable point (e.g., only the first 10 instances are displayed).

4 **Evaluate.** The similarity value and the position in the list give decision support for choosing the most appropriate instance. Of course, whether the most appropriate instance will actually be reused depends on the effort needed to tailor the instance to the needs at hand.[4] In order to make the

1 Similarity functions are not symmetric. For instance, if the similarity between two modules is computed, the similarity value depends on the services provided by a particular module in the experience base (e.g., the deletion of a service requires less effort than providing a new service).
2 The extension includes the instances of the subconcepts of the »main« instance's concept (cf. Table 8).
3 In this report, the term »artifact« does not only refer to documents and/or files existing in reality, but also to parts thereof (e.g., GQM questions as part of a GQM plan) as well as to »nondiscriminant attributes' values«. »Nondiscriminant attributes« are attributes with a standard weight of 0. They cannot be specified by a query. For example, lessons learned can be stored completely as instances [GRA+98]. Therefore, the term »artifact« is used for all concepts that are justified with a usage scenario in the concept glossary.

right decision, it also has to be estimated how much effort would be needed to create the needed artifact from scratch.

4.1.2 Description

Simple query specification

The simplest query specification is one that involves only the »main« instance. Instances used for retrieval differ from instances stored in the experience base in that the:

- values for attributes declared as »mandatory« in the ontology, may be of the special value »undefined«.
- assertion does not need to be true.
- precondition does not need to be true.
- values can be specified for attributes whose values can be inferred. For attributes whose value is not specified, the value inference is used to try to infer a value before the actual query is performed.
- values for »nondiscriminant attributes« cannot be specified. An attribute is *nondiscriminant* if its standard weight is 0.

However, if specified, attribute values have to be either within the value range of the corresponding type or be of one of the special values »undefined«, »unknown«, or »n/a«.

Table 30 shows such an example. In the example, nondiscriminant values are shaded. The objective of the query is to find some kind of measure associated with effort measured in person-months. Result of the retrieval is the list of instances of the concept GQM measure (cf. Figure 8) together with the respective similarity values (see Table 31). The default values for the weights w_{artif}, $w_{I/F}$, w_{ctxt} are 0.3. The result shows a total similarity for the instance »failure_count_1«. The reason for this is that the value of »unit« for this instance is »n/a«. Thus, the local similarity (cf. footnote on page 12) computes to »undefined« (see Section 3.7.3) which is not considered in the concept's similarity function (see Section 3.1.4).

Table 30: Query specification 1 for a GQM measure

Concept: GQM Measure Id of temporary instance: "tmp23042"		
Layer	Attribute	Value
artif	id	undefined
	comments	
	definition	

4 At this point it is unclear in how far the defined ontology can be of help for estimating the tailoring effort.

Concept: GQM Measure Id of temporary instance: "tmp23042"		
Layer	Attribute	Value
artif	scale	"ratio"
	unit	"person-month"
	range	undefined
I/F	assumption	
	model	undefined
	data collections procedure	undefined
	questionnaire question	undefined
ctxt	gqm plan	undefined

Table 31: Result of query specification 1

Instance	Similarity value
failure_count_1	1.0000
effort_01	1.0000
fault_count_03	$0.833\overline{3}$

A system implementing REFSENO should also offer the possibility to inspect both the instance (i.e., the characterization of the artifact) as well as the artifact itself (possibly by invoking a specialized tool for the artifact). This inspection would support the evaluation by a human.

Complex query specification

In the first example only terminal attributes were specified. However, nonterminal attributes may also be specified as part of a query specification. In the latter case, the query specification is called *complex query specification*. There are two possibilities for specifying a nonterminal attribute: relationships to instances that already exist in the experience base, and relationships to temporary instances that are part of the query specification. After the retrieval is complete, these temporary instances will cease to exist. Also, temporary instances are user-specific. This means, if user A and user B query the experience base simultaneously, A cannot use the temporary instances of B and vice versa. Relationships between temporary instances are not bidirectional. The id of temporary instances is provided by the retrieval system.

The similarity function associated with a given element[1] of a nonterminal attribute's value depends on whether the element refers to an instance of the experience base or to a temporary instance. sim_1 is used for first case whereas

1 The term »element« refers to either the value itself if the cardinality is 1 or to any element of the list if the cardinality is greater than 1.

sim_2 is used for the latter (Section 3.4.3). The similarity function defined for the elements is extended to the attribute's value in exactly the same way as for terminal attributes (Section 3.3.3).

The specification of nonterminal attributes allows a more precise specification of the needed artifacts. Typically, context characteristics are specified using nonterminal attributes. Therefore, if no nonterminal attributes are specified, the context is barely considered or not considered at all at retrieval time. The applicability of suggested artifacts must be evaluated »manually«.

Table 32 through Table 39 show two complex query specifications. One specification references existing instances in the experience base (here, the instance »effort_023« is known by the user) while the other specification references temporary instances. Of course, combinations of references to existing and temporary instances are also allowed.

To avoid circular computations (Section 3.4.3), the similarity function sim_3 is used:

$$sim_3(i, q) = visited(q) \vee in-eb(q)? \; sim_1(i, q) : sim_2(i, q)$$

where the predicate visited(*q*) is true iff *q* refers to either the »main« temporary instance or to a temporary instance which has been involved in a sim_2-computation, and the predicate in-eb(*q*) is true iff *q* refers to an instance in the experience base.

Table 32: Query specification 2 for a GQM measure which is part of the GQM plan »effort_023«.

Concept: GQM Measure Id of temporary instance: "tmp23841"		
Layer	Attribute	Value
artif	id	undefined
	comments	
	definition	
	scale	"ratio"
	unit	"person-month"
	range	undefined
I/F	assumption	
	model	undefined
	data collections procedure	undefined
	questionnaire question	undefined
ctxt	gqm plan	{"effort_023"}

Table 33: Result of query specification 2

Instance	Similarity value
effort_01	1.0000
failure_count_1	$0.633\overline{3}$
fault_count_03	$0.533\overline{3}$

Table 34: Query specification 3 for a GQM measure

Concept: GQM Measure Id of temporary instance: "tmp39234"		
Layer	Attribute	Value
artif	id	undefined
	comments	
	definition	
	scale	"ratio"
	unit	undefined
	range	undefined
I/F	assumption	
	model	undefined
	data collections procedure	undefined
	questionnaire question	undefined
ctxt	gqm plan	{"tmp92387"}

Table 35: Temporary instance used for query specification 3

Concept: GQM Plan Id of temporary instance: "tmp92387"		
Layer	Attribute	Value
artif	id	undefined
	comments	
	gqm goal	undefined
	abstraction sheet	undefined
	gqm question	undefined
	gqm model	undefined
	gqm measure	undefined
ctxt	gqm product experience	"tmp24642"

Table 36: Temporary instance used for query specification 3

Concept: GQM Product Experience Id of temporary instance: "tmp24642"		
Layer	Attribute	Value
artif	id	undefined
	viewpoint	undefined
	representation form	undefined
	owner	
	status	undefined

Concept: GQM Product Experience Id of temporary instance: "tmp24642"		
Layer	Attribute	Value
artif	version	undefined
	last change	undefined
	readers	
	gqm plan	undefined
	measurement plan	undefined
I/F	preconditions for reuse	
ctxt	acquisition technique	undefined
	expected adaptations	
	expected cost of reuse	
	dates of reuse	undefined
	guidelines of reuse	
	comments	
	context	"tmp92653"

Table 37: Temporary instance used for query specification 3

Concept: Context Characterization Id of temporary instance: "tmp92653"		
Layer	Attribute	Value
artif	id	undefined
	organization context	undefined
	project context	"tmp12576"
	measurement context	undefined
ctxt	measurement experience	undefined
	Comments	

Table 38: Temporary instance used for query specification 3

Concept: Project Characterization Id of temporary instance: "tmp12576"		
Layer	Attribute	Value
artif	id	undefined
	project name	undefined
	project start	undefined
	project end	undefined
	duration	12
	team size	undefined
	effort	120
	application of standard software process	undefined
	life-cycle model used	undefined
	tools	undefined

Concept: Project Characterization Id of temporary instance: "tmp12576"		
Layer	Attribute	Value
artif	programming languages	undefined
	estimated product size	undefined
	type of software	undefined
	number of installations	undefined
	memory constraints	undefined
	performance constraints	undefined
	target platforms	undefined
	newness to state of art	undefined
	functionality	undefined
	reliability	undefined
	usability	undefined
	efficiency	undefined
	maintainability	undefined
	portability	undefined
ctxt	project goals	undefined
	context	undefined
	comments	

Table 39: Result of query specification 3

Instance	Similarity value
failure_count_1	$0.888\overline{8}$
effort_01	$0.833\overline{3}$
fault_count_03	$0.755\overline{5}$

As can be seen in specification 3, queries can get quite complex. Therefore, it is useful that specialized browsers provide »short cuts« for frequently used queries. For example, for specification 3 a specialized tool could realize this query by presenting only 2 temporary instances to be filled out by the user.

Filter

Advanced implementations will also allow to filter the result, that is, only those instances will be displayed that have attribute values within the range specified by the *filter*. Filters are a means to discard irrelevant instances. A filter range can be specified by:

- an interval for ordered types
- a set of values for types with a finite value range

User-defined weight

Another means of influencing the result of a query is the specification of *user-defined weights*. If user-defined weights are specified, they will replace the *standard weights* specified in the concept attribute table. How the weights influence the similarity value is defined by the similarity functions. Similarity functions can recall the *actual weight* (user-defined weight if specified; otherwise the standard weight) via the function *p* (see Section 3.7.3).

Actual weight

The weights above have to be distinguished from the weights w_{artif}, $w_{I/F}$, and w_{ctxt} which can also be changed if the system provides a means to do so.

Query specifications using filters or user-defined weights cannot be defined using the standard instance table. Instead, additional columns have to be provided. Table 40 and Table 41 show an example using a modified version of query specification 2. In the result, »fault_count_03« is filtered out and »failure_count_1« is rated slightly more similar due to the greater weight of the artifact layer.

Table 40: Query specification 4 for GQM measures using filters and user-defined weights

Concept: GQM Measure Id of temporary instance: "tmp23042" w_{artif} = 0.7, $w_{I/F}$ = 0, w_{ctxt} = 0.3				
Layer	Attribute	Value	Weight	Filter
artif	id	undefined		
	comments			
	definition			
	scale	"ratio"		"interval"-"ratio"
	unit	"person-month"		
	range	undefined		
I/F	assumption			
	model	undefined		
	data collections procedure	undefined		
	questionnaire question	undefined		
ctxt	gqm plan	{"effort_023"}		

Table 41: Result of query specification 4

Instance	Similarity value
effort_01	1.0000
failure_count_1	0.7000

4.2 Insertion of New Context-Specific Knowledge

New context-specific knowledge is inserted in the form of instances:

Input Set of instances.

Output Either a message telling that the insertion was successful or an error message telling which consistency rule was violated.

Side effects If a consistency rule would be violated, none. Otherwise the context-specific knowledge of the experience base will be extended by the new instances. In addition, all related instances will be updated by a reference to the new instances. If attributes of the inserted instances are part of value inferences, the corresponding value inferences will be performed.

4.2.1 Relation to Software Engineering

Once, context-specific knowledge has been collected, qualified (i.e., it was decided that it should be part of the experience base), and stored (i.e., the artifact itself – not its characterization – is physically stored), a characterization of the artifact has to be provided. In addition, the new artifact has to be integrated (i.e., it has to be specified which relationships exist to other already existing artifacts).

While the characterization is guided by the terminal attributes, the integration is guided by the nonterminal attributes. Both terminal and nonterminal attributes are part of an instance's intension. Thus, the insertion operation supports both the characterization and integration of new artifacts.

4.2.2 Description

New instances are specified using instance tables. Table 42 shows an example. Figure 9 shows the result of the inserted instance.

Table 42: Instance to be inserted in the experience base

Concept: GQM Measure		
Layer	Attribute	Value
artif	id	"test_role"
	comments	undefined
	definition	"records the test role"
	scale	"nominal"
	unit	n/a
	range	…
I/F	assumption	{""}
	model	{"dter_01"}
	data collections procedure	…
	questionnaire question	…
ctxt	gqm plan	{"reliability_01", "reliability_02"}

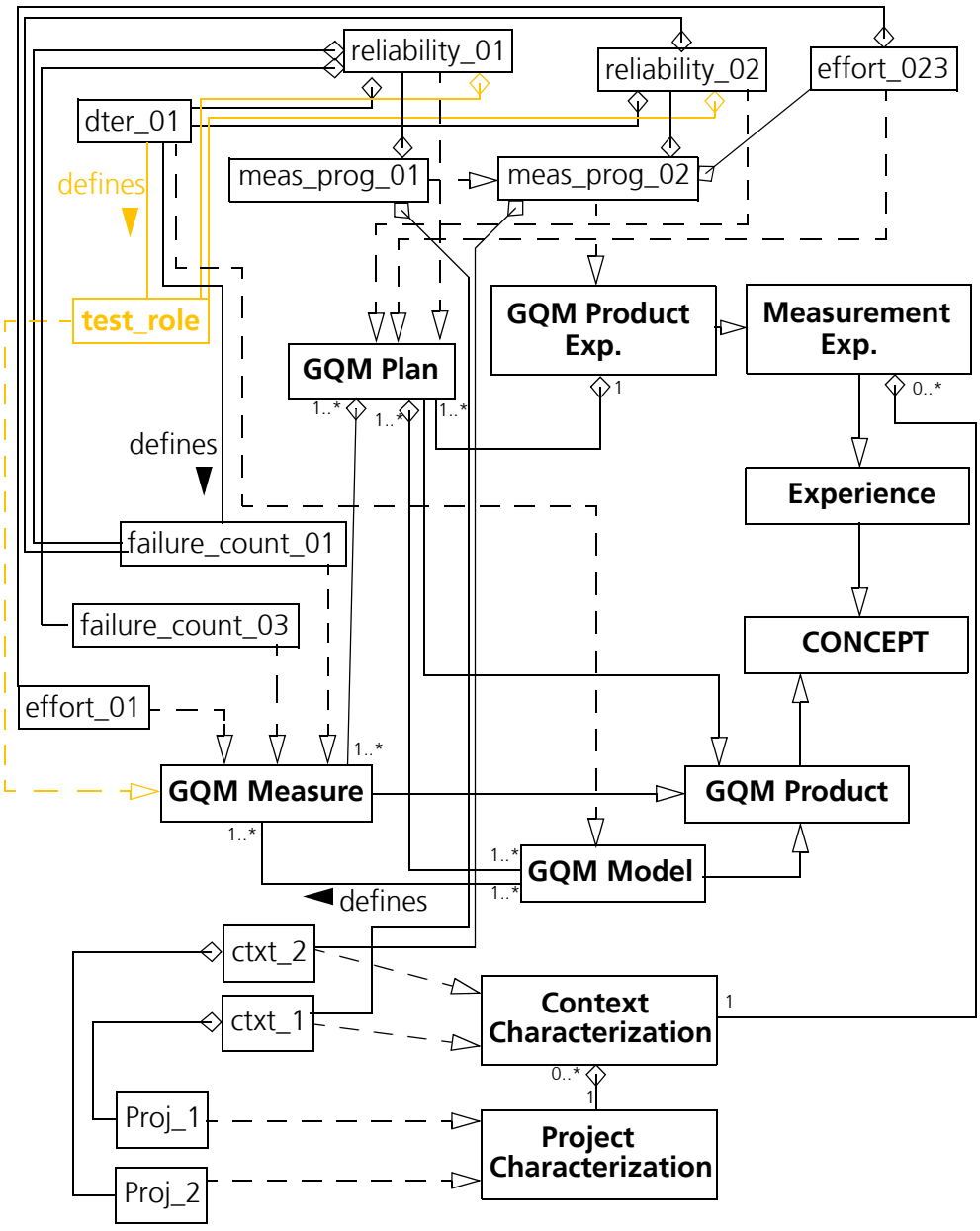


Figure 9: Overview of the contents of the after inserting the new instance

The insertion operation also updates the reverse relationships, i.e., the corresponding nonterminal attribute values of the destination instances. Table 43 gives an example.

Table 43: Automatically updated instance »dter_01«

Concept: GQM Model		
Layer	Attribute	Value
artif	id	"dter_01"
	comments	undefined

Concept: GQM Model		
Layer	Attribute	Value
artif	type	"quality model"
	category	"descriptive"
	definition	"distribution testing effectiveness per role"
I/F	assumption	{"testing is done by maintainers, testers, and users"}
	data source	…
	gqm measure	{"failure_count_1", "test_role"}
	question	{…}
ctxt	gqm plan	{"reliability_01", "reliability_02"}

Several consistency rules have to be fulfilled when inserting new instances. If any of the consistency rules is violated, the experience base remains unchanged. The consistency rules are:

1 **Mandatory attributes.** For every mandatory attribute of an instance's intension a value must be specified.

2 **Cardinality.** The number of values supplied for an attribute must be in the cardinality range as defined in the concept attribute table. If nonterminal attributes are specified, the cardinality range of the affected nonterminal attributes of the referenced instances must also be observed.

3 **Terminal attributes.** Values of terminal attributes must be within the value range of the corresponding type of the attribute. In addition the special values »unknown« and »n/a« (and »undefined« if the attribute is not mandatory) may be used.

4 **Nonterminal attributes.** After the operation, all elements of a nonterminal attribute's value (in case of cardinality > 1) or the value itself (in case of cardinality = 1) must refer to existing instances (i.e., dangling references are not allowed). In addition the special values »unknown« and »n/a« (and »undefined« if the attribute is not mandatory) may be used.

5 **Assertion.** After the operation, all applicable assertions must be true. Applicable assertions are the assertions of the concepts of all affected instances as well as all of the concepts' super concepts.

6 **Precondition.** The precondition must be true for all instances to be inserted *before* the insertion operation is executed. This means that the preconditions are checked before the bidirectional relationships are established.

Consistency rules 1, 2, 4, 5 and 6 place constraints on the order in which instances can be inserted.

Advanced implementation will also provide the capability to copy already existing instances and allow changing the copies. The changed copies will then be inserted into the experience base.

4.3 Removal of Context-Specific Knowledge

Context-specific knowledge is removed by deleting instances:

Input Specification of a set of instances to be deleted

Output Either a message telling that the removal was successful or an error message telling which consistency rule was violated.

Side effects If a consistency rule would be violated, none. Otherwise the specified instances will be removed from the experience base. In addition, all related instances will be updated by removing references to the deleted instance. If attributes of the concept of the deleted instances lead to value inferences, the corresponding value inferences are performed.

4.3.1 Relation to Software Engineering

Removal of context-specific knowledge is not addressed explicitly in the tasks of maturing a software development organization. However, some reorganizations of the experience base will require the removal of outdated knowledge. For instance, if software systems are no longer programmed in Fortran, all code modules written in Fortran may be removed from the experience base. Another example is the availability of a generic artifact. For instance, if a new stack module becomes available that can be easily instantiated for any type, the type-specific stack modules may be removed.

4.3.2 Description

The removal of instances will affect nonterminal attribute values of referenced instances by the instances to be deleted (references to the instances to be removed will be deleted). If the deletion of references in these nonterminal attributes violates any of the consistency rules 1–5, a removal of the instance is not possible. In such a case the removal is not performed.

If the instance to be removed is an aggregate, i.e., it references other instances via an »has-decomposition« nonterminal attribute, the referenced instances are also removed because they are not to exist independently.

An example for the removal of an instance is given by the instance defined in Table 42. If the operation is performed with the experience base contents shown in Figure 9, the resulting contents of the experience base will be those depicted in Figure 8.

4.4 Change of Existing Context-Specific Knowledge

Context-specific knowledge is changed by changing the attribute values of existing instances:

Input Specification of a set of instances to be changed

Output Either a message telling that the change was successful or an error message telling which consistency rule was violated.

Side effects If a consistency rule would be violated, none. Otherwise the specified instances will be changed. In addition, all related instances will be updated by adding and removing references to the changed instance. If attributes were changed which are part of a value inference, the inferences will be performed.

4.4.1 Relation to Software Engineering

As a software artifact is reused, application experience regarding the reused artifact is collected. Such application experience may result in a change of certain attributes of the characterization of the software artifact applied. For instance, if the characterization of a software engineering artifact has a terminal attribute for the timestamps of reuse, each reuse attempt will result in a change of this attribute's value. Also, with each successful application of an artifact, the validity of the artifact increases. If the validity is part of the artifact's characterization, it too would have to be changed.

Just as the removal of context-specific knowledge, the change of context-specific knowledge is not addressed by the tasks for a maturing software development organization. However, changing context-specific knowledge is a must as the examples above show.

4.4.2 Description

The changed instances must comply with all consistency rules stated in Section 4.2.2 (here, consistency rule 6 is applied *before* the change operation). Otherwise the contents of the experience base will not be changed. In addition, the value of a nonterminal attribute of the kind »decomposition-of« may not be changed.

5 Applying REFSENO: Benefits and Lessons Learned

This chapter lists the benefits and lessons learned gained so far by applying REFSENO. It is subdivided into 4 sections mirroring the representation levels introduced in Section 2.2.

5.1 Linguistic Level

Since there existed no system implementing REFSENO completely at the time this report was written, not much experience on the linguistic level has been gained. However, using an implementation of a very restricted version of REFSENO for demonstration purposes revealed:

- Knowledge as defined in Section 2.1 will only be acquired if there is a symbol glossary defining the meaning of each symbol unambiguously. If such a symbol glossary is not available or does not define symbols unambiguously, information supplied by the knowledge sources will consist of statements which are believed to be true by the knowledge source, but are not really true (because the meaning of a symbol varies from user to user).
- The first time, knowledge sources should supply their knowledge in the presence of a knowledge engineer. First results of guided interviews (with the knowledge engineer as the interviewer) were promising. This is probably due to the fact that the knowledge engineer can explain the difference between symbols. This increases the *terminological control* [Gau95] further.

But before knowledge can be acquired at all, the knowledge acquisition has to be initiated, that is, it has to be assured that context-specific knowledge that is to be stored (this is specified by the ontology) is actually collected. For instance, for the ontology defined in Appendix the dates of reuse, expected adaptations, and expected costs of reuse shall be recorded. Rules have to be established that prescribe when to supply the knowledge. One possibility is to update the knowledge when a new version of a GQM entity is checked-in.

5.2 Conceptual Level

At the conceptual level, ontologies are defined. This section takes a closer look at the building, evolution, and validation of ontologies.

5.2.1 Building an Ontology

Just like software systems, ontologies should be built in an engineer-like fashion [GP98, UG96]. At the time this report was written, a set of techniques existed for developing ontologies [GP98, UG96, vHFAH+95]. However, these ontologies do not use constructs known from case-based reasoning and database systems as REFSENO does. Furthermore, the techniques have not been applied for software engineering ontologies. Therefore, specialized techniques for developing software engineering ontologies have to be devised.

A first step is the definition of major documents. Gomez-Perez [GP98] suggests to have a specification of an ontology, the ontology itself (which corresponds to the design of a software system), and an implementation operationalizing the ontology. The latter requires that operations on the knowledge representation are defined [Rei91] as it is done in Chapter 4 for REFSENO.

The specification of an ontology should contain the domain modeled, the purpose of the ontology, the scope, and administrative information like the authors and knowledge sources [GP98]. For software engineering ontologies it has been shown that a refinement of the scope is helpful. The scope should list at least (major) concepts, instances (as far as they are part of the ontology – for a discussion on this topic see introduction to Section 3.6), and attributes common to all concepts. Table 44 shows an example for the ontology defined in Appendix .

Table 44: Ontology requirements specification

Domain	Measurement program planning
Date	June 25, 1998
Conceptualized by	Christiane Gresse von Wangenheim, Carsten Tautz
Purpose	Ontology about GQM entities to be used when information is required for planning a GQM-based measurement program
Level of formality	Semi-formal (REFSENO)
Scope	List of concepts: • GQM entities (Abstraction Sheet, Artifact Event, Context Characterization, Data Collection Event, Data Collection Instrument, Data Collection Procedure, Experience, GQM Goal, GQM Measure, GQM Model, GQM Outcome, GQM Plan, GQM Problem, GQM Problem Cause, GQM Problem Solution Experience, GQM Product, GQM Product Experience, GQM Question, GQM Solution, Item, Measurement Characterization, Measurement Experience, Measurement Plan, Organization Characterization, Periodic Event, Project Characterization, Quality Item, Questionnaire, Questionnaire Question, Variation Item) • Software process entities (Software Object, Tool, Software Attribute, Software Process, Software Product, Role) Instances: *none* Common concept attributes: *none*
Source of knowledge	C. Gresse von Wangenheim, »GQM Domain Model« V1.0, May 15, 1998.

Care should be taken when defining the purpose of the ontology. REFSENO is general enough to be used for structuring any kind of software engineering knowledge. The purpose will determine to which level of detail the domain will be modeled (defined by the scope of the ontology). For instance, in case of GQM, retrieval on the level of GQM plans only would be an alternative to the retrieval of all kinds of GQM entities (as shown in Appendix). The latter suggests to provide logical operations such as »insert GQM Plan« that would insert all instances related to a single GQM plan. Such a specialized operation cannot be provided by a general purpose tool, but must be provided by a special GQM tool.

Chapter 4 defines the basic operations allowing incremental storage. However, for specialized operations it may be necessary to allow not only the insertion of single instances, but also the insertion of a set of instances. The consistency checks would then be performed after the insertion of all instances. Such a complex operation would alleviate the deadlock problem described in Section 4.2.

The scope is the hardest part of the requirements specification to define. The development of usage scenarios helps to determine the scope. The usage scenarios should cover those activities to be supported by the experience base. In order to carry out the activities, knowledge is needed. This needed knowledge must be structured and modeled by the ontology. At the specification stage, concepts are identified that can be justified using usage scenarios (see discussion on purpose of concepts in Section 3.1.3). These concepts are considered to be the *major* concepts of the ontology. More concepts may be added later for modeling purposes.

After the requirements specification has been written, the ontology itself using REFSENO has to be developed. Here is a suggested process model for developing an ontology based on the experience gained thus far:

1 Take the concepts of the scope of the ontology requirements specification and define the concept glossary for them.

2 Identify the semantic relationships between the concepts using the alternate representation for nonterminal concept attributes described in Section 3.4.6. Each time a new kind of relationship is used, define the kind in the table for the kinds of nonterminal attributes (see Section 3.5.4).

3 Through relating concepts, common parts shared by two or more concepts may be identified. These parts should become concepts themselves. These are concepts introduced for modeling reasons. They also have to be defined in the concept glossary.

4 Identify the terminal attributes for all concepts. For each concept define a concept attribute table (see Section 3.2.4). Each time a new type of attribute

is used, define the type in the type table (see Section 3.3.4). If a symbol type is defined, define each of the symbols in the range in the symbol glossary (see Section 3.3.4). If the value can be computed automatically, define a value inference. Define the »to infer« entry for the attributes needed in the computation.

5 Complete the concept attribute tables by the nonterminal attributes. The nonterminal attributes have to be consistent with the graphical representation defined in step 2.

6 Check the completeness of all concept attribute tables: are there attributes describing the artifact itself, its interface, and its context? [BR91] Attributes describing the artifact itself are typically terminal attributes while attributes describing the interface are typically nonterminal attributes whose values reference the interfacing artifacts. The context is described using both terminal (for artifact-specific qualities) and nonterminal attributes (for references to a quality model valid for a class of artifacts, and for references to descriptions of the application domain and development process models).

7 Define instances specified in the requirements specification using instance tables (see Section 3.6.4).

Once an ontology is defined, it has to be implemented. In case of REFSENO, the operations for an implementation are already defined (Chapter 4).

5.2.2 Evolving an Ontology

Over time, the knowledge needs of an organization will change. This requires the evolution of ontologies. Evolving a software engineering ontology means changing the structure of an experience base. Since REFSENO provides epistemological primitives, the implementation does not need to be changed (i.e., the ontology is represented explicitly and interpreted by the implementation). However, the context-specific knowledge of an experience base is based on the structural knowledge the ontology provides. Hence, the context-specific knowledge has to be reorganized with each tailoring of the underlying ontology. Advanced implementations will provide support for this kind of reorganizations. For example, if a new attribute to a concept is added, the attribute values of the concept's instances may be computed automatically (e.g., if a value inference is defined for the attribute). If the value range of a type is changed, the new values may be computed using the old values.

Dynamic symbol types

In addition there may be some »minor« changes to the ontology which should be possible to perform »on the fly«. Extending symbol types is such an example. Often, the complete value range of symbol types is not known at the time an ontology is developed. If a new instance is inserted, it has to be possible to

extend such *dynamic* symbol types.[1] Extending symbol types involves the precise definition of the new symbol in the symbol glossary as well as a redefinition of the similarity function. However, for all symbol types using the default similarity function, the similarity function can be adapted automatically to the extended value range.

5.2.3 Validating an Ontology

The evolution of an ontology is triggered by a continuous validation of the adequacy of an ontology. If the structure of the context-specific knowledge is no longer perceived as adequate, the ontology must be evolved.

In addition, an initial validation should be performed after a new ontology has been developed. This can lead to changes in the ontology before context-specific knowledge is inserted in the experience base. It is important that inadequacies of an ontology are discovered as early as possible because the adaptation of context-specific knowledge may have to be performed manually. The less context-specific knowledge exists the less adaptation effort will be required.

For instance, during the validation of the ontology defined in Appendix [2] it turned out that the symbols for the types »organizational process model« (»no«, »high-level«, »low-level«) and »level of automation« (»high«, »low«) could not be defined unambiguously. Therefore, the symbol type »organizational process model« was changed to boolean and attributes of type »level of automation« were removed.In this case, an automatic adaptation of already existing instances could have been performed (in the former case, »no« would have to be replaced with »false«, »high-level« and »low-level« with »true«; in the latter case, the removal of an attribute would not require any recalculation at all).

In general, measures for evaluating the adequacy of ontologies need to be defined. Such measurement programs may result in data collection during the retrieval of context-specific knowledge.

1 Dynamic symbol types are defined by the keyword »DYNAMIC« in the value range column of the type table.

2 The validation was done in form of a demonstration for reusing GQM plans. This demonstration included only parts of the ontology, but showed nevertheless some improvement potential.

5.3 Epistemological Level

On the epistemological level, knowledge is represented using REFSENO. Therefore, this section focuses on the benefits and preliminary validation results of REFSENO.

5.3.1 Benefits of REFSENO

The benefits of REFSENO include:

- the possibility to model software engineering knowledge explicitly in a precise, consistent, and complete manner using alternate representations.
- the possibility to conceptualize software engineering knowledge explicitly in various application domains and contexts
- a clear terminology differentiating between conceptual and context-specific knowledge enabling the management of knowledge from various contexts
- the possibility to validate conceptual models of software engineering knowledge
- the operationalization of an experience base based on the conceptualization of software engineering knowledge

In the following, the first two benefits will be presented in detail.

Conceptualization of Software Engineering Knowledge

Tools for creating and changing software engineering artifacts use an implicit conceptualization of the artifacts. If more than one tool needs to access a software engineering artifact, they need to *share* the conceptualization requiring the conceptualization to be explicit. The sharing of conceptual knowledge is one of benefits ontologies provide [UG96]. If only concepts and relationships between them need to be modeled, modeling approaches like UML [Cor97] suffice. However, if an experience base is to be developed, additional knowledge such as knowledge on similarity computation and automatic value calculations have to be captured. Existing approaches do not allow this.

REFSENO was developed with these special requirements in mind. Therefore, it allows to describe explicitly all structural knowledge necessary to specify an experience base. The explicit conceptualization (ontology) allows to:

- communicate the structure model of an experience base
- operationalize an experience base by defining operations on the knowledge representation (so done in Chapter 4).

Precise definition

Ontologies specified using REFSENO are precise. For instance, while developing the ontology described in Appendix , the following points of the original domain model could be improved:

- Semantic relationships were defined using narrative text. In REFSENO, the epistemistic primitive »kind of nonterminal concept attribute« is used for this purpose. The epistemistic primitive prescribes what kind of knowledge is needed to define a semantic relationship unambiguously.
- The statement »to represent the interdependencies the GQM entities, in general, have the following attributes …« cannot be expressed in REFSENO. Instead for all pairs of GQM entities the applicable interdependencies have to be defined.

Complete definition

Ontologies specified using REFSENO are complete in the sense that all conceptual knowledge necessary to instantiate an experience base is provided. This is done by using tabular representations for the epistemistic primitives. If an entry in the table is empty, it is not specified – thus the ontology is defined incompletely. For instance, while developing the ontology described in Appendix , the following points of the original domain model could be improved:

- No cardinality was specified for the attributes. REFSENO requires to specify the cardinality as part of the concept attribute specification.
- For some of the semantic relationships no kind was specified. REFSENO requires to specify the kind as part of the nonterminal concept attribute specification.
- No similarity functions were defined. REFSENO requires to specify type similarity functions for each type and concept similarity functions for each concept. If similarity functions are not specified, the retrieval of similar software engineering artifacts is not possible.

Consistent definition

Ontologies specified using REFSENO are consistent in the sense that certain consistency criteria have to be fulfilled. Some of these consistency rules can be enforced automatically:

- No two concepts may have the same names.
- No two attributes of a concept may have the same names. Polymorphism is not allowed, that is, if a concept is a specialization of another, it may not redefine any of the inherited attributes.
- No two types may have the same name.
- No two kinds may have the same name.
- No two instances may have the same name.
- Default values have to be within the value range of the attribute's type.
- Formulas for value inferences have to be type compatible with the attribute values they infer.
- Value inferences and *inferred attributes* must match. Advanced systems will compute the *inferred attributes* component automatically.

- Graphical representation of formulas must match formula. Advanced systems will construct the graphical representation automatically.
- Graphical representation of type hierarchy must match the type definitions. Advanced systems will construct the graphical representation automatically.
- For every nonterminal concept attribute there has to exist a corresponding reverse nonterminal concept attribute (i.e., all semantic relationships are bidirectional).
- The destination concept of a nonterminal attribute must be defined.
- Graphical representation of nonterminal concept attributes must match the tabular representation. Advanced systems will generate a graphical representation for each kind automatically.
- The syntax of formulas has to be correct.

For instance, while developing the ontology described in Appendix , the following points of the original domain model could be improved:

- »Question«/»GQM Question«, »Model«/»GQM Model«, and »Measure«/ »GQM Measure« were used as synonyms. This was discovered by using the graphical representation of relationships, but it could have been discovered automatically, too, because the destination concepts of some nonterminal attributes did not exist.

Alternate representations

Alternate representation allow to view a conceptualization from different viewpoints. For instance, the alternate (graphical) representation of nonterminal concept attributes can be used to overview the structural knowledge of an experience base. Such an overview can be used to talk about a conceptualization. This in turn helps to find inconsistencies and modeling errors early. For instance, while developing the ontology described in Appendix , the following points of the original domain could be improved:

- It was not clear whether questionnaires are a specialization or a part of data collection instruments (the different kinds of relationships were specified in different sections), i.e., the concept »data collection instruments« was used with different meanings. This inconsistency was discovered while using the graphical representation of semantic relationships because between the two concepts there was a »is-a« as well as a »has-part« relationship which did not make sense.
- The relationship from »GQM question« to »GQM measure« was not modeled because it would have duplicated knowledge and, thus, opened the way for storing inconsistent context-specific knowledge. The »GQM measure« of a »GQM question« can be accessed via navigation: »GQM question« to »GQM model« to »GQM measure«. Alternatively it could have been modeled using a value inference.
- For the specification of a data collection instrument, both a taxonomy and a reference to a detailed characterization was used for the same concept. It was decided to use the reference.

Clear Terminology

The clear distinction of conceptual and context-specific knowledge allows to define whether knowledge has to be defined *once* on the conceptual level or *for each instance* on the linguistic level. Knowledge to be supplied only once is specified by REFSENO, knowledge to be supplied for every instance is specified by an ontology. For example, while developing the ontology described in Appendix , the following points of the original domain model could be improved:

- The purpose of capturing a GQM entity and guidelines for reuse were originally modeled as attributes of a GQM entity. However, this is part of the structural knowledge because purpose and guidelines for reuse are specified using a usage scenario. REFSENO requires this structural knowledge as part of the concept glossary.

5.3.2 Validation of REFSENO

By defining the ontology of Appendix using REFSENO based on an independently developed domain model, REFSENO was validated for the first time. Some of the lessons learned have already contributed to the definition of REFSENO as it is presented in this report:

- The existence of concepts has to be justified. Description, purpose and intended user(s) specify for every concept »who and how the concept's instances will be used for what purpose«. This avoids a proliferation of concepts. Only concepts which are meaningful in the context of an experience base are defined.
- The existence of all kinds of relationships has to be justified. This avoids proliferation of relationship kinds. Only those relationships that have a meaning that can be precisely specified are introduced. Furthermore, the meaning and properties of all relationships have to be defined. This allows special operations to be defined based on the meaning (and properties). For example, a »has-parts« relationship can be used to generate reports automatically.
- A glossary for symbol types is required in order to avoid misunderstandings.
- Symbol sets may not contain the same symbol more than once. This restriction was extended to all sets, that is, to values of attributes with a cardinality > 1.[1]

1 It remains to be seen whether this restriction is not too restrictive. However, up to now no multiple sets needed to be modeled for software engineering knowledge.

The rework of the first ontology version (for GQM planning) led to further extensions of REFSENO:

- Addition of assertion component to the primitive »concept«.
- Addition of precondition component to the primitive »concept«.
- Addition of interval types.

It is expected that these changes of REFSENO will not be the last ones. The following changes are likely as more and more ontologies are defined using REFSENO:

- definition of more basic types
- definition of more functions for formulas
- for formulas: definition of an iterator and sequential computations using variables[1]
- specification of concept similarity functions: in the current version of REFSENO, user-defined definitions of concept similarity functions tend to be long and complex. On the other hand, up to now the standard concept similarity function was sufficient. It is not clear for which situations a user-defined similarity function is needed. Therefore, more experience needs to be gained before a redefinition of specifying concept similarity functions is possible.
- integration of views: every user has different knowledge needs depending on the task he is working on. Consequently, the conceptual knowledge used to guide the user at retrieval time should depend on the task to be performed. This can be modeled using views. However, experience regarding the difference between views (in terms of epistemological primitives) still needs to be collected. Moreover, it is unclear how beneficial a view mechanism would be, because persons using different views could no longer communicate about the conceptualization as a whole. Therefore, more experience on the adequacy of the current version of REFSENO and required extensions (for specific tasks) needs to be gained.

5.4 Implementation Level

As there existed no system implementing REFSENO completely at the time this report was written, only little experience has been gained at this level. However, it is clear that a commercial tool implementing an experience base based on REFSENO must have mechanisms for:

1 This would allow the specification of algorithms. Consequently the term »formula« should be replaced by »algorithm«. However, the decision on this extension should be made with care because algorithms can be complex in terms of time. Since value inferences are initiated while a user interacts with the system, the performance of a system implementing REFSENO may decrease dramatically.

- **Access rights.** »Knowledge is power«. Therefore, people and organizations are very fond of protecting their knowledge. This can only be realized through access rights. Who may access context-specific knowledge is determined by the:
  - project/department/organization people work in (project-specific or organization-specific knowledge)
  - role people play (e.g., project manager, software developer)
  - knowledge source (e.g., effort data may only be changed by the person supplying it)
- **Evolution of the underlying ontology.** The evolution of ontologies has already been discussed in Section 5.2.

6 Summary and Outlook

In this report, the representation formalism REFSENO for software engineering ontologies has been defined. Ontologies conceptualize structural software engineering knowledge explicitly. Such explicit conceptualizations can serve as a means for communicating about adequate structure models for experience bases. Through clearly defined epistemistic primitives and operations on the conceptual knowledge representation, REFSENO provides in addition a means for operationalizing an experience base based on a conceptualization of structural software engineering knowledge.

Ontologies specified using REFSENO are defined precisely, completely (in the sense that all conceptual knowledge necessary to operationalize an experience base for the modeled »real world« entities is supplied), and consistently. Therefore, REFSENO also provides the means for checking structural software engineering knowledge for completeness and consistency.

The epistemistic primitives of REFSENO constitute a notation for software engineering ontologies. However, REFSENO does not address on *how* to build ontologies (i.e., how to acquire conceptual knowledge). A coarse sequence of tasks to be performed has been described in Section 5.2.1. But further research on defining an »engineering process« for building and evolving ontologies is necessary.

Another issue to be addressed is the validation of REFSENO. As more and more software engineering ontologies are built, additional characteristics may be identified that need to be modeled in order to get an adequate (i.e., easy-to-use) operationalization of an experience base. Therefore, it is expected that REFSENO will evolve. Already defined epistemistic primitives are validated through the application of REFSENO for software engineering ontologies.

Finally, ontologies themselves must be validated. This requires to conduct case studies using ontologies specified using REFSENO. Such case studies will show whether the ontologies are adequate. If they are, this also validates REFSENO because then REFSENO obviously allows to specify adequate ontologies. However, at the present more research needs to be done to determine how adequacy can be measured and – more importantly – how it can be determined in what way to change the ontology (or REFSENO as its underlying representation formalism).

7 Acknowledgments

The authors would like to thank Klaus-Dieter Althoff and Markus Nick for many fruitful discussions and reviewing an earlier version of this report.

References

[Alt97] Klaus-Dieter Althoff. Evaluating case-based reasoning systems: The Inreca case study. Postdoctoral thesis (Habilitationsschrift), University of Kaiserslautern, July 1997.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Publishing Co., Reading: MA, 1986.

[BAB+87] Bruce A. Burton, Rhonda Wienk Aragon, Stephen A. Bailey, Kenneth D. Koehler, and Lauren A. Mayes. The reusable software library. *IEEE Software*, 4(7):25–33, July 1987.

[BCR94] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Experience Factory. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 469–476. John Wiley & Sons, 1994.

[BR91] Victor R. Basili and H. Dieter Rombach. Support for comprehensive reuse. *IEEE Software Engineering Journal*, 6(5):303–316, September 1991.

[Che76] P. P. Chen. The Entity-Relationship Model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[Cor97] Rational Software Corporation. *Unified Modeling Language*, 1997. Version 1.1.

[DBSB91] Premkumar Devanbu, Ronald J. Brachman, Peter G. Selfridge, and Bruce W. Ballard. LaSSIE: a knowledge-based software information system. *Communications of the ACM*, 34(5):34–49, May 1991.

[FG90] W. B. Frakes and P. B. Gandel. Representing reusable software. *Information and Software Technology*, 32(10):653–664, December 1990.

[Gäß95] Sven Gäßner. A language for the specification of characterization schemes and similarity measures for the purpose of reuse (in German). Master's thesis, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, February 1995.

[Gau95] Wilhelm Gaus. *Documentation and Classification Science (in German)*. Springer-Verlag, Berlin, 1995.

[GB97] Christiane Gresse and Lionel Briand. Requirements for the Knowledge-Based Support of Software Engineering Measurement Plans. In *Proceedings of the Ninth International Software Engineering and Knowledge Engineering Conference (SEKE'97)*, pages 559–568, Madrid, Spain, June 1997.

[GP98] Asunción Gómez-Pérez. Knowledge sharing and reuse. In Jay Liebowitz, editor, *The Handbook of Applied Expert Systems*. CRC Press, 1998.

[GRA+98] Christiane Gresse von Wangenheim, Alexandre Moraes Ramos, Klaus-Dieter Althoff, Ricardo M. Barcia, Rosina Weber, and Alejandro Martins. Case-based reasoning approach to reuse of experiential knowledge in software measurement programs. In Lothar Gierl, editor, *Proceedings of the 6th German Workshop on Case-Based Reasoning*, Berlin, Germany, 1998.

[Gru93] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Hen97] Scott Henninger. Capturing and formalizing best practices in a software development organization. In *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering*, pages 24–31, Madrid, Spain, June 1997.

[MBY97] John Mylopoulos, Ales Borgida, and Eric Yu. Representing Software Engineering Knowledge. *Automated Software Engineering*, (4):291–317, 1997.

[OB92] Markku Oivo and Victor R. Basili. Representing software engineering models: The TAME goal oriented approach. *IEEE Transactions on Software Engineering*, 18(10):886–898, October 1992.

[Ost92] Eduardo Jenkins Ostertag. *A Classification System for Software Reuse*. Dissertation, University of Maryland, 1992.

[PDF87] Rubén Prieto-Díaz and Peter Freeman. Classifying software for reusability. *IEEE Software*, 4(1):6–16, January 1987.

[Rei91] Ulrich Reimer. *Introduction to Knowledge Representation: Net-like and Schema-Based Representation Formats (in German)*. Leitfäden der angewandten Informatik. Teubner, Stuttgart, Germany, 1991.

[Rom96] H. Dieter Rombach. New institute for applied software engineering research. *Software Process Newsletter*, pages 12–14, Fall 1996. No. 7.

[SM83] Salton and McGill. *Introduction to Modern Information Retrieval*, chapter Retrieval Evaluation, pages 157–198. McGraw Hill, 1983.

[SPDM94] Wilhelm Schäfer, Rubén Prieto-Díaz, and Masao Matsumoto. *Software Reusability*. Ellis Horwood, 1994.

[Stu95] Jens Stummhöfer. A direct manipulatable user interface for maintaining and querying an experience base (in German). Master's thesis, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, June 1995.

[SWT89] J. Solderitsch, K. Wallnau, and J. Thalhamer. Constructing domain-specific ada reuse libraries. In *Proceedings of the Seventh Annual National Conference on Ada Technology*, Ft. Monmouth, NJ, March 1989. U.S. Army Communications-Electronics Command.

[TA97] Carsten Tautz and Klaus-Dieter Althoff. Using case-based reasoning for reusing software knowledge. In D. Leake and E. Plaza, editors, *Proceedings of the Second International Conference on Case-Based Reasoning Research and Development, (ICCBR97)*. Springer Verlag, 1997.

[Tau93] Carsten Tautz. Design and implementation of a tool for comprehensive reuse of software processes (in German). Master's thesis, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, July 1993.

[UG96] Mike Uschold and Michael Gruninger. Ontologies: Principles, methods, and applications. *The Knowledge Engineering Review*, 11(2):93–136, 1996.

[vHFAH+95] G. van Heijst, S. Falasconi, A. Abu-Hanna, A. Th. Schreiber, and M. Stefanelli. A case study in ontology library construction. *Artificial Intelligence in Medicine*, 7:227–255, 1995.

[ZS95] Mansour Zand and Mansur Samadzadeh. Software reuse: Current status and trends. *Journal of Systems and Software*, 30(3):167–170, September 1995.

Appendix A: Example Ontology

A.1 Concept Glossary

Table 45: Concept Glossary

Name	Description	Purpose	Intended user(s)
Abstraction Sheet	simplified version of the GQM plan serving as an interface between quality assurance personnel and viewpoints.	Usage scenario: - query for relevant abstraction sheet in order to prepare GQM interviews	quality assurance personnel
Artifact Event	defines a data collection event wrt. the state transition of an artifact, e.g. competition of design document.	Modelling: explicit representation of artefact and its state defining the event	quality assurance personnel
Context Characterization	describes the context of a GQM measurement program concerning its organizational, project specific and measurement specific environment.	Modelling: explicitly states the context from which the knowledge originates	experience engineer
Data Collection Event	defines a data collection event subsuming different types of events	Modelling: generalization of data collection events (Artefact Event, Periodic Event and Process Event)	quality assurance personnel
Data Collection Instrument	defines how the data is to be collected subsuming different types of instruments	Modeling: generalization of data collection instruments (Tool, Questionnaire, Interview)	quality assurance personnel
Data Collection Procedure	determines when, how, and by whom the data is collected	Usage scenario: - query for relevant a data collection procedure for a GQM measure	quality assurance personnel
Experience	describes a software engineering experience	Modelling: generalizes all experiences (e.g. on measurement, inspections)	quality assurance personnel, experience engineer
GQM Goal	specifies a goal to be achieved by the measurement program	Usage scenario: - query for relevant GQM goals in a specific context Modelling: important for the reuse of appropriate GQM plans	quality assurance personnel
GQM Measure	A GQM measure is an operational definition of an attribute. The data collected according to the measures are used by a model to answer the question in the GQM plan.	Usage scenario: - support development of GQM plan by supplying adequate measures for a model	quality assurance personnel
GQM Model	Models are used to define how to combine and compute the data measured in order to answer the questions.	Usage scenario: - support development of GQM plan by supplying adequate models for questions	quality assurance personnel

Name	Description	Purpose	Intended user(s)
GQM Outcome	The resulted outcome of a solution applied to a problem occurred during the GQM planning process.	Usage scenario: - anticipate expected outcome in the future reusing the solution	quality assurance personnel
GQM Plan	A GQM plan contains information necessary to motivate and define measures and interpret measurement data. Elementary components are GQM goal, questions, models and measures.	Modelling: structures GQM products	quality assurance personnel
GQM Problem	describes a problem occurred during the GQM planning process	Usage scenario: - points out potential problems in future measurement program planning Modelling: important for the reuse of appropriate GQM Problem Solution Experiences	quality assurance personnel
GQM Problem Cause	describes the cause of a problem occurred during the GQM planning process	Usage scenario: - prevention of potential problems in future measurement program planning	quality assurance personnel
GQM Problem Solution Experience	describes experiential knowledge on the GQM planning process primarily focusing on a specific problem and its applied solution.	Usage scenario: - prevention of potential problems - solution of existing problems in future measurement program planning	quality assurance personnel
GQM Product	describes products developed by the GQM process, e.g. GQM plan, data collection instrument.	Modelling: generalizes all GQM products	quality assurance personnel
GQM Product Experience	documents the planning of a measurement program by representing the related GQM products	Usage scenario: - support the planning of a GQM measurement program Modelling: structures all GQM (planning) products related to a measurement program	quality assurance personnel
GQM Question	A set of questions operationally define the measurement goal, expressing the respective need for information in natural language.	Usage scenario: - support development of GQM plan by supplying adequate questions refining the goal	quality assurance personnel
GQM Solution	describes the solution applied to a problem occurred during the GQM planning process.	Usage scenario: - solution of existing problems in future measurement program planning	quality assurance personnel
Item	describes a refinement of the quality focus on the Abstraction Sheet in natural language	Modelling: generalizes the items of the Abstraction Sheet (quality item, variation item)	quality assurance personnel
Measurement Characterization	characterizes the environment where the measurement program takes place, focusing especially on characteristics, constraints, etc. regarding measurement.	Modelling: explicitly states the measurement context from which the knowledge originates	experience engineer

Name	Description	Purpose	Intended user(s)
Measurement Experience	describes experiences on measurement	Modelling: generalizes experiences on measurement (GQM Product Experience, GQM Problem Solution Experience)	quality assurance personnel
Measurement Plan	contains data collection procedures and data collection instruments of a measurement program	Modelling: structures GQM products	quality assurance personnel
Organization Characterization	describes the organizational context. (The term organization is here used for any type of commercial, industrial or public institution covering various levels, e.g., company, division, or department level)	Modelling: explicitly states the organizational context from which the knowledge originates	experience engineer
Periodic Event	defines a periodic data collection event, e.g. weekly.	Modelling: explicit representation of period defining the event	quality assurance personnel
Process Event	defines a data collection event wrt. a process, e.g. end of requirement analysis.	Modelling: explicit representation of process defining the event	quality assurance personnel
Project Characterization	characterizes a specific software project	Modelling: explicitly states the organizational context from which the knowledge originates	experience engineer
Quality Item	describes a quality factor stated on the Abstraction sheet refining the quality focus of the GQM goal	Modelling: structures the quality factors of the quality focus on the Abstraction sheet	quality assurance personnel
Questionnaire	set of questions to be used for the collection of data	Modelling: structures Questionnaire Questions on the Questionnaire	quality assurance personnel
Questionnaire Question	a question on a particular questionnaire for the collection of data	Usage scenario: - facilitates the design of questionnaires given a measure and measurement procedure	quality assurance personnel
Software Object	describes any object of software process, product or resource by its attributes	Modelling: explicit representation of software objects, e.g. data sources	quality assurance personnel
TCardinal	A cardinal is an integer greater or equal to zero.	Modelling: representation of cardinal types	experience engineer
TGlossaryEntryOrdered	Symbol and Description for TOrderedSymbol	Modelling: glossary entry for ordered symbols	experience engineer
TGlossaryEntryTax	Symbol and Description for TTaxonomyRoot	Modelling: glossary entry for a taxonomy	experience engineer
TGlossaryEntryUnordered	Symbol and Description for TUnorderedSymbol	Modelling: glossary entry for unordered symbols	experience engineer
TInteger	An TInteger is a number out of the set of natural numbers.	Modelling: representation of integer types	experience engineer
TIntegerInterval	TIntegerInterval describes an interval. The lower bound is out of a left interval and the higher bound is out of a right interval.	Modelling: representation of integer interval types	experience engineer

Name	Description	Purpose	Intended user(s)
Tool	describes tools used during the software process, e.g., CASE tools, compilers, debuggers, editors or measurement process, e.g. data collection tools.	Modelling: explicit representation of tools	quality assurance personnel
TOrderedSymbol	Ordered symbol is the type for a set of symbols with a total order.	Modelling: explicit representation of cardinal values	experience engineer
TReal	A real-number is an element out of the set of the real numbers.	Modelling: representation of real types	experience engineer
TRealInterval-Value	The value of an instance of TRealInterval	Modelling: representation of real interval types	
TTaxonomy	The supertype for TTaxonomyNode and TTaxonomyRoot	Modelling: representation of taxonomy types	experience engineer
TTaxonomy-Node	A node of a Taxonomy.	Modelling: explicit representation of taxonomy nodes	experience engineer
TTaxonomy-Root	Taxonomy is a collection of hierarchical ordered Symbols. TTaxonomyRoot is the root of the hierarchical tree	Modelling: explicit representation of the root node of a taxonomy	experience engineer
TText	A not limited string	Modelling: representation of String type	experience engineer
TType	The supertype of all the types	Modelling: represents an arbitraty type	experience engineer
Variation Item	defines a variation factor with potential impact on the quality factors wrt. the GQM goal	Modelling: structures the variation factors on the Abstraction sheet	quality assurance personnel

A.2 Terminal And Nonterminal Concept Attributes

Table 46: Terminal, Nonterminal Concept Attributes

Concept: Abstraction Sheet
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	quality item	refinement of the quality focus of the GQM goal	1..*	has-parts [Quality Item].[abstraction sheet]	-	yes	-	-	1
	variation item	set of relevant variation factors with expected impact on quality items	0..*	has-parts [Variation Item].abstraction sheet]	-	no	-	-	1
	baseline hypothesis	states the expected values of quality item(s)	0..*	Text	-	no	-	-	0
	impact on baseline hypothesis	states the expected impact of variation item on baseline hypothesis of quality item	0..*	Text	-	no	-	-	0
I/F	gqm goal	GQM goal to which the knowledge documented in the Abstraction Sheet refers	1	Text	-	yes	-		0
ctxt	gqm plan	references the respective GQM Plan	1	part-of [GQM Plan].[abstraction sheet]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Artefact Event
Super concept: Data Collection Event

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	artefact state	describes if the state	1	State	-	yes	-	-	1
	Comments	any additional information or comment	0..1	Text	-	no	-	-	0

Concept: Artefact Event
Super concept: Data Collection Event

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
I/F	artifact	describes the artifact which state transition triggers data collection	1	SWProductTaxonomy	-	yes	-	-	1
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Context Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	organization context	characterizes the organization in which the measurement program takes place	1	has-parts [Organization Characterization]	-		-	-	
	project context	characterizes the specific software project in which the measurement program takes place	1	has-parts [Project Characterization]	-		-	-	
	measurement context	characterizes the specific characteristics, constraints etc. concerning the measurement program	1	has-parts [Measurement Characterization]	-		-	-	
	Comments	any additional information or comment	0..1	Text	-		-	-	
I/F									

Concept: Context Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt	measurement experience	references the respective Measurement Experience	0..1	part-of [Measurement Experience].[context]	-		-	[Measurement Characterization]: [number of goals], [Measurement Characterization]: [number of questions], [Measurement Characterization]: [number of measures]	

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Data Collection Event
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	data collection procedure	references the respective Data Collection Procedure	1	part-of [Data Collection Procedure].[event]	-	yes	-	-	1
	Comments	any additional information or comment	0..1	Text	-	no	-	-	0
I/F									

Concept: Data Collection Event
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Data Collection Instrument
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	data collection procedure	references the respective Data Collection Procedure	0..1	defined-by [Data Collection Procedure].[instrument]	-	no	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0
I/F	measurement plan	references the respective measurement plan	0..1	part-of [Measurement Plan].[data collection instruments]	-	no	-	-	1
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Data Collection Procedure
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	resource	states whether the data is derived by questioning or interviewing a person or by invoking a tool	1	Resource	-	yes	-	-	1
	collector	states role or position of people in the organization by whom the data is collected	0..1	Role	-	no	-	-	1
	validator	specifies who validates the collected data	1	Role	-	yes	-	-	1
	active	states whether data is currently collected or not	1	Boolean	-	yes	-	-	1
	data storage	defines where the data is stored	0..*	Data Store	-	no	-	-	1
I/F	event	specifies when the data will be collected	1	has-parts [Data Collection Event].[Data Collection Procedure]	-	yes	-	-	1
	data source	specifies the object and its respective attribute to be measured	1	refers-to [Software Object]	-	yes	-	-	1
	dependency	states on which other measure(s) the collection of this measure depends	0..*	(depends-on [Data Collection Procedure]	-	no	-	-	1
	instrument	references the data collection instrument which is used to collect the corresponding data	1	(defines [Data Collection Instrument].[Data Collection Procedure)	-	yes	-	-	1

Concept: Data Collection Procedure
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt	measure	references the respective GQM Measure of the GQM Plan	1..*	defined-by [GQM Measure].[data collection procedure]	-	yes	-	-	1
	measurement plan	references the respective Measurement Plan	1	part-of [Measurement Plan].[data collection procedure]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Experience
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	viewpoint	states the role from whom the experience captured was acquired	0..*	Role	–	no	-	-	1
	representation form	states in which form the experience is captured	0..1	RepresentationForm	–	no	-	-	1
	owner	specifies who is responsible for maintaining the experience	0..1	Text	–	no	-	-	0
	status	specifies the current status of the experience	1	Status	"non-existent"	yes	-	-	1
	version	specifies the version of the experience	0..1	Version	–	no	-	-	1
	last change	specifies when the experience was modified last	1	Date	–	yes	-	-	1
	readers	specifies who is allowed to have read access (the owner has per default read and write access)	0..*	Text	–	no	-	-	0

Concept: Experience
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
I/F	preconditions for reuse	describing necessary preconditions for the reuse of the entity	0..*	Text	-	no	-	-	0
ctxt	acquisition technique	specifies how the entity was derived	0..*	AcquisitionTechnique	–	no	-	-	1
	expected adaptations	describing adaptations done in the past when reusing the entity and the relevant factors which caused the adaptations	0..*	Text	-	no	-	-	0
	expected cost of reuse	describing the expected cost of reusing the entity as a basis to decide whether the entity should be reused or rather be developed from scratch.	0..*	Text	-	no	-	-	0
	dates of reuse	in order to provide an overview on when and how often this entity was reused, facilitating also the removal of entities of the knowledge base, which are never used	0..*	Date	-	no	-	-	1
	guidelines of reuse	on how to reuse the entity	0..*	Text	-	no	-	-	0
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Goal
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	object of study	defines the object to be analyzed	1	SwObjectTaxonomy	-	yes	-	-	1
	purpose	states why the object will be analyzed	1	Purpose	-	yes	-	-	1
	quality focus	specifies which property of the object will be analyzed	1	SWQualityTaxonomy	-	yes	-	-	1
	viewpoint	expresses who will use the data collected and analysis results	1..*	Role	-	yes	-	-	1
I/F	quality item	enumerates questions defined by this goal	1..*	defines [Quality Item]	-	yes	-	-	1
ctxt	context	identifies the context in which the analysis takes place	1	Text	-	yes	-	-	0
	gqm plan	references respective GQM Plan	1	part-of [GQM Plan].[gqm goal]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Measure
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	definition	defines what data has to be collected	1	Text	-	yes	-	-	0
	scale	defines the scale of the measure	1	Scale	-	yes	-	-	1
	unit	declares the unit of the measure	0..1	Unit	-	no	-	-	1
	range	declares the range of the values of the measures	0..1	has-parts[TType].[GQMMeasure-Range]	-	no	-	-	1

Concept: GQM Measure
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
I/F	assumption	states assumptions about the environment for the applicability of the measure	0..*	Text	-	no	-	-	0
	model	references the corresponding model	1..*	defined-by [GQM Model].[gqm measure]	-	yes	-	-	1
	data collections procedure	references the corresponding data collection procedure	1	defines [Data Collection Procedure].[measure]	-	yes	-	-	1
	questionnaire question	references the corresponding question on the questionnaire, if collected by a questionnaire	0..1	defines [Questionnaire Question].[measure]	-	no	-	-	1
ctxt	gqm plan	references the respective GQM Plan	1..*	part-of [GQM Plan].[gqm measure]	-	yes	-	-	1

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Model
Super concept: GQM Plan

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	type	specifies if the model focuses on resources or qualities	1	Model-Type	-	yes	-	-	1
	category	specifies the category of the model	1	Model-Category	-	yes	-	-	1
	definition	defines the abstract concepts the model expresses	1	Text	-	yes	-	-	0

Concept: GQM Model
Super concept: GQM Plan

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
I/F	assumption	states assumptions about the environment for the application of the model	0..*	Text	-	no	-	-	0
	data source	defines the attributes to be measured as "input" of the model	1..*	refers-to [Software Object]	-	yes	-	-	1
	gqm measure	defines the model operationally denoting data that can be directly collected	1..*	defines [GQM Measure].[model]	-	yes	-	-	1
	question	references the corresponding question	1..*	defined-by [GQM Question].[model]	-	yes	-	-	1
ctxt	gqm plan	references the respective GQM Plan	1..*	part-of [GQM Plan].[gqm model]	-	yes	-	-	1

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Outcome
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	assessment	states explicitly if the problem was successfully solved by the solution or failed	1	Assessment	-	yes	-	-	1
	failure explanation	If the applied solution failed to solve the problem, an explanation is given on why the goal of the respective task was still not achieved	1..*	Text	-	yes	-	-	0

Concept: GQM Outcome
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
I/F	results	describes the results of the solution applied by the state of the object stated in the problem description after the application of the solution	1..*	refers-to [Software Object]	-	yes	-	-	1
	next gqm-pse	If the applied solution failed to solve the problem, the next attempt to solve the problem, stored as a new case in the experience base, is referenced, e.g. *case_43*.	0..1	next [GQM Problem Solution Experience]	-	no	-	-	1
ctxt	gqm problem solution experience	references the respective GQM Problem Solution Experience	1	part-of [GQM Problem Solution Experience].[outcome]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Plan
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	gqm goal	goals to be achieved by the measurement program	1	has-parts [GQM Goal].[gqm plan]	-	yes	-	[Measurement Characterization]: [number of goals]	1
	abstraction sheet	summarizes the GQM plan in a simplified form	1..*	has-parts [Abstraction Sheet].[gqm plan]	-	yes	-	-	1
	gqm question	expresses the information need wrt. the GQM goal	1..*	has-parts [GQM Question].[gqm plan]	-	yes	-	[Measurement Characterization]: [number of questions]	1
	gqm model	operationalizes the GQM Question	1..*	has-parts [GQM Model].[gqm plan]	-	yes	-	-	1
	gqm measure	specifies data to be collected	1..*	has-parts [GQM Measure].[gqm plan]	-	yes	-	[Measurement Characterization]: [number of measures]	1
I/F									
ctxt	gqm product experience	references the respective GQM Product Experience	1	part-of [GQM Product Experience].[gqm plan]	-	yes	-	-	1

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Problem
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	problem object state	the object affected by the problem and its state causing the problem are explicitly stated	1	refers-to [Software Object]	-	yes	-	-	1
	object type	the type of the objects affected by the problem is stated explicitly	1	SWObjectType	-	yes	-	-	1
	problem task	the task in which the problem occurred is stated	1	GQMProcessTaxonomy	-	yes	-	-	1
	problem role	roles of the software organization involved in the problem are listed, e.g., *developer*.	0..*	Role	-	no	-	-	1
	goal unattained	the goal of the respective gqm task which has not been attained because of the problem is stated	0..1	Text	-	no	-	-	0
I/F	gqm problem solution experience	references the respective GQM Problem Solution Experience	1	part-of [GQM Problem Solution Experience].[problem]	-	yes	-	-	1
ctxt	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Problem Cause
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	explanation	for each cause is provided in order to explain the relation between the problem and the stated cause	0..1	Text	-	no	-	-	0
	cause task	the task causing the actual problem, which can be different from the task of problem occurrence, is identified	0..1	GQM-ProcessTaxonomy	-	no	-	-	1
	cause role	roles of the organization involved in causing the problem are stated	0..*	Role	-	no	-	-	1
	constraint	constraints wrt. the software project which influenced the problem	0..*	Text	-	no	-	-	0
I/F	cause object state	describes the cause of the problem by the respective object and its state	1	refers-to [Software Object]	-	yes	-	-	1
ctxt	gqm problem solution experience	references the respective GQM Problem Solution Experience	1	part-of [GQM Problem Solution Experience].[cause]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Problem Solution Experience
Super concept: Measurement Experience

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	problem	describes the problem occurred during the planning of a GQM-based measurement program	1	has-parts [GQM Problem].[gqm problem solution experience]	-	yes	-	-	1
	solution	describes the solution strategy adopted	1	has-parts [GQM Solution].[gqm problem solution experience]	-	yes	-	-	1
	outcome	describes the outcome resulting of the solution applied	1	has-parts [GQM Outcome].[gqm problem solution experience]	-	yes	-	-	1
I/F									
ctxt	cause	describes the cause(s) of the problem, if known	0..*	has-parts [GQM Problem Cause].[gqm problem solution experience]	-	no	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Product
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif									
I/F									
ctxt	Comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Product Experience
Super concept: Measurement Experience

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	gqm plan	describes the GQM goal, the corresponding abstraction sheet and the refinement of the GQM goal into questions, models, and measures	1..*	has-parts [GQM Plan].[gqm product experience]	-	yes	-	[Measurement Characterization]:[number of goals], [Measurement Characterization]:[number of questions], [Measurement Characterization]:[number of measures]	1
	measurement plan	describes who measures what, when and how concerning the GQM measures defined in the GQM plan(s) and includes data collection instruments	1	has-parts [Measurement Plan].[gqm product experience]	-	yes	-	[GQM Plan]:[gqm measure]	1
I/F									
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Question
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	question	represents informational needs wrt. the measurement goal in natural language	1	Text	-	yes	-	-	0
	hypothesis	specifies the expected values wrt. quality dimensions or variation factors focused in the question	0..*	Text	-	no	-	-	0
	question category	categorization of questions by their concerns	1	Question-Category	-	yes	-	-	1
I/F	model	operationalizes this question	1	defines [GQM model].[question]	-	yes	-	-	1
	item	states the respective item (quality item or variation item) of the abstraction sheet	1	defined-by [Item]	-	yes	-	-	1
ctxt	gqm plan	references the respective GQM Plan	1..*	part of [GQM Plan].[gqm question]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: GQM Solution
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	changed object state	describes the solution by stating the modified, added or deleted object(s) and its state	1..*	refers-to [Software Object]	-	yes	-	-	1
	justification	for the solution, focusing on the interdependencies between the cause, its explanation and the applied solution	1	Text	-	yes	-	-	0

I/F

Concept: GQM Solution
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt	gqm problem solution experience	references the respective GQM Problem Solution Experience	1	part-of [GQM Problem Solution Experience].[solution]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Item
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif									
I/F									
ctxt	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Measurement Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	measurement integrated	measurement programs regularly established accompanying software development and maintenance	0..1	Boolean	-	no	-	-	1
	experiences with measurement	specifies if no experiences are available or either positive or negative experiences have been made with measurement in the past	0..1	Measurement-Knowledge	-	no	-	-	1
	core measures	specifies if a set of core measures is collected in each project if measurement programs are performed regularly	0..1	Boolean	-	no	-	-	1
	attitude	of management and project personnel concerning software quality improvement in general	0..1	Attitude	-	no	-	-	1
	effort	on the planning and execution of the measurement program in person-months	0..1	EffortPM	-	no	-	-	1
	duration	of the measurement program in calendar months	0..1	DurationM	-	no	-	-	1
	duration of data collection	period in calendar months	0..1	DurationM	-	no	-	-	1
	frequency of feedback sessions	during the execution phase per calendar month	0..1	FreqM	-	no	-	-	1
	training	describes training(s) of the participants regarding the GQM approach and its application which took place during the planning phase	0..1	Text	-	no	-	-	1
	number of goals	size of the measurement program in terms of number of GQM goals	0..1	Cardinal	-	no	card(union(filter([GQM Product Experience], [context].[measurement experience]).[gqm plan].[gqm goal]))	-	1

Concept: Measurement Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
	number of questions	size of the measurement program in terms of number of questions in the GQM plans	0..1	Cardinal	-	no	card(union(union(filter([GQM Product Experience], [context].[measurement experience]).[gqm plan]).[gqm question]))	-	1
	number of measures	size of the measurement program in terms of number of measures in the GQM plans	0..1	Cardinal	-		card(union(union(filter([GQM Product Experience], [context].[measurement experience]).[gqm plan]).[gqm measure]))		
I/F	constraints	on the measurement program, e.g., fixed amount of effort assigned to the measurement program	0..*	Text	-	no	-	-	0

Concept: Measurement Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt	context	references the respective context characterization	0..1	part-of [Context Characterization].[measurement context]	-	no	-	[Measurement Characterization]: [number of goals], [Measurement Characterization]: [number of questions], [Measurement Characterization]: [number of measures]	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Measurement Experience
Super concept: Experience

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif									
I/F									

Concept: Measurement Experience
Super concept: Experience

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt	context	describes the environment in which the measurement program takes place (including organizational, project and measurement specific characteristics)	1	has-parts [Context Characterization].[measurement experience]	-	yes	-	-	1

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Measurement Plan
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif									
I/F	data collection procedure	determines when, how, and by whom data is to be collected	1..*	has-parts [Data Collection Procedure].[measurement plan]	-	yes	-	-	1
	data collection instruments	defines the instrument used for the data collection	1..*	has-parts [Data Collection Instrument].[measurement plan]	-	yes	-	-	1
ctxt	gqm product experience	references the respective GQM Product Experience	1	part-of [GQM Product Experience].[measurement plan]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

Concept: Measurement Plan
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Measurement Tool
Super concept: Data Collection Instrument

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	name	states the name of the tool	1	Text	-	yes	-	-	0
	functionality	describes the functionality offered by the tool	0..1	Text	-	no	-	-	0
	call command	describes how to invoke the tool	0..1	Text	-	no	-	-	0
	documentation	describes where to find documentation and further information on the tool	0..1	Text	-	no	-	-	0
I/F	preconditions for use	describes environment in which tool may be used, e.g., platform or operating system	0..1	Text	-	no	-	-	0
ctxt	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Organization Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	organization name	states the name of the organization	0..1	Text	-	no	-	-	1
	size of software development	number of employees	0..1	Cardinal	-	no	-	-	1
	business sectors	set of business sectors the organization operates in	0..*	BusinessSector	-	no	-	-	1
	certifications	regarding the software process held by the organization	0..*	Certification	{}	no	-	-	1
	quality assurance group	availability of an independent quality assurance group at the organization	0..1	Boolean	"false"	no	-	-	1
	activities documented	describes if a standard software process exists in terms of activities to be performed	0..1	Boolean	"false"	no	-	-	1
	entry/exit criteria documented	describes if a standard software process exists in terms of entry and exist criteria for each activity	0..1	Boolean	"false"	no	-	-	1
	input/output documented	describes if a standard software process exists in terms of inputs and outputs for each activity	0..1	Boolean	"false"	no	-	-	1
	life-cycle models used	set of life-cycle models used for the development and maintenance by the organization	0..*	LifecycleModel	-	no	-	-	1
	tools	used during development and maintenance, e.g., CASE tools, compilers, debuggers, editors	0..*	Text	-	no	-	-	0
	programming languages	used for software development	0..*	ProgLang	-	no	-	-	1
	types of software	produced through the organization	0..*	SwType	-	no	-	-	1
	average number of installations	of the software products	0..1	Cardinal Interval	-	no	-	-	1
	memory constraints	typically formulated wrt. the system developed, e.g., regarding the required size of working memory	0..1	Constraint	-				

Concept: Organization Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
I/F									
ctxt	strategic goals	of the organization, reflecting the long-term organizational goals, e.g., "obtain a market share of 60%"	0..*	Text	-	no	-	-	0
	business goals	of the organization, focusing on the short-term organizational goals, e.g., "reduce production cost of product xyz by 10%"	0..*	Text	-	no	-	-	0
	improvement goals	of the software organization, e.g., "reduce cycle time and/or costs", "improve the quality of the software systems", "increase user satisfaction"	0..*	Text	-	no	-	-	0
	context	references the respective context characterization	0..*	part-of [Context Characterization].[organization context]	-	no	-	-	0
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Periodic Event
Super concept: Data Collection Event

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	period		1	CollectionPeriod	-	yes	-	-	1
I/F									
ctxt	Comments	any additional information or comment	0..1	Text	-	no	-	-	0

Concept: Periodic Event
Super concept: Data Collection Event

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Process Event
Super concept: Data Collection Event

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	timing	describes if the data is collected at the begin or end of the respective phase/activity	1	CollectionTiming	-	yes	-	-	1
I/F	process	describes the phase or activity when data is collected	1	SWProcessTaxonomy	-	yes	-	-	1
ctxt	Comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Project Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	project name	states name of project	0..1	Text	-	no	-	-	0
	project start	date of the start of the project	0..1	Date	-	no	-	-	1
	project end	date of the end of the project	0..1	Date	-	no	-	-	1
	duration	of the project in calendar months	0..1	DurationM	-	no	-	-	1
	team size	number of project team members allocated to the project	0..1	Cardinal Interval	-	no	-	-	1
	effort	of the project in person months	0..1	EffortPM	-	no	-	-	1
	project-specific goals	e.g., to complete the development in time	0..*	Text	-	no	-	-	0
	application of standard software process	states the degree to which the standard process model was applied in the specific project	0..1	ReuseType	-	no	-	-	1
	life-cycle model used	life-cycle model used in the project	0..1	LifecycleModel	-	no	-	-	1
	tools	used during development and maintenance, e.g., CASE tools, compilers, debuggers, editors	0..*	Text	-	no	-	-	0
	programming languages	used in the project	0..*	ProgLang	-	no	-	-	1
	estimated product size	in KLOC	0..1	Cardinal	-	no	-	-	1
	type of software	produced in the project	0..1	SwType	-	no	-	-	1
	number of installations	of the software products	0..1	Cardinal	-	no	-	-	1
	memory constraints	typically formulated wrt. the system developed, e.g., regarding the required size of working memory	0..1	Constraint	-	no	-	-	1
	performance constraints	typically formulated wrt. the system developed, e.g., regarding the response time to user requests	0..1	Constraint	-	no	-	-	1
	portability	of the system to different hardware and/or software environments	0..1	Importance	-				

Concept: Project Characterization
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	portability	of the system to different hardware and/or software environments	0..1	Importance	-	no	-	-	1
I/F									
ctxt	project goal	project-specific goal, e.g., "terminate project within budget and time"	0..*	Text	-	no	-	-	0
	context	references the respective Context Characterization	0..*	part-of [Context Characterization].[project context]	-	no	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Quality Item
Super concept: Item

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	quality factor	informal information on the quality focus of the GQM goal	1	Text	-	yes	-	-	0
	hypothesis	specifies the expected value(s) of the quality factor	0..*	Text	-	no	-	-	0
I/F	gqm question		1..*	defines [GQM Question]	-	yes	-	-	1

Concept: Quality Item
Super concept: Item

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt	variation factors	captures the factors of the context which are expected to influence the baseline hypothesis	0..*	defines [Variation Item].[quality factors]	-	no	-	-	1
	abstraction sheet	references the corresponding abstraction sheet	1	part-of [Abstraction sheet].[quality item]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Questionnaire
Super concept: Data Collection Instrument

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	collector	states role or position of people in the organization by whom the corresponding data is collected	1..*	Role	-	yes	-	-	1
	administrative paragraph	includes information for the administration of the collected data, e.g., project identifier	1	Text	-	yes	-	-	0
	question	references the questions on the questionnaire	1..*	has-parts [Questionnaire Question].[questionnaire]	-	yes	-	-	1
I/F	event	specifies when the data will be collected	1..*	Data Collection Event	-	yes	-	-	1
ctxt	comments	any additional information or comment	0..1	Text	-	no	-	-	0

Concept: Questionnaire
Super concept: Data Collection Instrument

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Questionnaire Question
Super concept: GQM Product

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	question	textual form of question	1	Text	-	yes	-	-	0
	question explanation	explains the question, e.g. terms used.	0..1	Text	-	no	-	-	0
	answer unit	defines the unit of the answer	0..1	Text	-	no	-	-	0
	answer range	declares the range of the answer	0..1	Text	-	no	-	-	0
	answer explanation	explanation on the range of the answer	0..1	Text	-	no	-	-	0
	measure	references the corresponding measure	1	defined-by [GQM Measure].[questionnaire question]	-	yes	-	-	1
I/F									
ctxt	questionnaire	references the corresponding questionnaire	1..*	part-of [Questionnaire].[question]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Software Object
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	object	states the software object	1	SwObjectTaxonomy	-	yes	-	-	1
	attribute	measurable data item from the software entity	0..1	SwAttribute	-	no	-	-	1
	state	value of attribute	0..1	has-parts[TType].[SoftwareObjectState]	-	no	-	-	1
I/F									
ctxt	Comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: TCardinal
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	Lower Bound	Lower Bound of the interval	1	Cardinal	0	yes	-	-	1
	Upper Bound	The upper bound.	1	Cardinal	536870912	yes	-	-	1
I/F									
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: [LowerBound]≤[UpperBound]

Concept: TGlossaryEntryOrdered
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	Symbol	The name of the symbol	1	String	-	yes	-	-	1
	Description	Description of the symbol	1	String	-	yes	-	-	1
	VRange	Reverse link to the attribute in unordered Symbol	1	decomposition-of[TOrderedSymbol].[VRange]	-	yes	-	-	1
I/F									
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: TGlossaryEntryTax
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	Symbol	The name of the symbol	1	String	-	yes	-	-	1
	Description	Description of the symbol	1	String	-	yes	-	-	1
	VRange	A symbol of the range of the taxonomy	0..1	range-of[TTaxonomy-Root].[VRange]	-	no	-	-	1
	TaxNode	Reverse link	1	decomposition-of[TTaxonomyNode].[Symbol]	-	no	-	-	1
I/F									
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: TGlossaryEntryUnordered
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	Symbol	The name of the symbol	1	String	-	yes	-	-	1
	Description	Description of the symbol	1	String	-	yes	-	-	1
	VRange	Reverse link to the attribute in unordered Symbol	1	decomposition-of[TUnorderedSymbol].[VRange]	-	yes	-	-	1
I/F									
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: TInteger
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	Lower Bound	Lower Bound of the interval	1	integer	-536870912	yes	-	-	1
	Upper Bound	Upper Bound of the interval	1	integer	536870912	yes	-	-	1
I/F									
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: [LowerBound]≤[UpperBound]

Concept: TIntegerInterval
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	LeftLowerBound	Lower Bound of the left interval	1	integer	-536870912	yes	-	-	1
	LeftUpperBound	Upper Bound of the left interval	1	integer	536870912	yes	-	-	1
	RightLowerBound	Lower Bound of the right interval	1	integer	-536870912	yes	-	-	1
	RightUpperBound	Upper Bound of the left interval	1	integer	536870912	yes	-	-	1

I/F

ctxt

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: [LeftLowerBound]≤[LeftUpperBound] AND [RightLowerBound]≤[RightUpperBound] AND [LeftLowerBound]≤[RightLowerBound] AND [LeftUpperBound]≤[RightUpperBound]

Concept: TOrderedSymbol
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	VRange	Defines the range of a Symbol	1..*	has-decomposition[GlossaryEntryOrdered].[VRange]	-	yes	-	-	1

I/F

ctxt

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: TReal
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	Lower Bound	Lower Bound of the interval	1	real	-1.0e30	yes	-	-	1
	Upper Bound	Upper Bound of the interval	1	real	1.0e30	yes	-	-	1

I/F

ctxt

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: [LowerBound]≤[UpperBound]

Concept: TRealnterval
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	LeftLower Bound	Lower Bound of the left interval	1	real		yes	-	-	1
	LeftUpper Bound	Upper Bound of the left interval	1	real		yes	-	-	1
	Right-Lower Bound	Lower Bound of the right interval	1	real		yes	-	-	1
	RightUpper Bound	Upper Bound of the left interval	1	real		yes	-	-	1

I/F

ctxt

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: [LeftLowerBound]≤[LeftUpperBound] AND [RightLowerBound]≤[RightUpperBound] AND [LeftLowerBound]≤[RightLowerBound] AND [LeftUpperBound]≤[RightUpperBound]

Concept: TTaxonomy
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif									

Concept: TTaxonomy
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
I/F	LowerLevel	The next deeper level in the hierarchy	0..*	has-decomposition[TTaxonomyNode].[UpperLevel]	-	no	-	-	1

ctxt

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: TTaxonomyNode
Super concept: TTaxonomy

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	Symbol	Symbol stored in this node	1	has-decomposition[TGlossaryEntryTax].[TaxNode]	-	yes	-	[TTaxonomyRoot]:[VRange]	1
I/F	UpperLevel	The next higher level	1	decomposition-of[TTaxonomy].[LowerLevel]	-	yes	-	-	0
	RootLevel	Link to the root of the taxonomy	1	to-root[TTaxonomyRoot].[All]	-	yes	-	-	0

ctxt

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond:pos(1, [RootLevel].[VRange].[Symbol], [Symbol].[Symbol])=0
assertion: TRUE

Concept: TTaxonomyRoot
Super concept: TTaxonomy

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	VRange	The Range of the whole taxonomy	1..*	has-range[GlossaryEntryTax].[VRange]		yes	[All].[Symbol]	-	1
I/F	All	The link from root to all the nodes of the taxonomy. Needed because the precondition.	0..*	from-root[TTaxonomyNode].[RootLevel]	-	no	-	-	1
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: TText
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif									
I/F									
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: TUnorderedSymbol
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	Number	The number of the possible choosen Symbols (Symbol Set).	1	Cardinal	1	yes	-	-	1
	VRange	Shows the range of the value of a Symbol	1..*	decomposition[GlossaryEntryUnordered].[VRange]	-	yes	-	-	1
I/F									

Concept: TUnorderedSymbol
Super concept: TType

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: card([ValueLink])≤[Number]

Concept: TType
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif									
I/F	GQMMeasureRange	Reverse range link	0..1	part-of[GQM-Measure].[range]	-	no	-	-	1
	VariationItem-Range	Reverse range link	0..1	part-of[Variation Item].[range]	-	no	-	-	1
	Software-ObjectState	Reverse value link	0..1	part-of[SoftwareObject].[state]	-	no	-	-	1
	VariationItemExpectedValue	Reverse value link	0..1	part-of[VariationItem].[ExpectedValue]	-	no	-	-	1
ctxt									

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

Concept: Variation Item
Super concept: Item

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight
artif	variation factor	factor pertaining to the object of interest and the domain with potential impact on a quality item	1	Text	-	yes	-	-	0
	range	range of the variation factor as defined in general in the specific environment	0..1	has-parts[Type].[VariationItemRange]	-	no	-	-	1
	impact	describes the expected impact of the variation factor on the baseline hypothesis of the affected quality factor	1..*	Text	-	yes	-	-	0
I/F	quality factors	specifies the quality item affected by this variation item[a]	1..*	defined-by[Quality Item].[variation factors]	-	yes	-	-	1
ctxt	expected value	of the variation factor in the specific environment where the actual measurement program takes place	0..1	has-parts[TType].[VariationItemExpectedValue]	-	no	-	-	1
	abstraction sheet	references corresponding abstraction sheet	1..1	part-of [Abstraction Sheet].[variation item]	-	yes	-	-	1
	comments	any additional information or comment	0..1	Text	-	no	-	-	0

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

precond: TRUE
assertion: TRUE

a.

A.3 Type Table

Table 47: Type Table

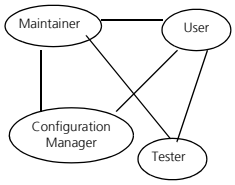
Name	Supertype	Value range	Unit of measure	Similarity
AcquisitionTechnique	UnorderedSymbol	"observation", "statistical analysis", "interview", "discussion"	n/a	Standard
Attitude	OrderedSymbol	"rejecting", "disinterested", "motivated"	n/a	see Supertype
BusinessSector	UnorderedSymbol	"aerospace", "electrical engineering", "energy", "information technology", "mechanical engineering", "motor vehicles", "telecommunications"	n/a	Standard
Certification	UnorderedSymbol	"ISO9000", "CMM2", "CMM3", "CMM4", "CMM5"	n/a	Standard
CollectionPeriod	OrderedSymbol	"daily", "weekly", "monthly", yearly"	n/a	see Supertype
CollectionTiming	UnorderedSymbol	"begin", "end"	n/a	Standard
Constraint	OrderedSymbol	"minimal", "normal", "severe"	n/a	see Supertype
DataStorage	Identifier	-	n/a	-
DurationM	Real	[0.00..*]	calendar months	Standard
EffortPM	Real	[0.00..*]	person months	Standard
FreqM	Real	[0.00..*]	per month	Standard
GQMProcessTaxonomy	TaxonomySymbol	Taxonomy 7:	n/a	Standard
Importance	OrderedSymbol	"unimportant", "desirable", "important", "crucial"	n/a	Standard
KLOC	Cardinal	[0..*]	KLOC	see Supertype

Name	Supertype	Value range	Unit of measure	Similarity
Lifecy-cleModel	UnorderedSym-bol	“iterative enhance-ment”, “proto-typing”, “spiral”, “waterfall”	n/a	Standard
Measure-ment-Knowledg e	OrderedSymbol	“not availa-ble”, “nega-tive”, “positive”	n/a	Standard
Model-Category	UnorderedSym-bol	“descriptive”, “evaluation”, “predictive”	n/a	Standard
Model-Type	UnorderedSym-bol	“quality model”, “resource mode”	n/a	Standard
Newness	OrderedSymbol	“version with-out new fea-tures”, “enhanced ver-sion”, “initial delivery”, “pro-totype”, “research pro-totype”	n/a	see Supertype
Platform	UnorderedSym-bol	“embedded processors”, “main frame computers”, “mini comput-ers”, “PC”, “workstations”	n/a	Standard
ProgLang	UnorderedSym-bol	“Ada”, “Assembler”, “C”, “C++”, “COBOL”, “Fortran”, “Smalltalk”	n./a	Standard
Purpose	UnorderedSym-bol	“characteriza-tion”, “moni-toring”, “evaluation”, “prediction”, “control”, “change”	n/a	Standard
Question-Category	TaxonomySym-bol	Taxonomy 4:	n/a	Standard

Name	Supertype	Value range	Unit of measure	Similarity
RepresentationForm	UnorderedSymbol	"diagram", "graph", "rule", "structured text", "table", "unstructured text"	n/a	Standard
Resource	UnorderedSymbol	"human", "tool"	n/a	Standard
ReuseType	OrderedSymbol	"not used", "used with many modifications", "used with few modifications", "used as is"	n/a	see Supertype
Role	UnorderedSymbol		n/a	Role-Graph
Scale	OrderedSymbol	"nominal", "ordinal", "interval", "ratio", "absolute"	-	Standard
Status	OrderedSymbol	"non-existent", "incomplete", "complete"	n/a	-
SwAttribute	TaxonomySymbol	Taxonomy 5:	n/a	Standard
SwObjectTaxonomy	TaxonomySymbol	Taxonomy 1:	n/a	Standard
SwObjectType	UnorderedSymbol	"product", "process", "resource"	n/a	Standard
SwProcessTaxonomy	TaxonomySymbol	Taxonomy 2:	n/a	Standard
SwProductTaxonomy	TaxonomySymbol	Taxonomy 3:	n/a	Standard
SwType	UnorderedSymbol	"batch processing", "decision support", "embedded/real-time systems", "interactive/reactive systems", "product/manufacturing systems", "transaction processing"	n/a	Standard

Name	Supertype	Value range	Unit of measure	Similarity
Unit	TaxonomySym-bol	Taxonomy 6:	n/a	Standard
Version	Real	[0.01..99.99]	n/a	-

Graph 1: **Role-Graph [BMS95]**



Taxonomy 1: SwObjectTaxonomy

root
- SW process
  - QA activities
  - maintenance process
  - development process
    - requirements analysis
    - design
    - implementation
    - test
      - component testing
      - unit testing
- SW product
  - development document
    - requirements document
    - design document
    - code
    - test document
      - test cases
      - test procedures
- resource
  - hardware
  - software
- communication

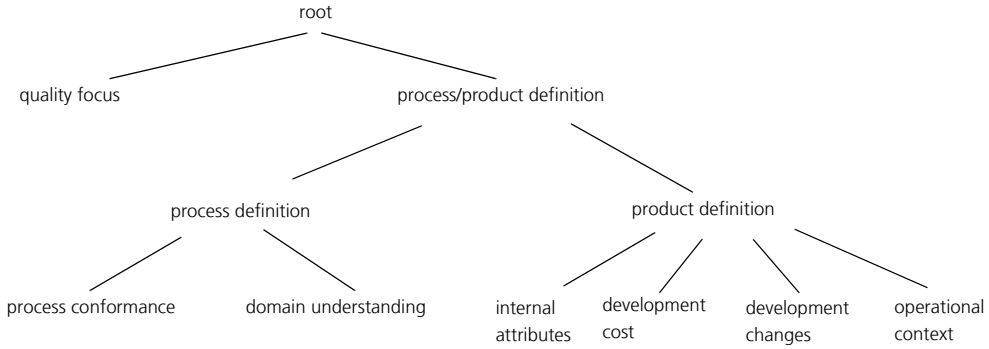
Taxonomy 2: SwProcessTaxonomy

root
- QA activities
- maintenance process
- development process
  - requirements analysis
  - design
  - implementation
  - test
    - component testing
    - unit testing

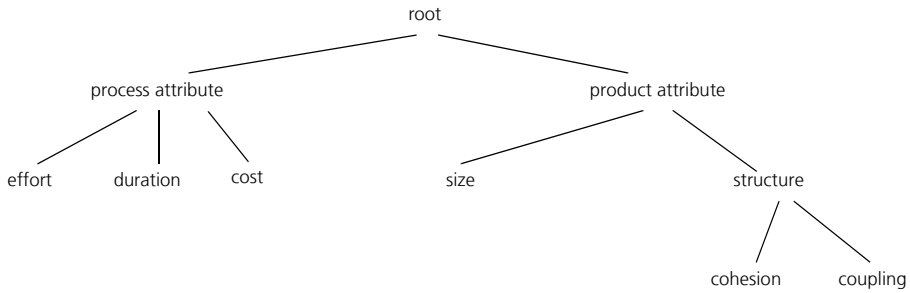
Taxonomy 3: SwProductTaxonomy

root
- development document
  - requirements document
  - design document
  - code
  - test document
    - test cases
    - test procedures

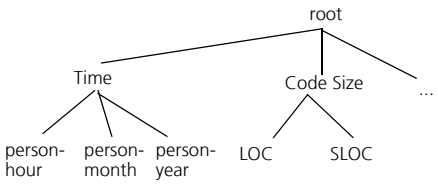
Taxonomy 4: QuestionCategoryTaxonomy



Taxonomy 5: wAttributeTaxonomy



Taxonomy 6: UnitTaxonomy



Taxonomy 7: GQMProcessTaxonomy

- GQM process
  - Prestudy
  - Identification of GQM goals
  - Development of GQM plan
    - GQM interviews
    - Development of questions
    - Development of models
    - Development of measures
    - Review of GQM plan
  - Development of measurement plan
    - Development of data collection procedures
    - Development of data collection instruments
    - Review of measurement plan
  - Data collection
  - Data analysis and interpretation
    - Analysis
    - Interpretation
  - Packaging

A.4 Symbol Glossary

Table 48: Symbol Glossary

Type	Symbol	Description
Acquisi-tionTech-nique	observation	gathering information by noting facts or occurrences
	statistical analysis	mathematics dealing with the analysis of masses of numerical data
	interview	a meeting at which information is obtained from a person
	discussion	consideration of a question in open usually informal debate
Attitude	rejecting	refusing to accept and support measurement
	disinterested	without any interest wrt. measurement
	motivated	interested in and agreeing on the application of measurement
Business-Sector	aerospace	manufacture or use of vehicles used in aerospace
	electrical engineering	engineering that deals with the practical applications of electricity
	energy	manufacture or use of power plants
	information technology	technologies of computers and telecommunications
	mechanical engineering	related to machinery
	motor vehicles	related to motor vehicles
	telecommunicat ions	related to means of distance communication
Certifica-tion	ISO9000	organization certified with ISO9000
	CMM2	organization certified on level 2 of CMM
	CMM3	organization certified on level 3 of CMM
	CMM4	organization certified on level 4 of CMM
	CMM5	organization certified on level 5 of CMM
Collection-Period	daily	data is collected once a day
	weekly	data is collected once a week
	monthly	data is collected once a month
	yearly	data is collceted once a year
Collection-Timing	begin	data is collected at the begin of an activity
	end	data is collected at the end of an activity
Constraint	minimal	not considered as critical
	normal	not considered as strongly critical but also not neglected completely
	severe	strongly critical

Type	Symbol	Description
GQMProcessTaxanomy	GQM process	defines the planning, execution and packaging of GQM-based measurement programs
	Prestudy	describes the activities to establish all preconditions necessary for a GQM-based measurement program
	Identification of GQM goals	describes the activity of identifying GQM goals
	Development of GQM plan	describes the activities wrt. the derivation of measures via questions and models.
	GQM interviews	describes the planning, performance and documentation of GQM interviews
	Development of questions	describes the development of questions based on the interview results
	Development of models	describes the development of models wrt. the questions
	Development of measures	describes the development of measures based on the models
	Review of GQM plan	describes the review process of GQM plan to check completeness and correctness
	Development of measurement plan	describes the development of data collection procedures and data collection instruments wrt. the measures defined in the GQM plan
	Development of data collection procedures	describes the definition of collection procedures determining when, how and by whom data has been collected wrt. the measures defined in the GQM plan
	Development of data collection instruments	describes the development of data collection instruments wrt. the data collection procedures
	Review of measurement plan	describes the review of measurement plan
	Data collection	describes the collection, validation and storage of measurement data
	Data analysis and interpretation	describes the analysis and interpretation of measurement data
	Analysis	describes the analysis of measurement data
	Interpretation	describes the planning, execution and documentation of feedback sessions
	Packaging	describes the packaging of measurement data and experiences in models, standards, etc.
Importance	unimportant	having no influence
	desirable	worth seeking
	important	having great influence
	crucial	of the utmost importance

Type	Symbol	Description
Lifecy-cleModel	iterative enhancement	software development technique in which requirement analysis, design, implementation and testing occur in an overlapping, iterative (rather than sequential) manner, resulting in incremental completion of the overall software product.
	prototyping	A development technique in which a preliminary version of part or all of the software is developed to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process.
	spiral	A model of the software development process in which the constituent activities, requirement analysis, preliminary and detailed design, coding, integration and testing, are performed iteratively until the software is completed.
	waterfall	A model of the development process in which the constituent activities, requirement phase, design phase, implementation phase, test phase, are performed in that order, possibly with overlap, but with little or no iteration.
Measure-ment-Knowledg e	not available	measurement has never been applied in organization
	negative	measurement has been introduced before with negative results
	positive	mesurement has been established before with positive results
Model-Category	descriptive	describe a measure based on integrating other measures, e.g. m=F(x1,..,xn)
	evaluation	capture situation in which a particular attribute needs to be evaluated based on one or more of its measures, e.g. d=f(x1,..,xn) with d=(d1,...d2) decisions
	predictive	predict a particular attribute based on one or more of its measures, e.g., ê=f(x1,..xn) or the occurrence of a certain event p(e)=f(x1,..xn)
Model-Type	quality model	model concerning a quality of a sw object, e.g. reliability, reusability
	resource mode	model concerning resources related to a sw object, e.g. effort
Newness	version without new features	improved version (correction of faults) but without adding any new features
	enhanced version	improved version with new features
	initial delivery	no delivered versions existed before
	prototype	A preliminary type, form or instance of a system that serves as a model for later stages or for the final, complete version of the system.
	research prototype	A preliminary type, form or instance of a system that serves for research objectives.
Platform	embedded processors	used as part of a system or machine.
	main frame computers	
	mini computers	
	PC	
	workstations	

Type	Symbol	Description
ProgLang	Ada	Imperative programming language unifying different concepts from several programming languages.
	Assembler	Machine-oriented programming language.
	C	Programming language with properties and elements of assembler similar and higher programming languages.
	C++	Object-oriented programming language based on C.
	COBOL	Imperative programming language for commercial data processing.
	Fortran	Imperative programming language for applications in natural sciences and engineering.
	Smalltalk	Object-oriented programming language.
Purpose	characterization	aims at forming a snapshot of the current state/performance of the software development products and processes
	monitoring	aims at following the trends/evolution of the state/performance of processes and products
	evaluation	aims at comparing and evaluating products and processes
	prediction	aims at identifying relationships between various process and product factors using these relationships to predict relevant external attributes of products and processes
	control	aims at identifying causal relationships that influence the state/performance of products and processes
	change	aims at identifying causal relationships in order to change the development process to obtain higher product quality and process productivity
Question-Category	domain understanding	quantitative characterization of the object to which the process is applied and an analysis of the process performer´s knowledge concerning this object
	process conformance	quantitative characterization of the process and an assessment of how well it is performed
	process definition	category that contains questions concerning factors that may have an impact on the values of the quality attributes wrt. the studied process
	process/product definition	category that contains questions concerning factors that may have an impact on the values of the quality attributes
	product definition	category that contains questions concerning factors that may have an impact on the values of the quality attributes wrt. the studied product
	internal attributes	quantitative characterization of the product in terms of physical attributes such as size, complexity, etc.
	development cost	quantitative characterization of the resources expended related to this product in terms of effort, computer time, etc.
	development changes	quantitative characterization of the errors, faults, failures, adaptations, corrections, and enhancements related to this product
	operational context	quantitative characterization of the customer community using this product and their operational profiles
	quality focus	descriptive models of the quality perspective of interest

Type	Symbol	Description
Represen-tationForm	diagram	drawing, sketch, plan or chart that makes something clearer or easier to under-stand
	graph	diagram that represents change in one variable factor in comparison with that of one or more other factors or pictorial representation of a set of points (as a line or curve) that satisfy a mathematical equation or belong to a given set.
	rule	prescribed guide for conduct or an action
	structured text	a textual description organized by a defined structure
	table	systematic arrangement of data in rows and columns
	unstructured text	a textual description
Resource	human	A human resource allocated to the development, testing, analysis, maintenance or measurement of a program or its documentation.
	tool	A computer program that is used in the development, testing, analysis, mainte-nance or measurement of a program or its documentation.
ReuseType	not used	no object has been reused
	used with many modifications	object has been reused, but with many modifications
	used with few modifications	object has been reused with few modifications
	used as is	object has been reused without any modifications
Role	Configuration Manager	integrates updates into the system, coordinates the production and release of versions of the system, and provides tracking of change requests.
	Maintainer	analyze changes, make recommendations, perform changes, perform unit and change validation testing after linking the modified units to the existing system, perform validation and regression testing after the system is recompiled by the Configuration Manager.
	Testers	present acceptance test plans, perform acceptance test and provide change request to the maintainers when necessary.
	Users	suggest, control and approve performed changes.
Scale	nominal	classification of objects, where the fact that objects are different is preserved (one-to-one mappings)
	ordinal	objects are ranked according to some criteria, but no information about the dis-tance between the values is given (monotonic increasing transformations)
	interval	differences between the values are meaningful (M´=aM+b (a>0))
	ratio	there is a meaningful "zero" value, and ratios between values are meaningful (M´= aM (a>0))
	absolute	no transformation is meaningful (M´= M)
Status	non-existent	Product does not exist.
	incomplete	Product does exist, but is still incomplete.
	complete	Product exists and is complete.

Type	Symbol	Description
SwAttrib-ute	cohesion	cohesion of a module is the extent to which its individual components are needed to perform the same task.
	coupling	coupling is the degree of interdependence between modules.
	duration	duration of a software process/phase/activity
	effort	human effort allocated to a software process/phase/activity
	process attribute	attribute of software process
	product attribute	attribute of software product
	size	size of software product, e.g. in terms of length, functionality, complexity or reuse.
	structure	structure of software products, concerning control flow, data flow and data structure.
SwObject-Taxonomy	test	An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.
	test cases	A set of inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
	test document	Documentation describing plans for, or results of, the testing of a system or component.
	unit test	Testing of individual software units or groups of related units.
SwObject-Type	process	processes are activities which are performed during a project. They create, read and modify products.
	product	the final software product is called product as well as all by-products, artifacts, and parts of a product´s documentation.
	resource	resources are entities that are necessary to perform the process.
SwType	batch processing	inputs to the system are collected an processed all at one time, rather than being processed as they arrive
	decision support	system aiming at the support of decisions.
	embedded/real-time systems	software as part of a larger system which performs some of the requirements of that system/computation is performed during the actual time that an external process occurs.system in which each user
	interactive/ reactive systems	
	product/ manufacturing systems	
	transaction processing	

Type	Symbol	Description
SwObject-Taxonomy	code	Computer instructions and data definitions expressed in a programming language or in a form output by an assembler, compiler or other translator.
	communication	Activities related to the exchange of information, e.g., meetings, business trips.
	component testing	Testing of individual components or groups of related components.
	design	The process of defining the architecture, components, interfaces, and other characteristics of a system or component.
	design document	A document that describes the design of a system or component.
	development document	A collection of material pertinent to the development of a given software unit or set of related units.
	development process	The process of developing a software system, typically includes requirement phase, design phase, implementation phase and test phase.
	hardware	Physical equipment used to process, store, or transmit computer programs or data.
	implementation	The process of translating a design into software components.
	maintenance process	The process of modifying a software system or component after delivery to correct faults, improve performance, or other attributes, or adapt to a changed environment.
	QA activities	A set of activities designed to evaluate the process by which products are developed or maintained.
	requirements analysis	The process of studying user needs to arrive at a definition of system, hardware, or software requirements.
	requirements document	A document that specifies the requirements for a system or a component.
	resource	Means used to develop a product or perform a service.
	software	Computer programs, procedures, and possibly documentation and data pertaining to the operation of a computer system.
	SW process	A sequence of steps performed for the development or maintenance of software.
	SW product	The complete set of computer programs, procedures, and possibly associated documentation and data designated for delivery to a user or any of these individual items.
	test	An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.
	test cases	A set of inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
	test document	Documentation describing plans for, or results of, the testing of a system or component.
	unit test	Testing of individual software units or groups of related units.

Type	Symbol	Description
SwProductTaxonomy	code	Computer instructions and data definitions expressed in a programming language or in a form output by an assembler, compiler or other translator.
	design document	A document that describes the design of a system or component.
	development document	A collection of material pertinent to the development of a given software unit or set of related units.
	hardware	Physical equipment used to process, store, or transmit computer programs or data.
	requirements document	A document that specifies the requirements for a system or a component.
	resource	Means used to develop a product or perform a service.
	software	Computer programs, procedures, and possibly documentation and data pertaining to the operation of a computer system.
	SW product	The complete set of computer programs, procedures, and possibly associated documentation and data designated for delivery to a user or any of these individual items.
	test cases	A set of inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
	test document	Documentation describing plans for, or results of, the testing of a system or component.
SwProcessTaxonomy	communication	Activities related to the exchange of information, e.g., meetings, business trips.
	component testing	Testing of individual components or groups of related components.
	design	The process of defining the architecture, components, interfaces, and other characteristics of a system or component.
	development process	The process of developing a software system, typically includes requirement phase, design phase, implementation phase and test phase.
	implementation	The process of translating a design into software components.
	maintenance process	The process of modifying a software system or component after delivery to correct faults, improve performance, or other attributes, or adapt to a changed environment.
	QA activities	A set of activities designed to evaluate the process by which products are developed or maintained.
	requirements analysis	The process of studying user needs to arrive at a definition of system, hardware, or software requirements.
	SW process	A sequence of steps performed for the development or maintenance of software.
	test	An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.
	unit test	Testing of individual software units or groups of related units.

Type	Symbol	Description
Unit	codesize	units refering to the size of code
	LOC	total number of lines of code
	person-hour	one working hour
	person-month	one working month (=24 days)
	SLOC	number of source lines of code excluding comments
	time	units refering to time

A.5 Predefined Kinds

Table 49: Predefined Kinds

Kind	Reverse name	Description	Structure	Properties
defines	defined-by	In the context of GQM measurement planning a specific organizational interdependency between GQM products has been identified, the *defines* relation (see Figure 12). This relation describes the fact that a GQM product (e.g., a question in the GQM plan) is defined based on another GQM product (e.g., a quality dimension in the abstraction sheet). The explicit modelling of this interdependency guarantees the traceability between the individual GQM products in a measurement program.	DAG	transitivity
depends	R(depends)	A special kind used for data collection procedures. The collection of data may depend on the collection of other data. For example, the finish date needs only be collected if the start date has also been collected in order to compute the duration of a process step. In this case, the collection of the finish date depends on the collection of the start date. If a measure is deactivated (i.e., temporarily no data for this measure is collected), all of its dependents should also be deactivated. (see Figure 14)	DAG	transitivity

Kind	Reverse name	Description	Structure	Properties
next	previous	A special kind of relationship used to point at the next solution applied, in case a solution applied to a problem which occurred failed. This helps to keep track of the solutions applied until the problem has been successfully solved. (see Figure 15)	DAG	transitivity
refers-to	referred-by	Relationship that indicates entities of the software process, e.g., design document or requirement analysis which are related to measurement entities (see Figure 13)	DAG	transitivity
to-root	from-root	Link from all nodes to the root of the taxonomy.	tree	
has-range	range-of	Declares th e range of values.	tree	

Figure 10: Kind »is-a« (generalization)

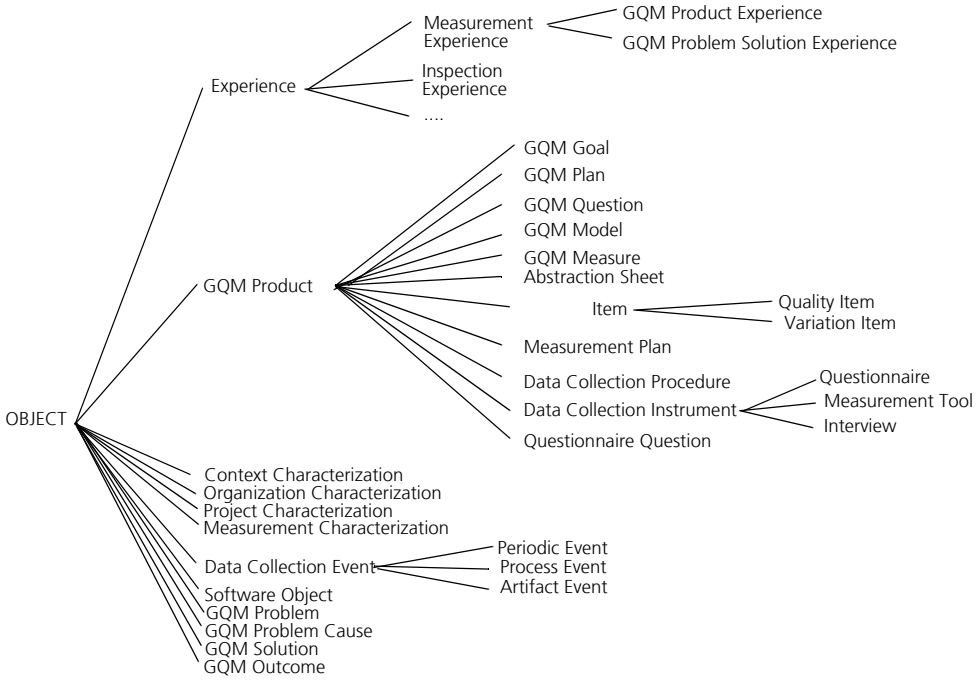


Figure 11: Kind »has-parts« (aggregation)

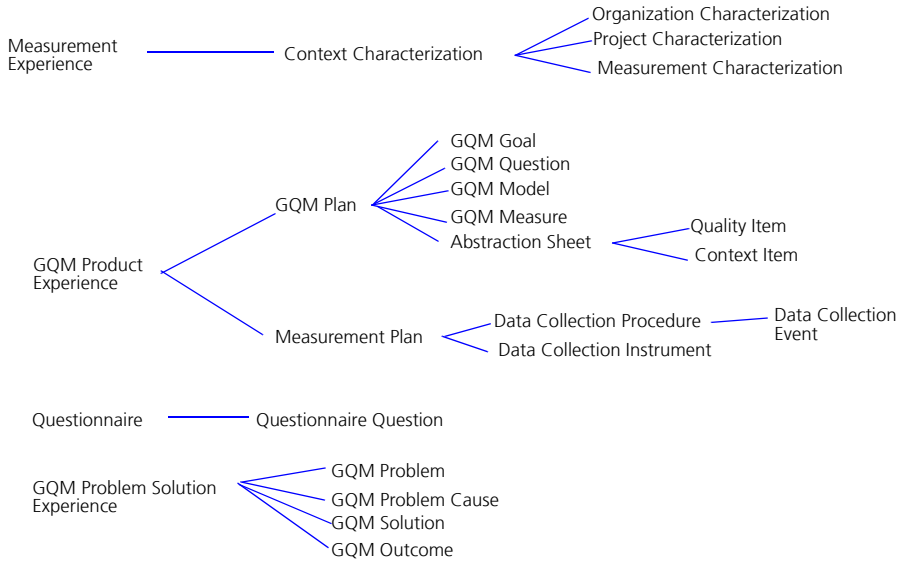


Figure 12: Kind »defines«

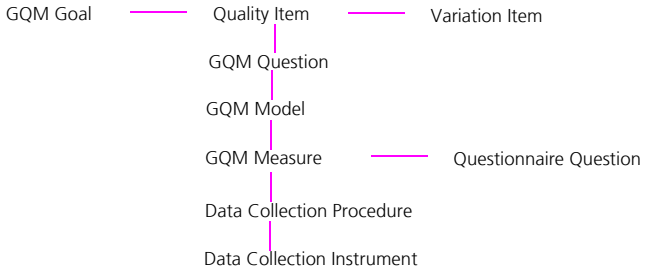


Figure 13: Kind »refers-to«

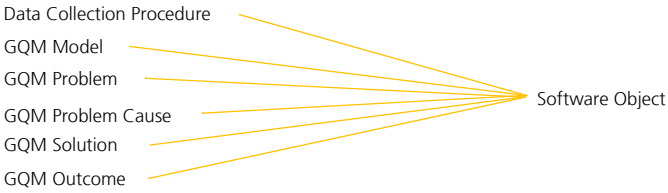


Figure 14: Kind »depends«

Figure 15: Kind »next«

Document Information

Title:	REFSENO – A Representation Formalism for Software Enginering Ontologies
Date:	October 20, 1998
Report:	IESE-015.98/E
Status:	Final
Distribution:	Public