INTERNATIONAL STANDARD

ISO/IEC 24744

First edition
2007-02-15

Software Engineering — Metamodel for Development Methodologies

Ingénierie du logiciel — Métamodèle pour les méthodologies de développement

Reference number
ISO/IEC 24744:2007(E)

Contents

Table of Figures

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 24744 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

Introduction

Development methodologies may be described in the context of an underpinning metamodel, but the precise mechanisms that permit them to be defined in terms of their metamodels are usually difficult to explain and do not cover all needs. For example, it is difficult to devise a practice that allows the definition of properties of the elements that compose the methodology and, at the same time, of the entities (such as work products) created when the methodology is applied. This International Standard introduces the Software Engineering Metamodel for Development Methodologies SEMDM, a comprehensive metamodel that makes use of a new approach to defining methodologies based on the concept of powertype. The SEMDM is aimed at the definition of methodologies in information-based domains, i.e. areas characterized by their intensive reliance on information management and processing, such as software, business or systems engineering. The SEMDM combines key advantages of other metamodelling approaches with none of their known drawbacks, allowing the seamless integration of process, modelling and people aspects of methodologies. Refer to Annex B where other metamodels are mapped to SEMDM and a brief synopsis of problems is provided.

Various methodologies are defined, used or implied by a growing number of standards and it is desirable that the concepts used by each methodology be harmonized. A vehicle for harmonization is the SEMDM. Conformance to this metamodel will ensure a consistent approach to defining each methodology with consistent concepts and terminology.

Software Engineering — Metamodel for Development Methodologies

1 Scope

This International Standard defines the Software Engineering Metamodel for Development Methodologies (SEMDM), which establishes a formal framework for the definition and extension of development methodologies for information-based domains (IBD), such as software, business or systems, including three major aspects: the process to follow, the work products to use and generate, and the people and tools involved.

This metamodel can serve as a formal basis for the definition and extension of any IBD development methodology and of any associated metamodel, and will be typically used by method engineers while undertaking such definition and extension tasks.

The metamodel does not rely upon nor dictate any particular approach to IBD development and is, in fact, sufficiently generic to accommodate any specific approach such as object-orientation, agent-orientation, component-based development, etc.

1.1 Purpose

This International Standard follows an approach that is minimalist in depth but very rich in width (encompassing domains that are seldom addressed by a single approach). It therefore includes only those higher-level concepts truly generic across a wide range of application areas and at a higher level of abstraction than other extant metamodels. The major aim of the SEMDM is to deliver a highly generic metamodel that does not unnecessarily constrain the resulting methodologies, while providing for the creation of rich and expressive instances.

In order to achieve this objective, the SEMDM incorporates ideas from several metamodel approaches plus some results of recent research (see [1-7] for details). This will facilitate:

- The communication between method engineers, and between method engineers and users of methodology (i.e. developers);
- The assembly of methodologies from pre-existing repositories of method fragments;
- The creation of methodology metamodels by extending the standard metamodel via the extension mechanisms provided to this effect;
- The comparison and integration of methodologies and associated metamodels; and
- The interoperability of modelling and methodology support tools.

The relation of SEMDM to some existing methodologies and metamodels is illustrated in Annex B.

1.2 Audience

Since many classes in the SEMDM represent the endeavour domain (as opposed to the methodology domain), it might look like developers enacting the methodology would be direct users of the metamodel. This is not true. Classes in the SEMDM that model endeavour-level elements serve for the method engineer to establish the structure and behaviour of the endeavour domain, and are not used directly during enactment. Only methodology elements, i.e. classes and objects created by the method engineer from the metamodel, are used by developers at the endeavour level, thus supporting both the creation of “packaged” methodologies as well as tailored, project-specific methodologies.

Here the term “method engineer” refers collectively to either a person constructing a methodology on site for a particular purpose or a person creating a "packaged" methodology as a "shrink-wrapped" process product.

2 Conformance

A metamodel is defined in accordance with this International Standard if it:

- describes the scope of the concepts in the metamodel in relation to the scope of the elements defined in Clause 7; and
- defines the mapping between the concepts that are addressed in the metamodel, and that are within the scope of this International Standard, and the corresponding elements of this International Standard (i.e. its elements cannot be substituted by others of identical intent but different construction).

A development methodology is defined in accordance with this International Standard if it is generated from a conformant metamodel as defined in the first paragraph of this clause (2 Conformance).

A development or engineering tool is developed in accordance with this International Standard if it implements a conformant metamodel as defined in the first paragraph of this clause (2 Conformance). If the purpose of the tool involves the creation of methodologies, then it is developed in accordance with this International Standard if it also implements the necessary features so as to make the mechanisms described in 8.1 available to the tool’s users. If the purpose of the tool involves the extension of the metamodel, then it is developed in accordance with this International Standard if it also implements the necessary features so as to make the mechanisms described in 9.1 available to the tool’s users.

NOTE 1 The metamodel thus defined does not necessarily have to include all the elements defined in Clause 7 – only those that are relevant to the purpose of the said metamodel are required.

NOTE 2 Conformance for methodologies or conformance for tools can be established without any necessity of explicitly including the detailed metamodel for any relevant work product kind or model unit kind. It is adequate to define the mappings of any such work products to the WorkProductKind and ModelUnitKind classes of the SEMDM.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply. Unless otherwise noted, the definitions are specific to this International Standard.

The following concepts are defined only for their usage throughout this International Standard.

NOTE – This International Standard uses a self-consistent set of core concepts that is as compatible as possible with other International Standards (such as ISO/IEC 12207, ISO/IEC 15504, etc.).

3.1
information-based domain
IBD
realm of activity for which information is the most valuable asset

NOTE This means that information creation, manipulation and dissemination are the most important activities within information-based domains. Typical information-based domains are software and systems engineering, business process reengineering and knowledge management.

3.2
methodology
specification of the process to follow together with the work products to be used and generated, plus the consideration of the people and tools involved, during an IBD development effort

NOTE A methodology specifies the process to be executed, usually as a set of related activities, tasks and/or techniques, together with the work products that must be manipulated (created, used or changed) at each moment and by whom, possibly including models, documents and other inputs and outputs. In turn, specifying the models that must be dealt with implies defining the basic building blocks that should be used to construct.

3.3
method
synonym of methodology

NOTE The term “methodology” is used throughout this International Standard, reserving the term “method” for conventional phrases such as “method engineer” or “method fragment”.

3.4
metamodel
specification of the concepts, relationships and rules that are used to define a methodology

3.5
endeavour
IBD development effort aimed at the delivery of some product or service through the application of a methodology

EXAMPLES Projects, programmes and infrastructural duties are examples of endeavours.

3.6
methodology element
simple component of a methodology

NOTE Usually, methodology elements include the specification of what tasks, activities, techniques, models, documents, languages and/or notations can or must be used when applying the methodology. Methodology elements are related to each other, comprising a network of abstract concepts. Typical methodology elements are Capture Requirements, Write Code for Methods (kinds of tasks), Requirements Engineering, High-Level Modelling (kinds of activities), Pseudo-code, Dependency Graphs (notations), Class, Attribute (kinds of model building blocks), Class Model, Class Diagram, Requirements Specification (kind of work products), etc.

3.7
endeavour element
simple component of an endeavour

NOTE During the execution of an endeavour, developers create a number of endeavour elements, such as tasks, models, classes, documents, etc. Some examples of endeavour elements are Customer, Invoice (classes), Name, Age (attributes), High-Level Class Model number 17 (a model), System Requirements Description (a document), Coding Cycle number 2, Coding Cycle number 3 (tasks), etc.

3.8
generation
act of defining and describing a methodology from a particular metamodel. Generating a methodology includes explaining the structural position and semantics of each methodology element using the selected metamodel. Thus, what methodology elements are possible, and how they relate to each other, are constrained by such a metamodel. Usually, method engineers perform generation, yielding a complete and usable methodology.

3.9
enactment
act of applying a methodology for some particular purpose, typically an endeavour

NOTE Enacting a methodology includes using the existing generated methodology to create endeavour elements and, eventually, obtain the targeted IBD system. Thus, what kinds of endeavour elements can be created, and how they relate to each other, is governed by the methodology being used. Usually, technical managers, together with other developers, perform enactment.

3.10
method engineer
person who designs, builds, extends and maintains methodologies

NOTE Method engineers create methodologies from metamodels via generation.

3.11
developer
person who applies a methodology for some specific job, usually an endeavour

NOTE Developers apply methodologies via enactment.

3.12
powertype
A powertype of another type, called the *partitioned type*, is a type the instance of which are subtypes of the partitioned type. This definition is interpreted in the context of the object-oriented paradigm. For example, the class TreeSpecies is a powertype of the class Tree, since each instance of TreeSpecies is also a subclass of Tree.

3.13
clabject
dual entity that is a class and an object at the same time

NOTE This definition is interpreted in the context of the object-oriented paradigm. Because of their dual nature, clabjects exhibit a class facet and an object facet, and can work as either at any time. Instances of powertypes are usually viewed as clabjects, since they are objects (because they are instances of a type, the powertype) and also classes (subtypes of the partitioned type).

4 Naming, diagramming and definition conventions, and abbreviated terms

4.1 Naming, diagramming and definition conventions

The SEMDM is defined using different kinds of instruments that complement each other. These instruments are:

- Definitions. Each concept in the SEMDM is defined using natural language. Also, a description is given, including the context in which the concept occurs and its most distinctive properties. Examples are also given for each concept.
- Class diagrams. Concepts of interest to the SEMDM are formalized as classes. Consequently, class diagrams are used to show these classes together with their attributes and relationships. UML 1.4.2 (i.e. ISO/IEC 19501) is used throughout with some noticeable exceptions. First, a special notation is used to depict powertype patterns, consisting of a dashed line between the powertype and the partitioned type with a black dot on the side of the powertype. Secondly, “white diamonds” are used to depict whole/part relationships without making any reference to their secondary characteristics (see [8] for more details).
- Text tables. Text tables are included to provide additional descriptions of attributes and relationships.
- Mappings to other approaches. Each concept in the SEMDM is related to equivalent or similar concepts in other metamodelling approaches, so that translation between approaches is easier.

These instruments are used simultaneously.

Two different types of class diagrams are provided. Clause 6 presents some diagrams that aim to give an overall picture of the structure of SEMDM. These diagrams are designed to give an idea of the main classes and relationships within the metamodel, and are not comprehensive, i.e. do not display every single detail of the metamodel. Clause 7, on the other hand, includes a class diagram for each class in the metamodel. The class under discussion is shown in the centre, and is surrounded by its closest neighbours. Each of these diagrams, together with the accompanying attribute and relationship tables, do contain all the details for the particular class being discussed.

The philosophy of the SEMDM is to offer broad coverage for all the issues often found in methodology definition avoiding, at the same time, unnecessary structural constraints on the resultant methodologies. Therefore, only a minimal set of attributes and associations is provided by the metamodel. Using powertype pattern instantiation (see sub-clause 8.1.2), and thanks to the usage of powertypes in the metamodel, additional attributes and associations can be easily added at the methodology domain.

4.2 Abbreviations

IBD	information-based domain
SEMDM	software engineering metamodel for development methodologies

5 Basic Concepts

Metamodels are useful for specifying the concepts, rules and relationships used to define methodologies. Although it is possible to describe a methodology without an explicit metamodel, formalizing the underpinning ideas of the methodology in question is valuable when checking its consistency or when planning extensions or modifications. A good metamodel must address all of the different aspects of methodologies, i.e. the process to follow, the work products to be generated and those responsible for making all this happen. In turn, specifying the work products that must be developed implies defining the basic modelling building blocks from which they are built.

Metamodels are often used by method engineers to construct or modify methodologies. In turn, methodologies are used by developers to construct products or deliver services in the context of endeavours. *Metamodel*, *methodology* and *endeavour* constitute, in this approach, three different areas of expertise that, at the same time, correspond to three different levels of abstraction and three different sets of fundamental concepts. As the work performed by developers at the endeavour level is constrained and directed by the methodology in use, the work performed by the method engineer at the methodology level is constrained and directed by the chosen metamodel. Traditionally, these relationships between “modelling layers”, here called “domains”, are seen as *instance-of* relationships, in which elements in one layer or domain are instances of some element in the layer or domain below (Figure 1).

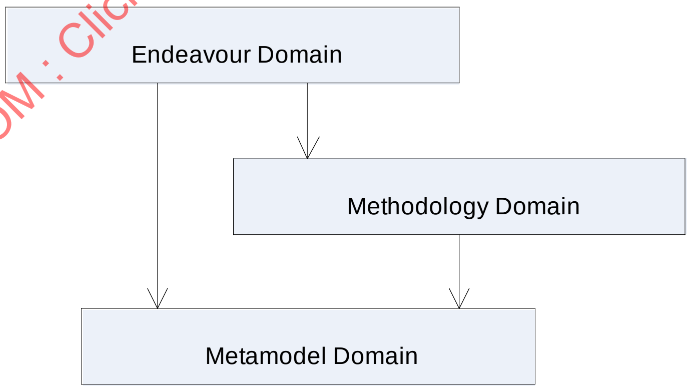


Figure 1 – The three areas of expertise, or domains, which act as a context for SEMDM

Regarding the methodology domain, it must be noted that more than one “methodology” may exist at this level, interlinked by refinement relationships. For example, it is common that organizations create organization-wide, generic methodologies from a metamodel, and then adjust and customize said methodologies for each particular endeavour. In cases like this, both kinds of methodologies (organization-wide and endeavour-specific) belong in the methodology domain and are connected via a refinement relationship (as opposed to instance-of). Cases with more than two steps of refinement are also possible.

5.1 Method Engineering

In accordance with most of the above-mentioned approaches to metamodelling, the SEMDM accepts the idea of method engineering (see [9, 10] for an introduction), defining the metamodel as a set of classes from which "methodology chunks" can be generated and then composed into a usable methodology [11]. However, the method engineering approach has been used primarily in the process realm (and hence the often-used name of "process engineering"), whereas the SEMDM extends it to the modelling domain as well (see 5.2).

5.2 Dual-Layer Modelling

Most metamodelling approaches define a metamodel as a model of a modelling language, process or methodology that developers may employ. Following this conventional approach, classes in the metamodel are used by the method engineer to create instances (i.e. objects) in the methodology domain and thus *generate* a methodology. However, these *objects* in the methodology domain are often used *as classes* by developers to create elements in the endeavour domain during methodology *enactment*. This apparent contradiction, not solved by any of the existing metamodelling approaches, is addressed by the SEMDM and solved by conceiving a metamodel as a model of *both the methodology and the endeavour* domains. While offering a strict model of the endeavour domain in the metamodel, the SEMDM maintains a high degree of flexibility, allowing the method engineer to configure the development process and address the modelling issues as necessary.

5.3 Powertypes and Clabjects

Two concepts, new to methodology modelling, must be introduced in order to support the features required by the SEMDM. First of all, modelling the methodology and endeavour domains at the same time gives rise to pairs of classes in the metamodel that represent the same concept at different levels of classification. For example, the Document class in the metamodel represents documents managed by developers, while the DocumentKind class in the metamodel represents different *kinds* of documents that can be managed by developers. Notice how Document represents a concept that belongs in the endeavour domain (documents that people manage) while DocumentKind represents a concept that belongs in the methodology domain (kinds of documents described by the methodology). For example, the concept of ClassDiagram is an instance of DocumentKind, but a given class diagram in the endeavour, with a particular author and creation time, is an instance of Document. In turn, these two classes are related by a classification relationship, since every document (in the endeavour domain) is an example (instance) of some particular kind of document (as defined in the methodology domain). This pattern of two classes in which one of them represents "kinds of" the other is called a *powertype pattern*, since the class with the "kind" suffix is a powertype (see [12] for an introduction to the powertype concept) of the other class, called the partitioned type. In this International Standard, the notation Document/*Kind is used to refer to the powertype pattern formed by the powertype DocumentKind and the partitioned type Document.

At the same time, endeavour-level elements must be instances of some methodology-level elements, and methodology-level elements must be instances of metamodel-level elements. This means that (at least some) elements in the methodology domain act *at the same time* as objects (since they are instances of metamodel classes) and classes (since endeavour-level elements are instances of them). This class/object hybrid concept has been described in [13] and named *clabject*. Clabjects have a class facet and an object facet. Within the SEMDM, clabjects are the means to construct a methodology from the powertype patterns found in the metamodel. In this way, a powertype pattern can be "instantiated" into a clabject by making the object facet of the clabject an instance of the powertype class in the powertype pattern, and the class facet of the clabject a subclass of the partitioned type in the powertype pattern. For example, a method engineer wanting to support requirement specification documents in the methodology that he or she is constructing would create the clabject RequirementsSpecificationDocument (in the methodology domain) as an instance of Document-Kind *and* a subclass of Document. By using clabjects at the methodology level, every single element susceptible of being instantiated during enactment is represented by a class, which is appropriate for instantiation, and by an object, which is appropriate for automated manipulation by tools.

Notice how a given attribute of the powertype class acts as *discriminator* of the powertype pattern, meaning that unique values of that attribute will be assigned to each of the instances of the powertype class, and the same value will be used to name the corresponding subclass of the partitioned type. For example, in the Document/*Kind powertype pattern, DocumentKind.Name is the discriminator. This means that each instance of DocumentKind will have a unique value for Name and its associated class (a subtype of Document) will be named with that value. Following the previous example, a given instance of DocumentKind would have Name = “ClassDiagram”, and its corresponding subclass of Document would be called ClassDiagram. The discriminator attribute thus acts as the bond between the two facets of the clabject.

5.4 Uniting Process and Product

Most of the existing metamodelling approaches focus either on the process or on the modelling (i.e. product) side of methodologies. Most of these approaches, however, offer connection points for “plugging in” the complementary, as yet undefined, component of a full-fledged methodology. The SEMDM goes a step beyond by offering a complete metamodel that covers the process and modelling aspects of methodologies evenly. Not doing so would be like trying to define the actions to be performed without defining the concepts on which these actions must act (process focus), or the concepts to use without knowing what to do with them (modelling focus). This approach has the benefit of allowing a rich definition, at the methodology level, of the interactions between a process and the products generated by it.

5.5 Process Assessment

Usually, the maturity or capability of an organization regarding the performance of a process is measured by assigning a *capability level* to its enactment. The SEMDM adopts the concept of capability level and attaches it to work unit kinds expressed using the MinCapabilityLevel attribute of class WorkUnitKind, so a method engineer can easily establish the minimum capability level at which each work unit kind may be performed. Although different assessment approaches and standards have slightly different ranges of capability levels (see [14] for an example), the following exemplar list is generic enough to be applicable to nearly every situation:

- **Incomplete** (level 0): the organization fails to successfully execute the process.
- **Performed** (level 1): the process is successfully executed but may not be rigorously planned and tracked.
- **Managed** (level 2): the process is planned and tracked while it is performed; work products conform to specified standards and requirements.
- **Established** (level 3): the process is performed according to a well-defined specification that may use tailored versions of standards.
- **Predictable** (level 4): measures of process performance are collected and analysed, leading to a quantitative understanding of process capability and an improved ability to predict performance.
- **Optimizing** (level 5): continuous process improvement against business goals is achieved through quantitative feedback.

6 Introduction to the SEMDM

6.1 Highly Abstract View

From the most abstract perspective, the SEMDM defines the classes MethodologyElement and Endeavour-Element that represent, respectively, elements in the methodology and the endeavour domains. Methodology-Element, in turn, is specialized into Resource and Template, corresponding to methodology elements that are used “as is” at the endeavour level (i.e. resources) and methodology elements that are used by instantiation at the endeavour level (i.e. templates) [3]. Since Template is the abstract type of all elements at the methodology level that will have instances at the endeavour level, and EndeavourElement is the abstract superclass of the same elements, these two classes form a powertype pattern in which Template is the powertype, Endeavour-Element is the partitioned type and Template.Name is the discriminant. Powertype patterns and their usage are discussed in sub-clause 5.3. See Figure 2 for a graphical representation.

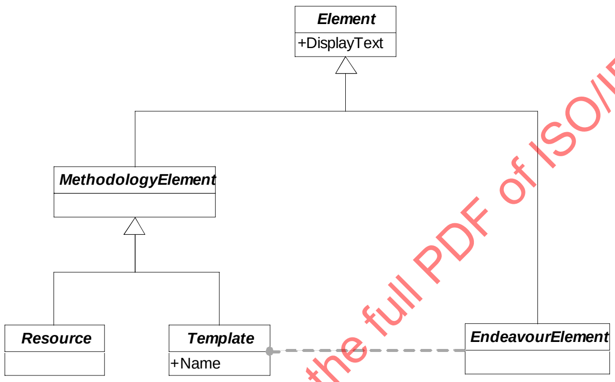


Figure 2 – Highly abstract view of the SEMDM

At the same time, a top class Element is defined to generalize MethodologyElement and EndeavourElement and allow homogeneous treatment of all elements across the methodology and endeavour domains when necessary. The DisplayText attribute of Element gives a short text describing each instance suitable to be shown to the instance’s final users.

6.2 Abstract View and Core Classes

There are three clusters of core classes: methodology templates, specializing from Template; methodology resources, specializing from Resource; and endeavour classes, specializing from EndeavourElement.

The powertype pattern formed by Template and EndeavourElement is refined into more specialized powertype patterns formed by subclasses of these two, namely: StageKind and Stage (representing a managed time frame within an endeavour), WorkUnitKind and WorkUnit (a job performed, or intended to be performed, within an endeavour), WorkProductKind and WorkProduct (an artefact of interest for the endeavour), ProducerKind and Producer (an agent that has the responsibility to execute work units) and ModelUnitKind and ModelUnit (an atomic component of a model). See Figure 3 for a graphical depiction.

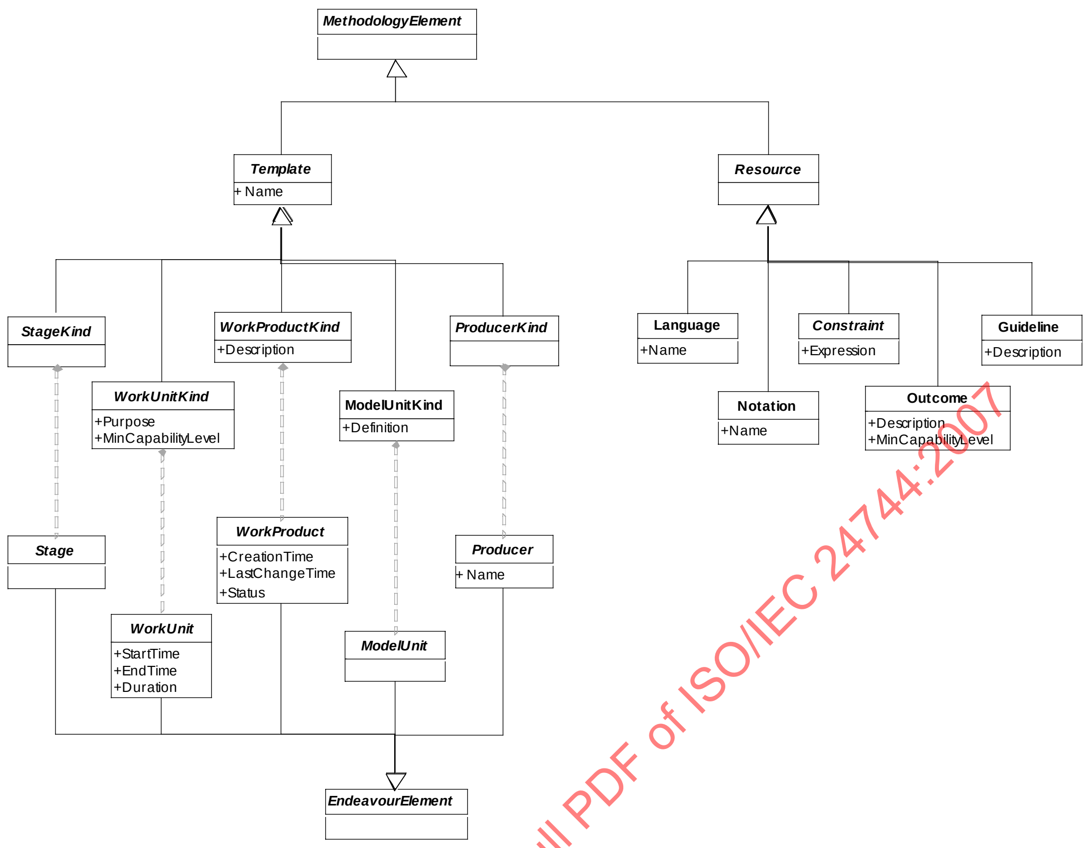


Figure 3 – Abstract view of the SEMDM, showing the core classes in the metamodel

At the same time, Resource is specialized into Language (a structure of model unit kinds that focus on a particular modelling perspective), Notation (a concrete syntax, usually graphical, which can be used to depict models created with certain languages), Guideline (an indication of how some methodology elements can be used), Constraint (a condition that holds or must hold at certain point in time) and Outcome (an observable result of the successful performance of a work unit).

6.3 Process Classes

The WorkUnit/*Kind powertype pattern is specialized into Process/*Kind (large-grained, operating within a given area of expertise), Task/*Kind (small-grained, focusing on *what* must be done in order to achieve a given purpose) and Technique/*Kind (small-grained, focusing on *how* the given purpose may be achieved).

WorkUnitKind is characterized by a purpose and a minimum capability level at which it makes sense to be performed, and is related to Outcome in a one-to-many fashion, so a set of outcomes can be defined for each specific kind of work unit. Also, WorkUnit/*Kind holds a whole/part relationship to Task/*Kind, so any work unit or work unit kind can be defined as a collection of tasks or task kinds, respectively. This allows for the recursive definition of units of work down to the necessary level of detail.

Since individual work units happen at the endeavour domain within a particular temporal frame (see below), the WorkUnit class incorporates the necessary attributes to describe this. The WorkUnitKind class, however, is only a specification of what must be done and does not contain any reference to any particular time frame; therefore, no time-related attributes are present. See Figure 4 for a graphical depiction.

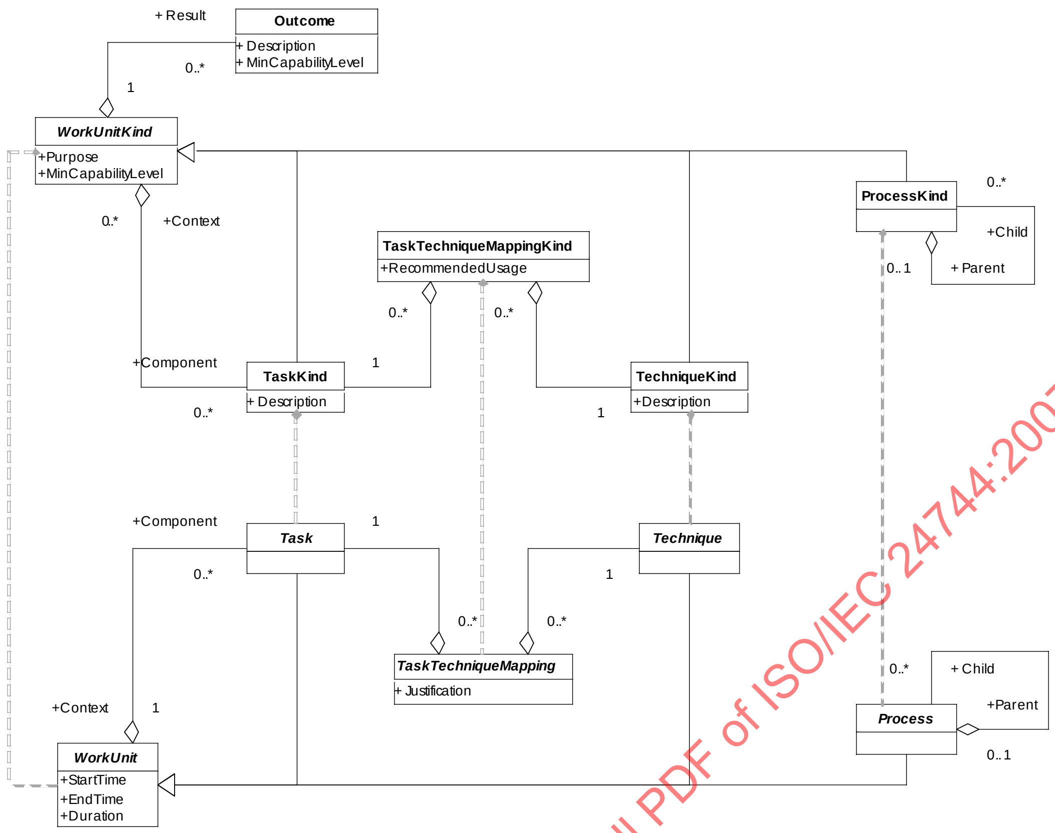


Figure 4 – Work units

On the temporal side, Stage/*Kind is specialized into StageWithDuration/*Kind (a managed interval of time within an endeavour) and InstantaneousStage/*Kind (a managed point in time within an endeavour). StageWithDuration/*Kind is, in turn, specialized into TimeCycle/*Kind (having as objective the delivery of a final product or service), Phase/*Kind (having as objective the transition between cognitive frameworks) and Build/*Kind (having as major objective the delivery of an incremented version of an already existing set of work products). TimeCycle/*Kind also holds a whole/part relationship to Stage/*Kind, allowing for the recursive composition of time cycles and other stages. Phase/*Kind, on the other hand, holds a whole/part relationship to Build/*Kind so any phase or phase kind can be linked to the corresponding builds or build kinds, respectively, that occur within it. At the same time, StageWithDuration/*Kind is associated with Process/*Kind so the temporal side of the process can be related to the appropriate elements on the job side. See Figure 5 for a graphical depiction.

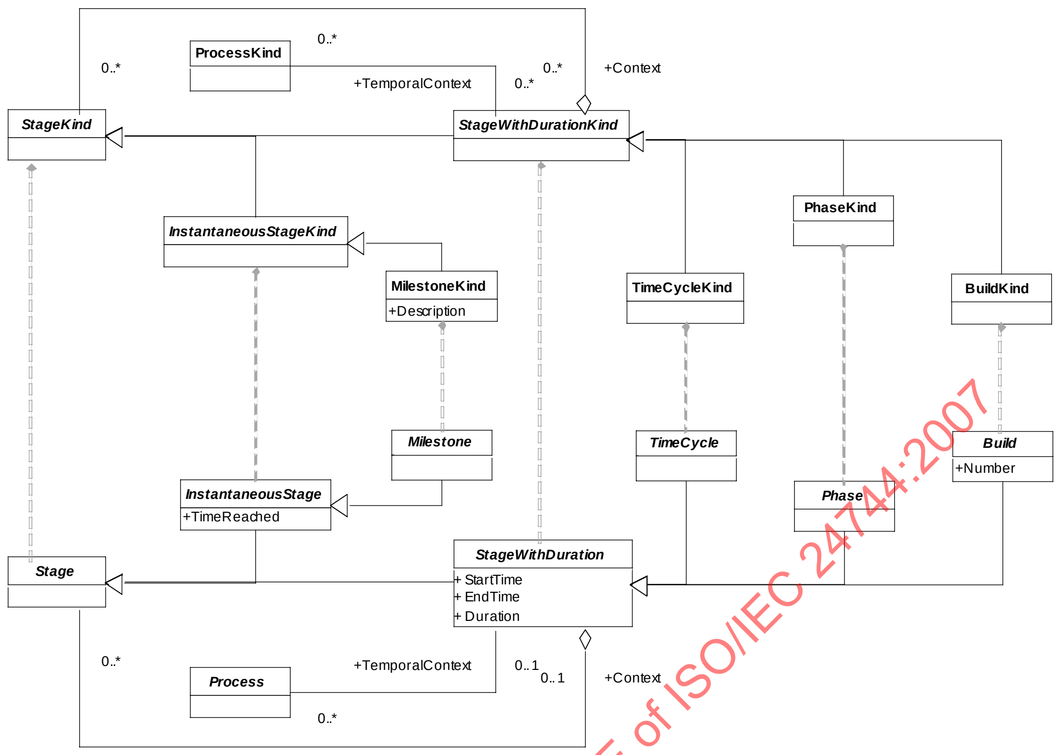


Figure 5 – Stages

NOTE – Temporal ordering and sequencing are achieved in SEMDM in two different ways. At a high level of abstraction, stage kinds allow the methodologist to specify the overall temporal structure of a methodology. Stage kinds, in this sense, are “empty containers” that can be “filled” with work unit kinds in order to specify when things are to be done. At a detailed level, however, time ordering and sequencing is not explicitly specified, but emerges from the collections of action kinds associated to each task kind. Action kinds of any given task kind determine what kinds of work products are necessary in order to accomplish the associated task. Thus, at any point in time during enactment, the set of “executable” tasks can be determined by looking at the pool of existing work products and the action kinds associated to each candidate task kind.

6.4 Producer Classes

Producer/*Kind is specialized into Role/*Kind (a collection of responsibilities that a producer can take), Tool/*Kind (an instrument that helps another producer to execute its responsibilities in an automated way). Producer has an additional subclass, Person, which allows taking into account individual persons at the endeavour level. Producer/*Kind is also related to WorkUnit/*Kind through WorkPerformance/*Kind, so links between units of work and the assigned and/or responsible producers are possible. See Figure 6 for a graphical depiction.

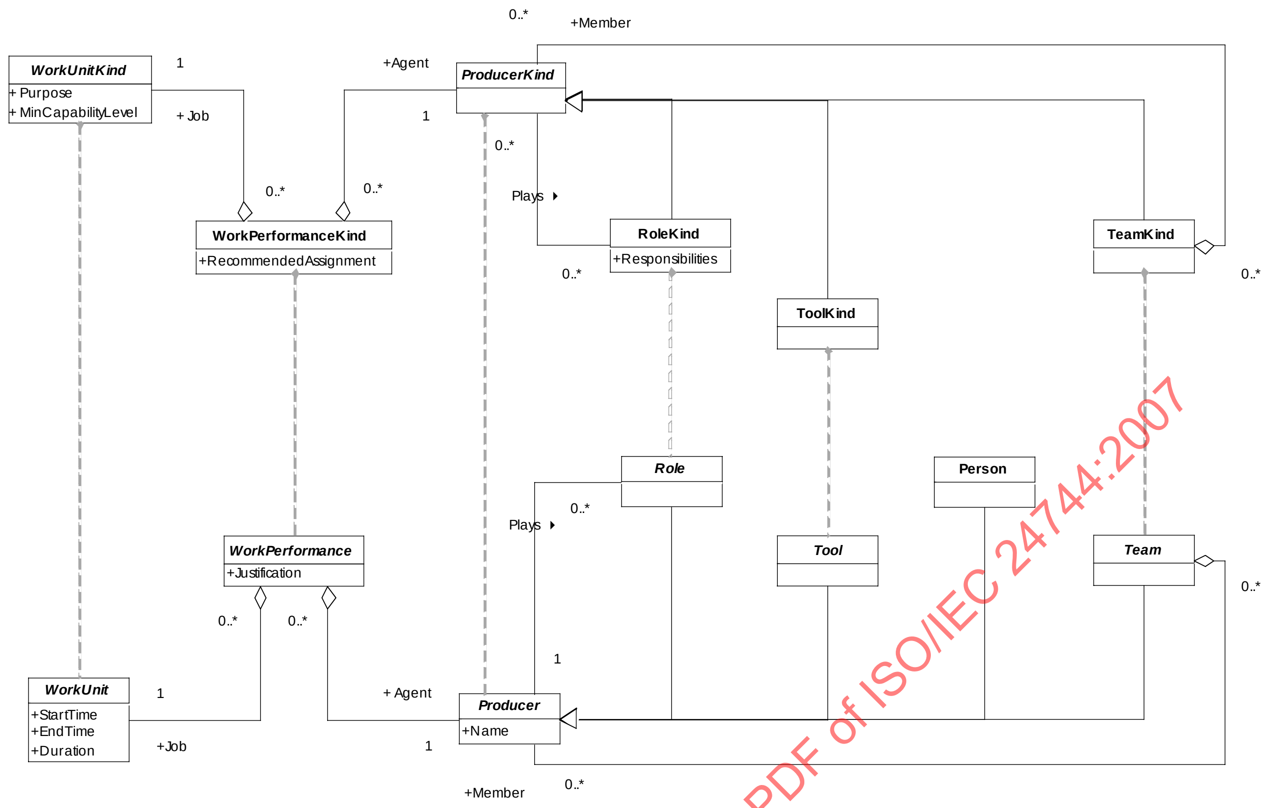


Figure 6 – Producers

6.5 Product Classes

WorkProduct/*Kind has five subtypes: SoftwareItem/*Kind, HardwareItem/*Kind (a piece of software or hardware, respectively, that is of interest to the endeavour), Model/*Kind (an abstract representation of some subject that acts as the subject's surrogate for some well defined purpose), Document/*Kind (a durable depiction of a fragment of reality) and CompositeWorkProduct/*Kind (an aggregate of other elements). Although documents would usually depict models, they can also depict other entities of interest or even other documents. In fact, Document/*Kind has an association to WorkProduct/*Kind to represent this fact. Also, Document/*Kind has a recursive whole/part relationship with itself so a given work product or work product kind can be defined as a collection of other work products or work product kinds, respectively. See Figure 7 for a graphical depiction.

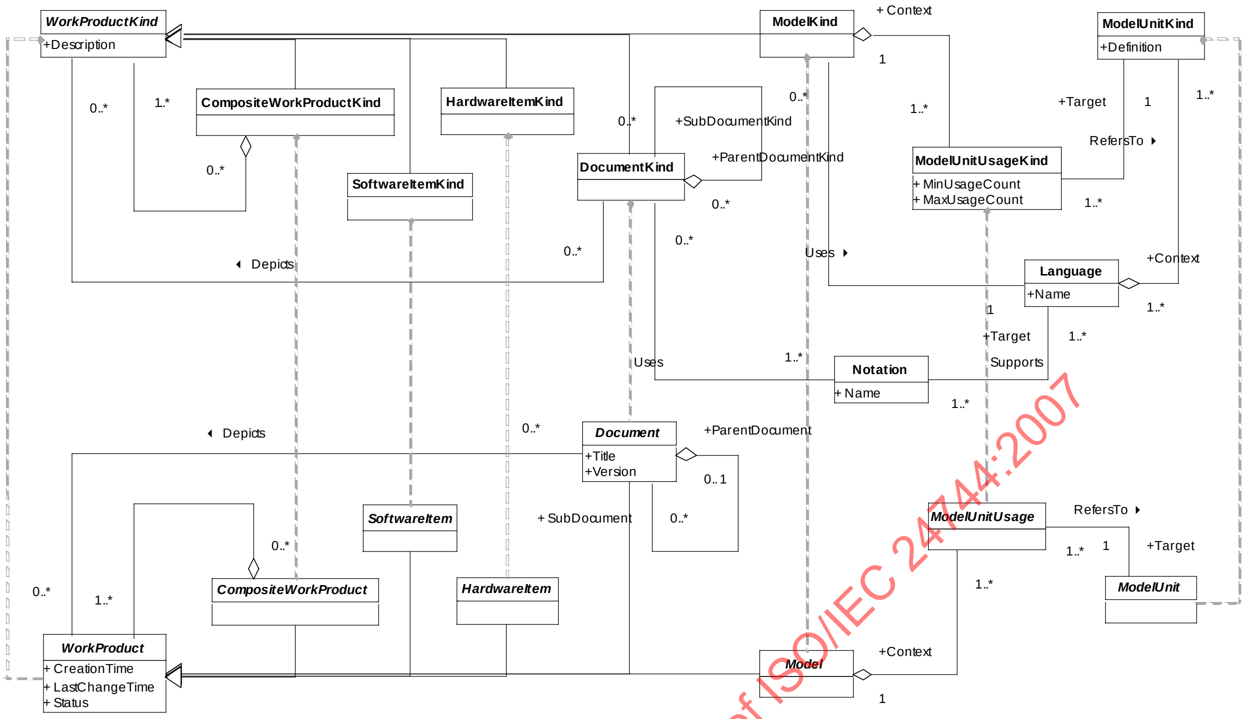


Figure 7 – Work product and modelling classes

Model/*Kind, in turn, holds a whole/part relationship to ModelUnitUsage/*Kind, which in turn is associated to a specific ModelUnit/*Kind. This chain of relationships makes it possible to describe what model units are used in which models, and how they are employed. In addition, every ModelUnitKind is always defined in the context of at least one Language. Different languages sharing the same model unit kinds allow for a single and interconnected network of model units and models across a system rather than having different separate, isolated models. In addition, ModelKind and Language are directly linked by an association to support cases in which a method engineer wishes to specify what language is used by a certain model kind without detailing the component model unit kinds. Also, Language is associated with Notation to represent the fact that different notations support (or can depict) different languages. Finally, Notation is also related to DocumentKind to represent that each document kind makes use of at least one notation.

Note that Language and ModelUnit/*Kind can generate any required modelling language.

6.6 Connection between Process and Product

The interaction between the process and the product sides of the metamodel is achieved by the powertype pattern Action/*Kind. An Action/*Kind is always performed in the context of a given Task/*Kind (process side), and acts upon a given WorkProduct/*Kind (product side). The ActionKind.Type attribute takes values showing whether actions of a specific kind create, modify or only read work products of a given kind. Note that some task kinds may not perform any action kinds whatsoever. See Figure 8 for a graphical depiction.

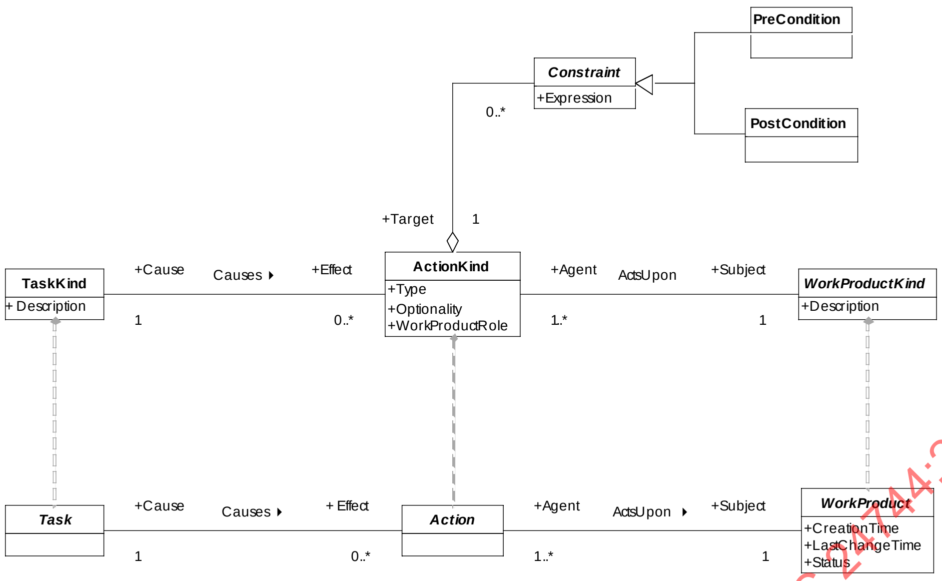


Figure 8 – Actions and constraints

ActionKind is also related to Constraint, which is specialized into PreCondition (a condition that must hold in order for the associated action to proceed) and PostCondition (a condition that is guaranteed to hold after the associated action has been successfully performed).

6.7 Support Classes

In addition to the classes necessary to construct methodologies, some support classes exist for the convenience of method engineers using the SEMDM. See Figure 9 for a graphical depiction.

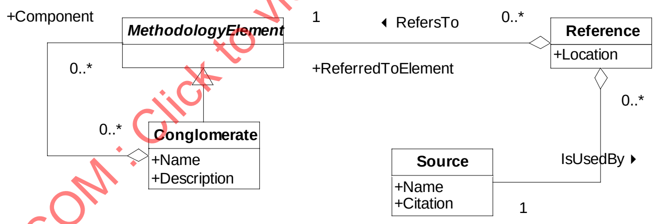


Figure 9 – Support classes

Some mechanism for specifying reusable methodology chunks is necessary. The Conglomerate class is defined to represent collections of related methodology elements (i.e. instances of the MethodologyElement class) that can be defined by a method engineer and then reused in different methodological contexts. Note that Conglomerate is also a subtype of MethodologyElement, so recursive composition of conglomerates is possible.

Also, some means of managing references to bibliographic sources and best practices is needed. The Source class represents literature items or other sources of information and experience that a method engineer may want to use when defining methodology elements. The Reference class acts as a link between Source and Element so any number of linkages between methodology elements and sources can be specified.

7 Metamodel Elements

7.1 Classes

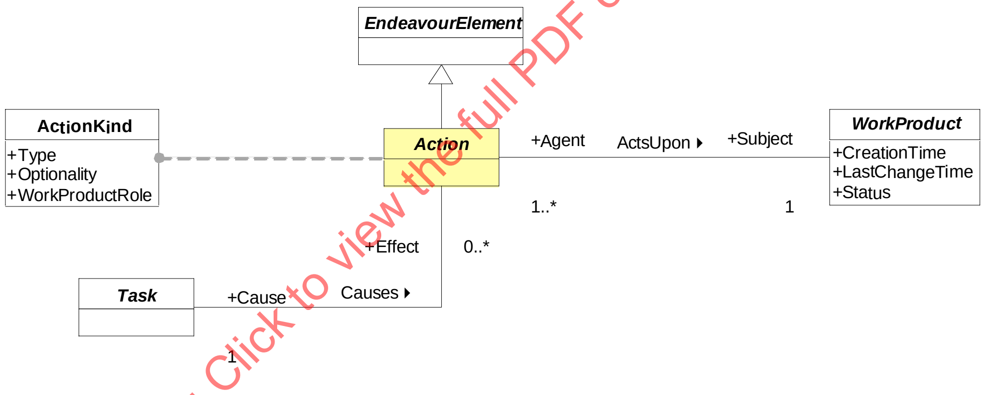
In this sub-clause, the classes in the SEMDM are described in alphabetical order. For each class, a definition is given in italics, and a description optionally follows. A diagram for each class is also included, showing the class being defined in the context of its immediate neighbours in the metamodel. Then, the attributes of the class are listed, including, for each one, its name, data type and semantics. Attribute data types are always one of the basic primitive types (Boolean, Integer, Timestamp or String) or an enumerated type defined in sub-clause 7.2. Finally, the relationships that the class is involved in are listed from the class' perspective, including, for each one, the name of the relationship if there is one, the role that the class being described plays in said relationship if there is one, the target class to which the class is associated, and its semantics.

7.1.1 Action

An action is *a usage event performed by a task upon a work product*. Actions represent the fact that specific tasks use specific work products.

Action is an abstract subclass of EndeavourElement.

This is a process- and product-related class.



7.1.1.1 Attributes

This class has no attributes of its own.

7.1.1.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	ActionKind	An action in the endeavour domain is always of some action kind defined in the methodology domain.
ActsUpon	Agent	WorkProduct	An action always acts upon a particular work product.
n/a	Effect	Task	An action is always the effect of a particular task.

7.1.1.3 Example

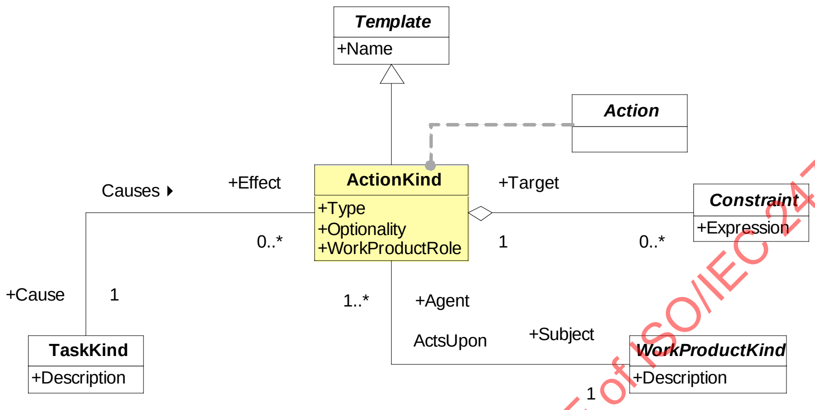
In a software development project, developer John executes a programming task (a task) that involves making modifications to the source code file “Invoice.cs” (a work product). The event of said task changing said work product is an action.

7.1.2 ActionKind

An action kind is *a specific kind of action, characterized by a given cause (a task kind), a given subject (a work product kind) and a particular type of usage.* Action kinds describe how tasks of specific kinds use work products of specific kinds, including the nature of such usage, i.e. creation, modification, etc.; its optionality (whether tasks of the associated kind always use work products of the associated kind via this action kind, or whether this is optional to some degree); and, optionally, the role that each work product kind plays when acted upon by the associated task kind. This last characterization is useful to differentiate work products when a single task kind is linked (via action kinds) to the same work product kind multiple times. In cases like this, it is expected that each action kind will be marked with a distinctive work product role.

ActionKind is a subclass of Template.

This is a process- and product-related class.



7.1.2.1 Attributes

Name	Type	Semantics
Type	ActionType	The nature of the usage that the associated task kind performs on the associated work product kind. See sub-clause 7.2.1 for possible values.
Optionality	DeonticValue	The degree of obligation for the associated task kind to act upon the associated work product kind. See 7.2.2 for possible values.
WorkProductRole	String	The role that work products of the associated kind will play during enactment within actions of this kind.

7.1.2.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Action	An action in the endeavour domain is always of some action kind defined in the methodology domain.
ActsUpon	Agent	WorkProductKind	An action kind is always the agent that acts upon a particular work product kind.
n/a	Effect	TaskKind	An action kind is always the effect of a particular task kind.
n/a	Target	Constraint	An action kind may be constrained by some constraints.

7.1.2.3 Example

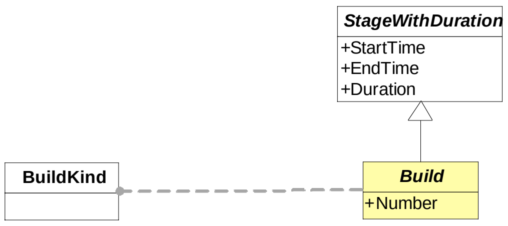
In a given methodology, a task kind “Determine business concepts” is defined, together with a work product kind “Business Concept Dictionary”. Both are related by the fact that tasks of the “Determine business concepts” kind will, when executed, create work products of the “Business Concept Dictionary” kind. Said relationship is modelled as an action kind with Type = Create and Optionality = Mandatory.

7.1.3 Build

A build is *a stage with duration for which the major objective is the delivery of an incremented version of an already existing set of work products.* Builds are often used to implement incremental, iterative time cycles.

Build is an abstract subclass of StageWithDuration.

This is a process-related class.



7.1.3.1 Attributes

Name	Type	Semantics
Number	String	The number of the build. Since builds are, by definition, incremental, some kind of numbering system is strongly recommended.

7.1.3.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	BuildKind	A build in the endeavour domain is always of some build kind defined in the methodology domain.

7.1.3.3 Example

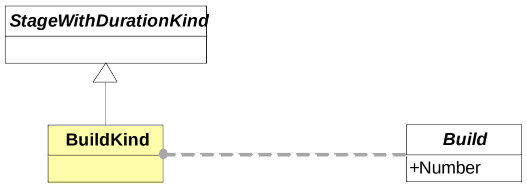
In a software development project, John's team spends two weeks focusing on analysing, modelling, implementing and testing a few new system features. After these two weeks, they deliver a partial implementation of the final system. Once this is done, they pick a new bunch of features and repeat the analysis, modelling, implementation and testing to construct an incremented version of the system. They keep doing this until the system contains all the required features. Each of the short time spans in which a new set of features is incrementally incorporated to the system, as performed by John's team, is a build.

7.1.4 BuildKind

A build kind is *a specific kind of build, characterized by the type of result that it aims to produce.*

BuildKind is a subclass of StageWithDurationKind.

This is a process-related class.



7.1.4.1 Attributes

This class has no attributes of its own.

7.1.4.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Build	A build in the endeavour domain is always of some build kind defined in the methodology domain.

7.1.4.3 Example

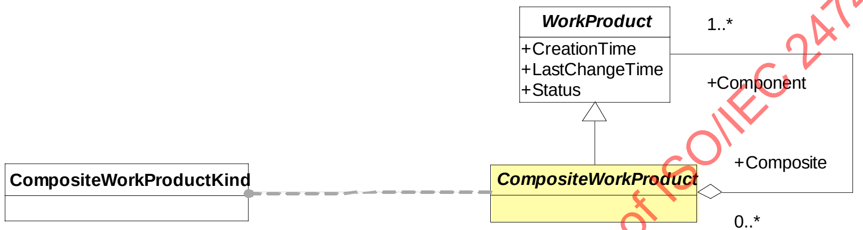
In a given methodology, a build kind "Construction Build" is defined to represent the fact that, when said methodology is enacted, a sequence of construction builds will be performed in order to construct the product incrementally.

7.1.5 CompositeWorkProduct

A composite work product is *a work product composed of other work products*.

CompositeWorkProduct is an abstract subclass of WorkProduct.

This is a product-related class.



7.1.5.1 Attributes

This class has no attributes of its own.

7.1.5.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	CompositeWork-ProductKind	A composite work product in the endeavour domain is always of some composite work product kind defined in the methodology domain.
IsMadeOf	Composite	WorkProduct	A composite work product is made of one or more work products.

7.1.5.3 Example

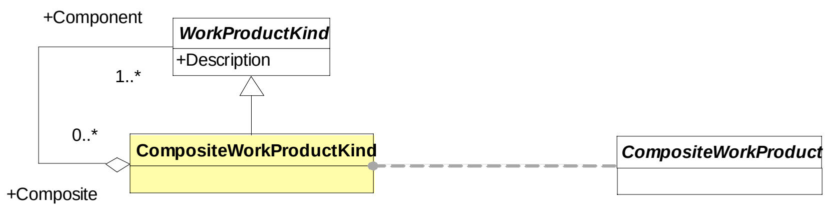
Upon finalization of a systems development project, a complex configuration of hardware, software and documentation is delivered to the customer. Each individual released work product can be modelled as either a hardware item, a software item or a document; the complete, final product delivered to the customer is a composite work product.

7.1.6 CompositeWorkProductKind

A composite work product kind is *a specific kind of composite work product, characterized by the kinds of work products that are part of it*.

CompositeWorkProductKind is a subclass of WorkProductKind.

This is a product-related class.



7.1.6.1 Attributes

This class has no attributes of its own.

7.1.6.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	CompositeWork-Product	A composite work product in the endeavour domain is always of some composite work product kind defined in the methodology domain.
IsMadeOf	Composite	WorkProductKind	A composite work product kind is made of one or more work product kinds.

7.1.6.3 Example

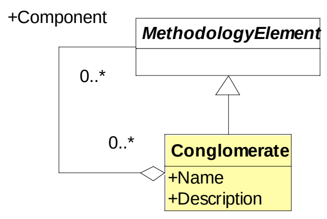
In a given methodology, the final product to be delivered to the customer upon completion of a project is modelled as a composite work product kind. Such a composite includes software and hardware item kinds plus the associated document kinds.

7.1.7 Conglomerate

A conglomerate is *a collection of related methodology elements that can be reused in different methodological contexts.* Conglomerates provide the basic reuse mechanism in the SEMDM.

Conglomerate is a subclass of MethodologyElement.

This is a support class.



7.1.7.1 Attributes

Name	Type	Semantics
Name	String	The name of the conglomerate.
Description	String	The description of the conglomerate, usually explaining the methodological contexts for which it has been designed.

7.1.7.2 Relationships

Name	Role	To class	Semantics
n/a	n/a	Methodology-Element	A conglomerate is composed of a collection of methodology elements.

7.1.7.3 Example

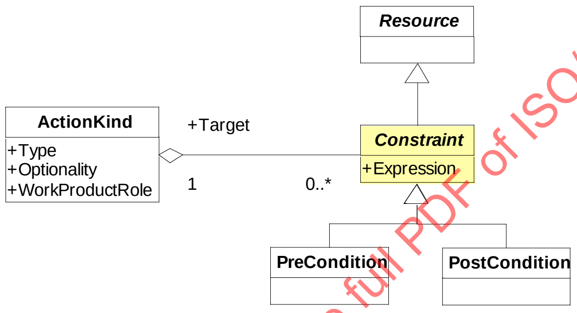
In a given methodology, the process kind “Quality Assurance”, the document kinds “Quality Standard” and “Quality Report”, and the team kind “Quality Assurance Team” (all of them methodology elements) are put into a conglomerate named “Quality-Related Fragments” so a methodologist can, easily and in a single step, incorporate it to a customized methodology or remove it from a methodology.

7.1.8 Constraint

A constraint is *a condition that holds or must hold at certain point in time*. Constraints are often used to declaratively characterize the entry and exit conditions of actions.

Constraint is an abstract subclass of Resource, specialized into PreCondition and PostCondition.

This is a process- and product-related class.



7.1.8.1 Attributes

Name	Type	Semantics
Expression	String	The expression that must evaluate as true for the constraint to hold. Note that this is an abstract attribute and therefore different subclasses attach different specific semantics to it.

7.1.8.2 Relationships

Name	Role	To class	Semantics
n/a	n/a	ActionKind	A constraint always characterizes a given action kind.

7.1.8.3 Example

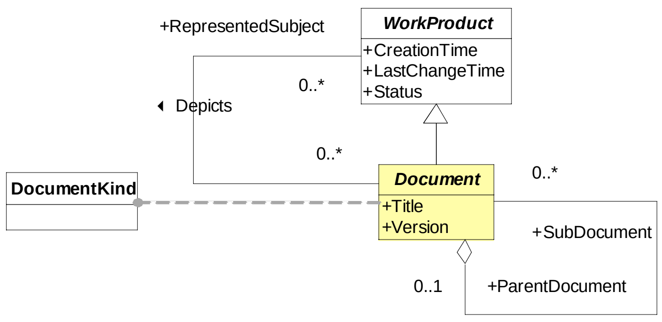
In a given methodology, a task kind “Deliver user documentation” is defined, together with a work product kind “User Documentation”. Both are related by the fact that tasks of the “Deliver user documentation” kind will, when executed, use work products of the “User Documentation” kind. Said relationship is modelled as an action kind with Type = ReadOnly. However, the method engineer wants to capture the need that user documentation is only delivered after the associated “User Interface Specification” work product has been approved. This is done by creating a constraint with Expression = (UserInterfaceSpecification.WorkProduct-Status is Approved) (a precondition) and attaching this constraint to the above mentioned action. Thus, the action will not be allowed to execute unless the required condition is met.

7.1.9 Document

A document is *a durable depiction of a fragment of reality*. Documents often represent models, but they can also represent other subjects.

Document is an abstract subclass of WorkProduct.

This is a product-related class.



7.1.9.1 Attributes

Name	Type	Semantics
Title	String	The title of the document.
Version	String	The version identifier of the document. Since documents are durable by definition, some version control is strongly recommended.

7.1.9.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	DocumentKind	A document in the endeavour domain is always of some document kind defined in the methodology domain.
n/a	ParentDocument	Document	A document may be the parent of sub-documents.
n/a	SubDocument	Document	A document may be a sub-document of a number of parent documents.
Depicts	n/a	WorkProduct	A document may depict a number of work products.

7.1.9.3 Example

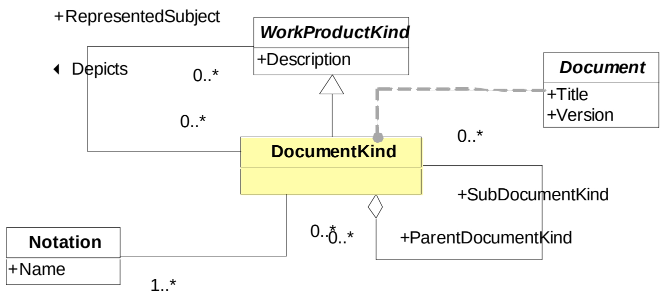
In order to organize a code inspection, Mary prints out the code to be inspected plus a copy of the inspection checklist. During the inspection, Mary takes notes of the defects found in the code and then she compiles these results into an inspection report. The code to be inspected, the inspection checklist and the inspection report, as used by Mary, are documents.

7.1.10 DocumentKind

A document kind is *a specific kind of document, characterized by its structure, type of content and purpose.*

DocumentKind is a subclass of WorkProductKind.

This is a product-related class.



7.1.10.1 Attributes

This class has no attributes of its own.

7.1.10.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Document	A document in the endeavour domain is always of some document kind defined in the methodology domain.
n/a	Parent-DocumentKind	DocumentKind	A document kind may be the parent of sub-document kinds.
n/a	SubDocument-Kind	DocumentKind	A document kind may be a sub-document kind of a number of parent document kinds.
Depicts	n/a	WorkProductKind	A document kind may depict a number of work product kinds.
n/a	n/a	Notation	A document kind uses some given notations.

7.1.10.3 Example

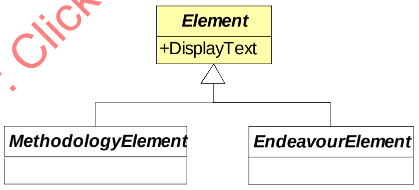
In a given methodology, the document kind “System Requirements Specification” is defined to represent the fact that, when said methodology is enacted, documents of such a kind will be created or used.

7.1.11 Element

An element is *an entity of interest to the metamodel.* Since the SEMDM addresses both the methodology and the endeavour domains (see sub-clause 5.2), any entity in either of these realms is subject to being modelled by the SEMDM and therefore to becoming represented by an element.

Element is an abstract class, specialized into MethodologyElement and EndeavourElement.

This is a high-level class.



7.1.11.1 Attributes

Name	Type	Semantics
DisplayText	String	Name or description suitable to be displayed to final users of the element. The value of this attribute can be, in many of the subclasses of Element, computed from other attributes.

7.1.11.2 Relationships

This class has no relationships of its own.

7.1.11.3 Example

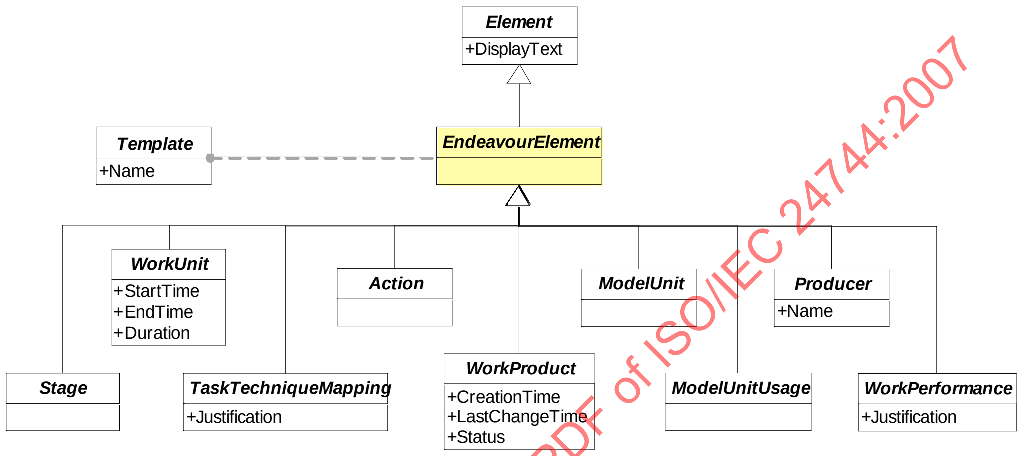
This class is too abstract to give a concrete example. Please see the examples for any of the subtypes of Element.

7.1.12 EndeavourElement

An endeavour element is *an element that belongs in the endeavour domain*. Any element created by a developer while using a methodology is represented by EndeavourElement.

EndeavourElement is an abstract subclass of Element, specialized into Stage, WorkUnit, TaskTechnique-Mapping, Action, WorkProduct, ModelUnit, ModelUnitUsage, Producer and WorkPerformance.

This is a high-level class.



7.1.12.1 Attributes

This class has no attributes of its own.

7.1.12.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	Template	An endeavour element is always of some template defined in the methodology domain.

7.1.12.3 Example

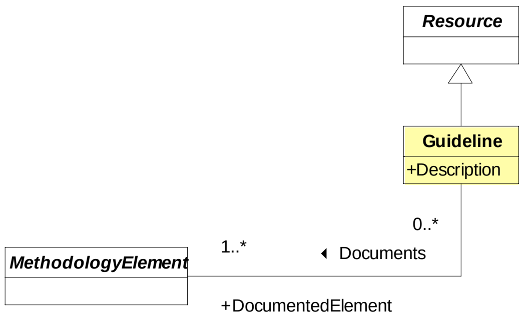
This class is too abstract to give a concrete example. Please see the examples for any of the subtypes of EndeavourElement.

7.1.13 Guideline

A guideline is *an indication of how a set of methodology elements can be used during enactment.*

Guideline is a subclass of Resource.

This is a high-level class.



7.1.13.1 Attributes

Name	Type	Semantics
Description	String	The description of the usage of the associated methodology elements.

7.1.13.2 Relationships

Name	Role	To class	Semantics
Documents	n/a	Methodology-Element	A guideline always documents some methodology elements.

7.1.13.3 Example

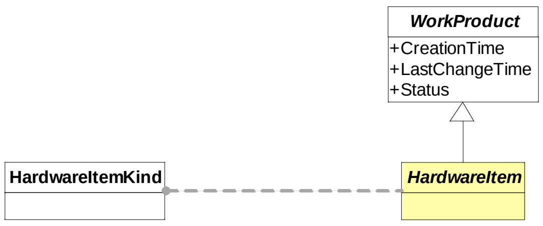
In a particular agent-oriented software development methodology, the model unit kind “Role” is defined to represent roles that agents may play at run-time. Since the concept of “role” in this context is different to the concept of “role” in object-oriented methodologies, the method engineer decides to create a guideline explaining how the model unit kind “Role” in this particular methodology is intended to be used, and attaches said guideline to said model unit kind.

7.1.14 HardwareItem

A hardware item is *a piece of hardware of interest to the endeavour*.

HardwareItem is an abstract subclass of WorkProduct.

This is a product-related class.



7.1.14.1 Attributes

This class has no attributes of its own.

7.1.14.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	HardwareItem-Kind	A hardware item in the endeavour domain is always of some hardware item kind defined in the methodology domain.

7.1.14.3 Example

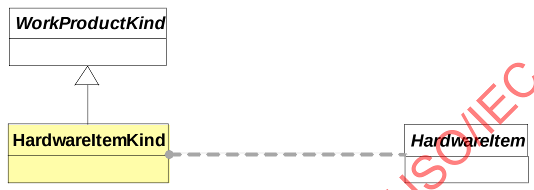
During an IT infrastructure deployment project, a number of sub-networks are organized and then interconnected via routers. Each sub-network and each router is a hardware item.

7.1.15 HardwareItemKind

A hardware item kind is *a specific kind of hardware item, characterized by its mechanical and electronic characteristics, requirements and features.*

HardwareItemKind is a subclass of WorkProductKind.

This is a product-related class.



7.1.15.1 Attributes

This class has no attributes of its own.

7.1.15.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	HardwareItem	A hardware item in the endeavour domain is always of some hardware item kind defined in the methodology domain.

7.1.15.3 Example

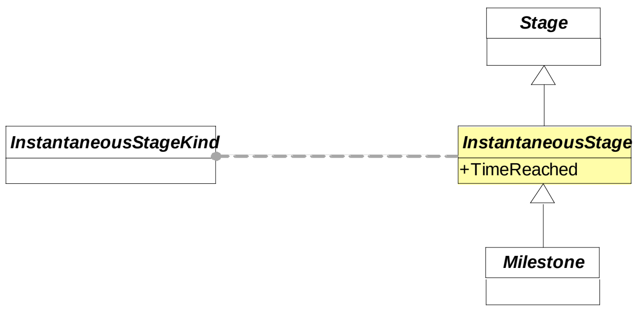
In a particular systems development methodology, the hardware item kind “Network” is defined to represent the fact that, when said methodology is enacted, hardware items of such kind will be created or used.

7.1.16 InstantaneousStage

An instantaneous stage is *a managed point in time within an endeavour.* Instantaneous stages usually correspond to significant events in the endeavour.

InstantaneousStage is an abstract subclass of Stage, specialized into Milestone.

This is a process-related class.



7.1.16.1 Attributes

Name	Type	Semantics
TimeReached	Timestamp	The point in time at which the instantaneous stage is reached.

7.1.16.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	Instantaneous-StageKind	An instantaneous stage in the endeavour domain is always of some instantaneous stage kind defined in the methodology domain.

7.1.16.3 Example

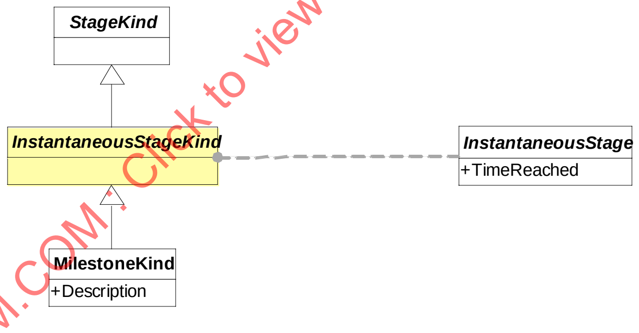
During a certain project for which John is project manager, the system definition phase is approaching completion. Once the system is fully defined, the system construction phase will take place. In order to manage the transition between phases, John tracks the point in time in which system definition is complete and stable, which marks said transition. This point in time is an instantaneous stage (to be precise, a milestone).

7.1.17 InstantaneousStageKind

An instantaneous stage kind is *a specific kind of instantaneous stage, characterized by the kind of event that it represents.*

InstantaneousStageKind is an abstract subclass of StageKind, specialized into MilestoneKind.

This is a process-related class.



7.1.17.1 Attributes

This class has no attributes of its own.

7.1.17.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Instantaneous-Stage	An instantaneous stage in the endeavour domain is always of some instantaneous stage kind defined in the methodology domain.

7.1.17.3 Example

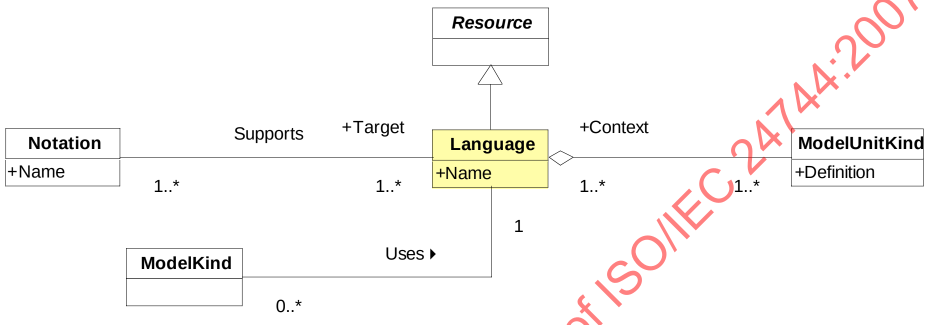
In a given methodology, two large phase kinds are defined in order to describe the definition of the system as opposed to the construction of the system. The transition between phases of these kinds needs to be marked by a managed point in time, and therefore the instantaneous stage kind “System definition is complete and stable” (a milestone kind) is defined to represent this fact.

7.1.18 Language

A language is *a structure of model unit kinds that focus on a particular modelling perspective.* This definition fits well with common definitions of “language”, such as “a systematic means of communicating by the use of sounds or conventional symbols“; in SEMDM’s definition, the symbols are the model unit kinds. Languages often focus on specific abstraction levels (i.e. informal view, high-level formal view, detailed formal view, etc.) and specific aspects of the modelled subject (i.e. structural, behavioural, visual, etc.).

Language is a subclass of Resource.

This is a product-related class.



7.1.18.1 Attributes

Name	Type	Semantics
Name	String	The name of the language.

7.1.18.2 Relationships

Name	Role	To class	Semantics
n/a	Context	ModelUnitKind	A language is always composed of some model unit kinds.
n/a	Target	Notation	A language is always targeted by at least one notation, which must be capable of depicting model units of the associated model unit kinds.
n/a	n/a	ModelKind	A language may be used by some model kinds.

7.1.18.3 Example

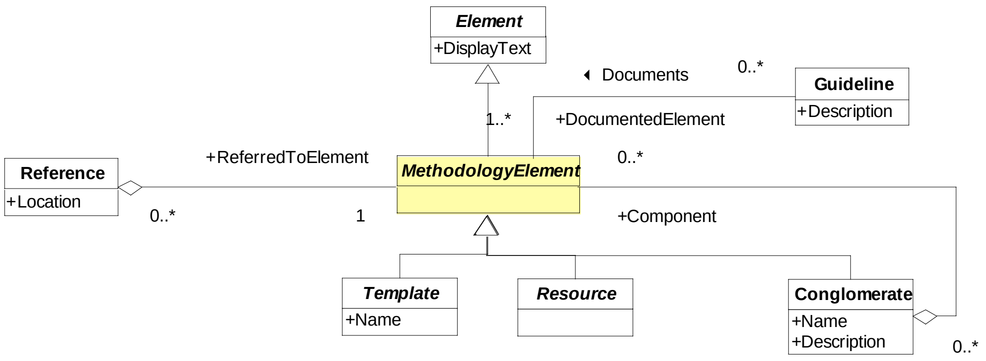
In a particular software development methodology, the detailed static structure of the system is represented using the object-oriented concepts of “class”, “attribute”, “generalization” and “association”. In order to capture this, the method engineer defines a language “Class Modelling Language” involving the above listed concepts plus their semantics and the relationships amongst them.

7.1.19 MethodologyElement

A methodology element is *an element that belongs in the methodology domain*. Any element created by a method engineer while constructing a new methodology (or extending an existing one) is represented by MethodologyElement.

MethodologyElement is an abstract subclass of Element, specialized into Resource, Template and Conglomerate.

This is a high-level class.



7.1.19.1 Attributes

This class has no attributes of its own.

7.1.19.2 Relationships

Name	Role	To class	Semantics
n/a	Component	Conglomerate	A methodology element may be a component of a number of conglomerates.
n/a	Documented-Element	Guideline	A methodology element may be documented by guidelines.
n/a	ReferredTo-Element	Reference	A methodology element can be referred to by references.

7.1.19.3 Example

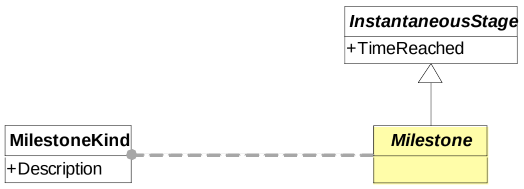
This class is too abstract to give a concrete example. Please see the examples for any of the subtypes of MethodologyElement.

7.1.20 Milestone

A milestone is *an instantaneous stage that marks some significant event in the endeavour*. Milestones may be used to mark the delivery of significant work products.

Milestone is an abstract subclass of InstantaneousStage.

This is a process-related class.



7.1.20.1 Attributes

This class has no attributes of its own.

7.1.20.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	MilestoneKind	A milestone in the endeavour domain is always of some milestone kind defined in the methodology domain.

7.1.20.3 Example

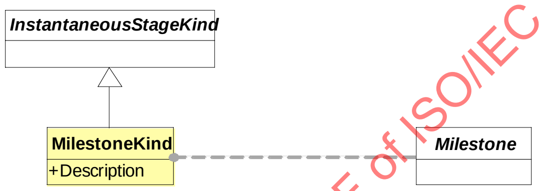
During a certain project for which John is project manager, the system definition phase is approaching completion. Once the system is fully defined, the system construction phase will take place. In order to manage the transition between phases, John tracks the point in time in which system definition is complete and stable, which marks said transition. This point in time is a milestone.

7.1.21 MilestoneKind

A milestone kind is *a specific kind of milestone, characterized by its specific purpose and kind of event that it signifies.*

MilestoneKind is a subclass of InstantaneousStageKind.

This is a process-related class.



7.1.21.1 Attributes

Name	Type	Semantics
Description	String	The description of the event signified by milestones of this kind.

7.1.21.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Milestone	A milestone in the endeavour domain is always of some milestone kind defined in the methodology domain.

7.1.21.3 Example

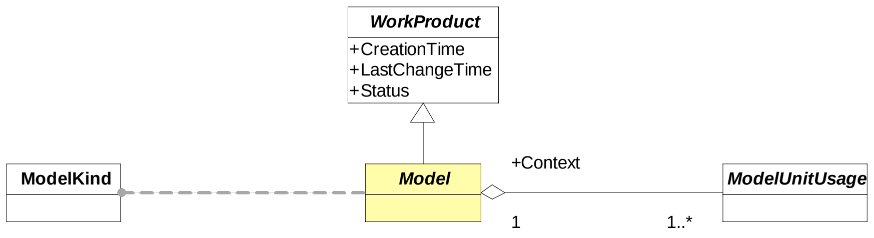
In a given methodology, two large phase kinds are defined in order to describe the definition of the system as opposed to the construction of the system. The transition between phases of these kinds needs to be marked by a managed point in time with specific semantics, and therefore the milestone kind "System definition is complete and stable" is defined by the method engineer to represent this fact.

7.1.22 Model

A model is *an abstract representation of some subject that acts as the subject's surrogate for some well defined purpose*. Notice that models are abstract constructs and therefore they are not visible or directly manageable. Documents are the perceivable, communicable counterparts of models.

Model is an abstract subclass of WorkProduct.

This is a product-related class.



7.1.22.1 Attributes

This class has no attributes of its own.

7.1.22.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	ModelKind	A model in the endeavour domain is always of some model kind defined in the methodology domain.
n/a	Context	ModelUnitUsage	A model is the context for some number of model unit usages.

7.1.22.3 Example

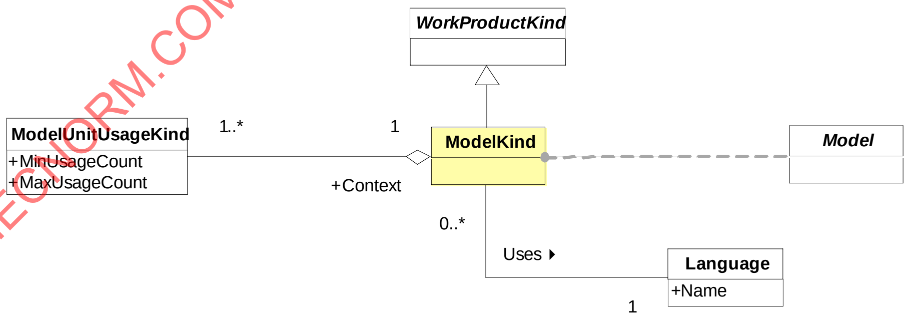
During the requirements process in a software development project, a list of candidate classes is made. After, during high-level modelling, this list is refined and classes are determined. Also, classes are fleshed out, and attributes, generalizations and associations are added. The resulting abstract construct is repeatedly used along the project to assist with other tasks, being different in different ways depending on each moment's needs. This construct is a model.

7.1.23 ModelKind

A model kind is *a specific kind of model, characterized by its focus, purpose and level of abstraction.*

ModelKind is a subclass of WorkProductKind.

This is a product-related class.



7.1.23.1 Attributes

This class has no attributes of its own.

7.1.23.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Model	A model in the endeavour domain is always of some model kind defined in the methodology domain.

Name	Role	To class	Semantics
n/a	Context	ModelUnitUsage-Kind	A model kind is defined to be the context for some number of model unit usage kinds.
Uses	n/a	Language	A model kind uses a certain language.

7.1.23.3 Example

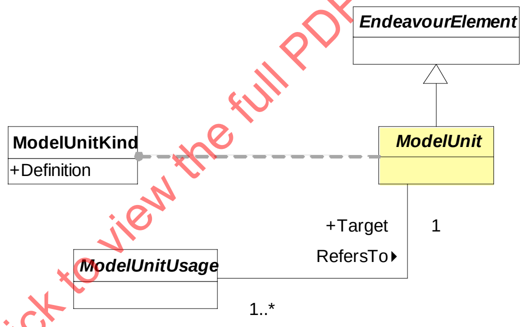
In a particular software development methodology, both the structure and behaviour of the software system to be built need to be described. Each time the methodology is enacted, the system's structure will be represented by a structural model, and the system's behaviour will be represented by a dynamic model. To capture this, the method engineer defines two model kinds, "Structural Model" and "Dynamic Model".

7.1.24 ModelUnit

A model unit is *an atomic component of a model, which represents a cohesive fragment of information in the subject being modelled*. Model units are usually linked to each other to form the semantic network that comprises the model. Furthermore, each model unit can appear in multiple models, thus achieving model connectivity.

ModelUnit is an abstract subclass of EndeavourElement.

This is a product-related class.



7.1.24.1 Attributes

This class has no attributes of its own.

7.1.24.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	ModelUnitKind	A model unit in the endeavour domain is always of some model unit kind defined in the methodology domain.
n/a	Target	ModelUnitUsage	A model unit is always the target of one or more model unit usages.

7.1.24.3 Example

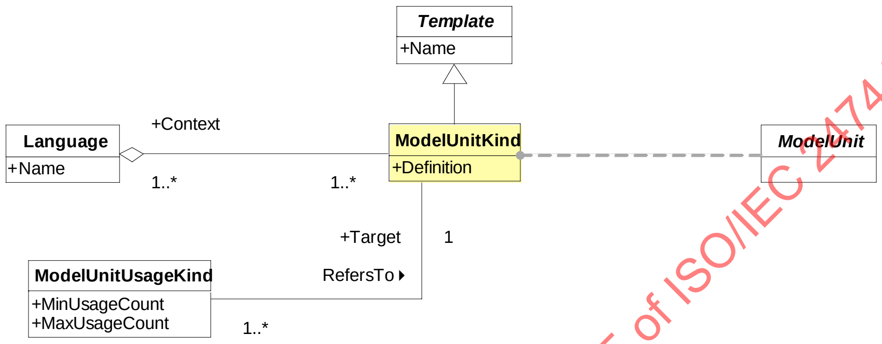
The class model of a software system contains a number of classes and associations. Each class, in turn, contains attributes and operations. Each class, association, attribute and operation in the model is a model unit.

7.1.25 ModelUnitKind

A model unit kind is *a specific kind of model unit, characterized by the nature of the information it represents and the intention of using such a representation.* Some model unit kinds, such as the conventional Class or Association, represent structural aspects of the subject being modelled, while others such as Service or Operation focus on its behavioural aspects. Other perspectives can be considered by using additional model unit kinds.

ModelUnitKind is a subclass of Template.

This is a product-related class.



7.1.25.1 Attributes

Name	Type	Semantics
Definition	String	The definition of this model unit kind.

7.1.25.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	ModelUnit	A model unit in the endeavour domain is always of some model unit kind defined in the methodology domain.
n/a	Target	ModelUnitUsage-Kind	A model unit kind is always the target of one or more model unit usage kinds.
n/a	n/a	Language	A model unit kind is a component of one or more languages.

7.1.25.3 Example

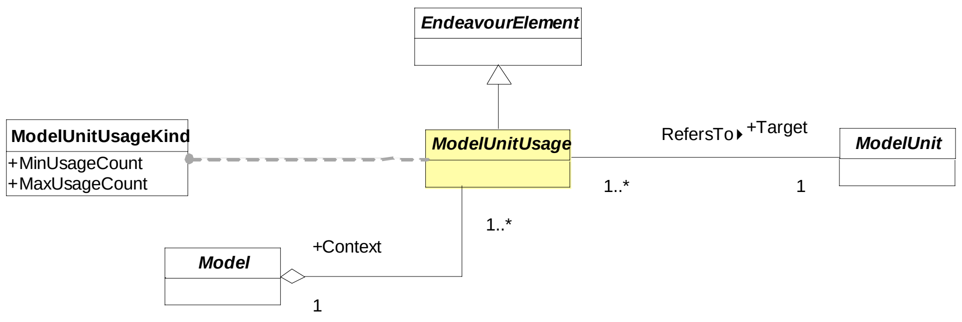
In a particular distributed systems development methodology, both object- and agent-oriented approaches are contemplated. Therefore, developers using this methodology would be able to use concepts such as “object” and “class” (typically object-oriented) to model the system under development, but also they could use “agent” and “message” (typically agent-oriented). To define the precise semantics of each of these concepts and how they relate to each other, the method engineer introduces the model unit kinds “Object”, “Class”, “Agent” and “Message”.

7.1.26 ModelUnitUsage

A model unit usage is *a specific usage of a given model unit by a given model*. Multiple models often include the same model units to achieve connectivity across all the representations of the subject being modelled.

ModelUnitUsage is an abstract subclass of EndeavourElement.

This is a product-related class.



7.1.26.1 Attributes

This class has no attributes of its own.

7.1.26.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	ModelUnitUsage-Kind	A model unit usage in the endeavour domain is always of some model unit usage kind defined in the methodology domain.
RefersTo	n/a	ModelUnit	A model unit usage always refers to a given model unit.
n/a	n/a	Model	A model unit usage always belongs to a given model.

7.1.26.3 Example

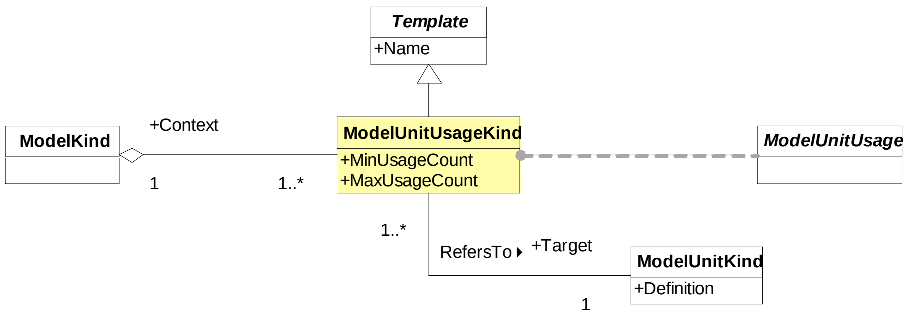
The class model of a word processing system contains classes named Document and Printer. Mary then constructs a state chart model for the Printer class. Both the class models and the Printer state chart model involve the Printer class. In addition, the class model involves the Document class, while the Printer state chart model involves a number of additional model units. Each of the relationships between a given model and a given model unit is a model unit usage.

7.1.27 ModelUnitUsageKind

A model unit usage kind is *a specific kind of model unit usage, characterized by the nature of the use that a given model kind makes of a given model unit kind.*

ModelUnitUsageKind is a subclass of Template.

This is a product-related class.



7.1.27.1 Attributes

Name	Type	Semantics
MinUsageCount	Integer	Minimum number of model units of the associated kind that can appear in a model of the associated kind.

Name	Type	Semantics
MaxUsageCount	Integer	Maximum number of model units of the associated kind that can appear in a model of the associated kind.

7.1.27.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	ModelUnitUsage	A model unit usage in the endeavour domain is always of some model unit usage kind defined in the methodology domain.
RefersTo	n/a	ModelUnitKind	A model unit usage kind always refers to a given model unit kind.
n/a	n/a	ModelKind	A model unit usage kind always belongs to a given model kind.

7.1.27.3 Example

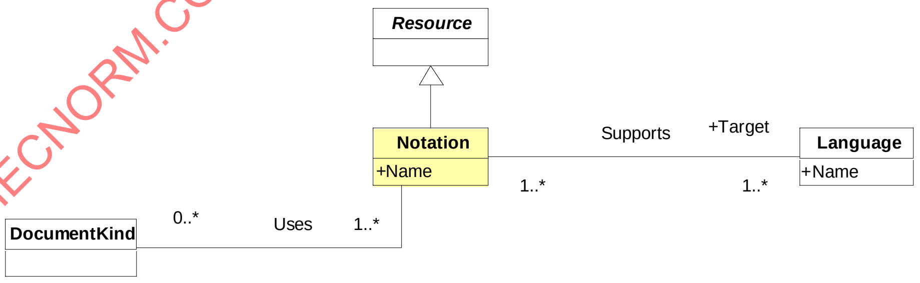
In a particular software development methodology, two model kinds exist: “Service Model”, which describes how the system will fulfil service requests from users for a given service, and “Class Model”, which describes the detailed structure of the whole system. At the same time, the following model unit kinds exist: “Class”, “Service” and “State”. In order to capture how model units of these kinds will be used by models of each of the defined kinds, the method engineer needs to define several model unit usage kinds. First, a model unit usage kind is introduced with Context = ClassModel, Target = Class, MinUsageCount = 1 and MaxUsageCount = *n*, meaning that class models will use as many classes as necessary. Secondly, another model unit usage kind is introduced with Context = ServiceModel, Target = Service and MinUsageCount = MaxUsageCount = 1, meaning that each service model will describe one and only one service. Additional model unit usage kinds would be defined similarly.

7.1.28 Notation

A notation is *a concrete syntax, usually graphical, that can be used to depict models created with certain languages*. Different notations may focus on different aspects of the same language, or support more than one language.

Notation is a subclass of Resource.

This is a product-related class.



7.1.28.1 Attributes

Name	Type	Semantics
Name	String	The name of the notation.

7.1.28.2 Relationships

Name	Role	To class	Semantics
Supports	n/a	Language	A notation always supports at least one language.
n/a	n/a	DocumentKind	A notation may be used by certain document kinds.

7.1.28.3 Example

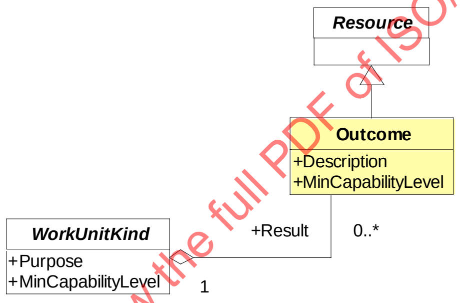
In a particular software development methodology, the detailed static structure of the system is represented using the language “Class Modelling Language”, which involves model unit kinds such as “Class”, “Attribute” and “Association”. In order to represent instances of these concepts when the methodology is enacted, a notation needs to be introduced that can support the above mentioned language. Therefore, the method engineer defines the notation “Class Diagrams” with Target = ClassModellingLanguage.

7.1.29 Outcome

An outcome is *an observable result of the successful performance of any work unit of a given kind*. Outcomes are often used to assess the performance of work units, since their presence define success. An unsuccessful work unit may or may not exhibit its defined outcomes; a successful work unit, by definition, will exhibit all of them.

Outcome is a subclass of Resource.

This is a process-related class.



7.1.29.1 Attributes

Name	Type	Semantics
Description	String	The description of the observable result.
MinCapabilityLevel	Integer	The minimum capability level at which the outcome may be considered. Enactments at capability levels lower than this level should not take this outcome into account.

7.1.29.2 Relationships

Name /	Role	To class	Semantics
n/a	Result	WorkUnitKind	An outcome is a result of work units of a particular kind.

7.1.29.3 Example

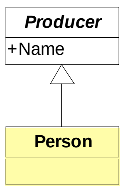
In a given methodology, the outcomes “All project stakeholders are aware of the scope of the system” and “The system requirements specification has been created and approved” are defined an attached to the process kind “Requirements Engineering” in order to describe the expected result of its successful performance when the methodology is enacted.

7.1.30 Person

A person is *an individual human being involved in a development effort.*

Person is a subclass of Producer.

This is a producer-related class.



7.1.30.1 Attributes

This class has no attributes of its own.

7.1.30.2 Relationships

This class has no relationships of its own.

7.1.30.3 Example

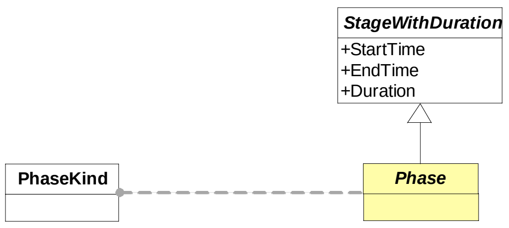
During a certain project, Mary leads a team of developers to construct a product. Each of the developers in the team, as well as Mary, are persons.

7.1.31 Phase

A phase is *a stage with duration for which the objective is the transition between cognitive frameworks.* Phases usually add detail to a previously existing set of work products.

Phase is an abstract subclass of StageWithDuration.

This is a process-related class.



7.1.31.1 Attributes

This class has no attributes of its own.

7.1.31.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	PhaseKind	A phase in the endeavour domain is always of some phase kind defined in the methodology domain.

7.1.31.3 Example

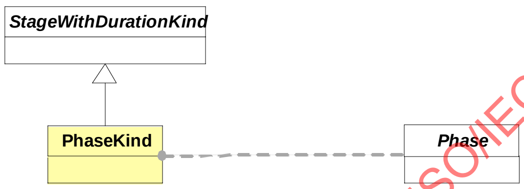
During a development project, the system is first defined and then, after definition is complete, the system is incrementally constructed. Each of these two large timeframes is a phase.

7.1.32 PhaseKind

A phase kind is *a specific kind of phase, characterized by the abstraction level and formality of the result that it aims to produce.*

PhaseKind is a subclass of StageWithDurationKind.

This is a process-related class.



7.1.32.1 Attributes

This class has no attributes of its own.

7.1.32.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Phase	A phase in the endeavour domain is always of some phase kind defined in the methodology domain.

7.1.32.3 Example

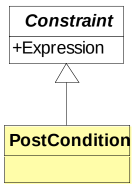
In a given methodology, the product to be built is first defined and then incrementally constructed. To capture this, the method engineer introduces the phase kinds “System Definition” and “System Construction”.

7.1.33 PostCondition

A postcondition is *a constraint that is guaranteed to be satisfied after an action of the associated kind is performed.*

PostCondition is a subclass of Constraint.

This is a process- and product-related class.



7.1.33.1 Attributes

This class has no attributes of its own.

7.1.33.2 Relationships

This class has no relationships of its own.

7.1.33.3 Example

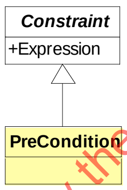
In a given methodology, a task kind "Sign off requirements" is defined, together with a work product kind "Requirements Specification". Both are related by the fact that tasks of the "Sign off requirements" kind will, when executed, change a work product of the "Requirements Specification" kind to the "Approved" state. Said relationship is modelled as an action kind with Type = Modify. However, the method engineer wants to capture the fact that signing off requirements means that the status of the associated "Requirements Specification" work product is changed to "Approved". This is done by creating a postcondition with Expression = (RequirementsSpecification.WorkProductStatus is Approved) and attaching this constraint to the above mentioned action. Thus, executing the task will guarantee that the stated condition is met.

7.1.34 PreCondition

A precondition is *a constraint that must be satisfied before an action of the associated kind can be performed.*

PreCondition is a subclass of Constraint.

This is a process- and product-related class.



7.1.34.1 Attributes

This class has no attributes of its own.

7.1.34.2 Relationships

This class has no relationships of its own.

7.1.34.3 Example

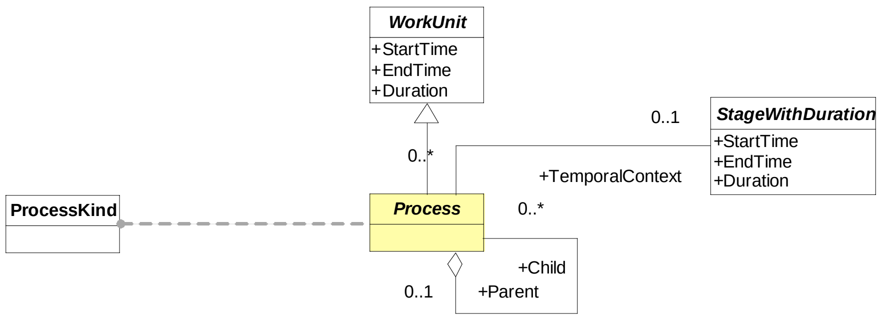
In a given methodology, a task kind "Deliver user documentation" is defined, together with a work product kind "User Documentation". Both are related by the fact that tasks of the "Deliver user documentation" kind will, when executed, use work products of the "User Documentation" kind. Said relationship is modelled as an action kind with Type = ReadOnly. However, the method engineer wants to capture the need that user documentation is only delivered after the associated "User Interface Specification" work product has been approved. This is done by creating a precondition with Expression = (UserInterfaceSpecification.WorkProduct-Status is Approved) and attaching this constraint to the above mentioned action. Thus, the action will not be allowed to execute unless the required condition is met.

7.1.35 Process

A process is *a large-grained work unit that operates within a given area of expertise.*

Process is an abstract subclass of WorkUnit.

This is a process-related class.



7.1.35.1 Attributes

This class has no attributes of its own.

7.1.35.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	ProcessKind	A process in the endeavour domain is always of some process kind defined in the methodology domain.
n/a	n/a	StageWith-Duration	A process can be performed within a particular stage with duration.
n/a	ParentProcess	Process	A process may be the parent of sub-processes.
n/a	SubProcess	Process	A process may be a sub-process of a number of parent processes.

7.1.35.3 Example

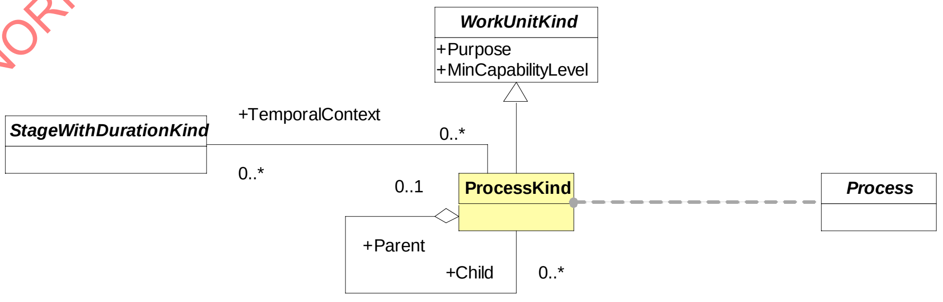
In an engineering project, developer Mary leads the team in charge of quality assurance. She must ensure that products generated in the project maintain the minimum levels of quality at all times. Such a focused job, as performed by Mary's team, is a process.

7.1.36 ProcessKind

A process kind is *a specific kind of process, characterized by the area of expertise in which it occurs.*

ProcessKind is a subclass of WorkUnitKind.

This is a process-related class.



7.1.36.1 Attributes

This class has no attributes of its own.

7.1.36.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Process	A process in the endeavour domain is always of some process kind defined in the methodology domain.
n/a	n/a	StageWith-DurationKind	A process kind can be defined to be performed within stages with duration of some particular kinds.
n/a	ParentProcess-Kind	ProcessKind	A process kind may be the parent of sub-process kinds.
n/a	SubProcessKind	ProcessKind	A process kind may be a sub-process kind of a number of parent process kinds.

7.1.36.3 Example

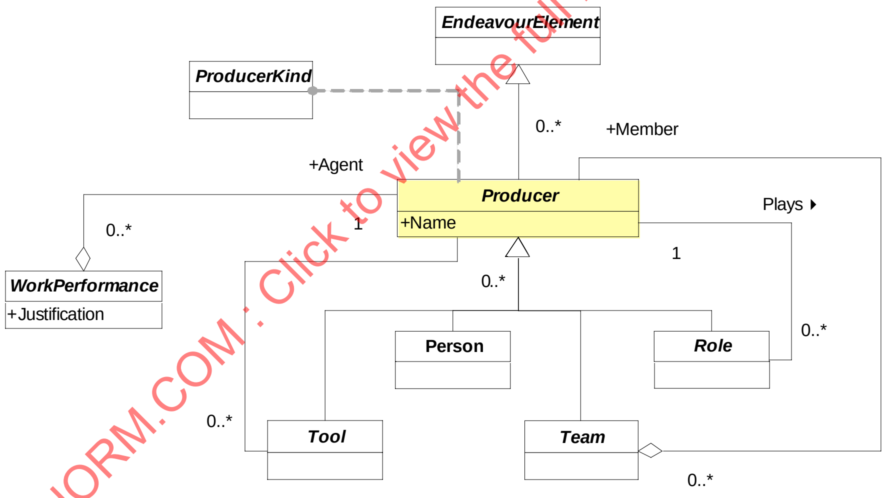
In a given methodology, the process kind "Quality Assurance" is defined to represent the fact that, when said methodology is enacted, somebody will have to take the ongoing responsibility of quality assurance.

7.1.37 Producer

A producer is *an agent that has the responsibility to execute work units*. Producers are usually people or groups of people, but can also be tools.

Producer is an abstract subclass of EndeavourElement, specialized into Person, Tool, Team and Role.

This is a producer-related class.



7.1.37.1 Attributes

Name	Type	Semantics
Name	String	The name of the producer. This is an abstract attribute and different subclasses of Producer can implement it differently.

7.1.37.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	ProducerKind	A producer in the endeavour domain is always of some producer kind defined in the methodology domain.
Plays	n/a	Role	A producer may play a number of roles.
IsInvolvedIn-Performance	Agent	Work-Performance	A producer may be involved in a number of work performances.
n/a	Member	Team	A producer may be a member of a team.
IsAssistedBy	n/a	Tool	A producer may be assisted by a set of tools.

7.1.37.3 Example

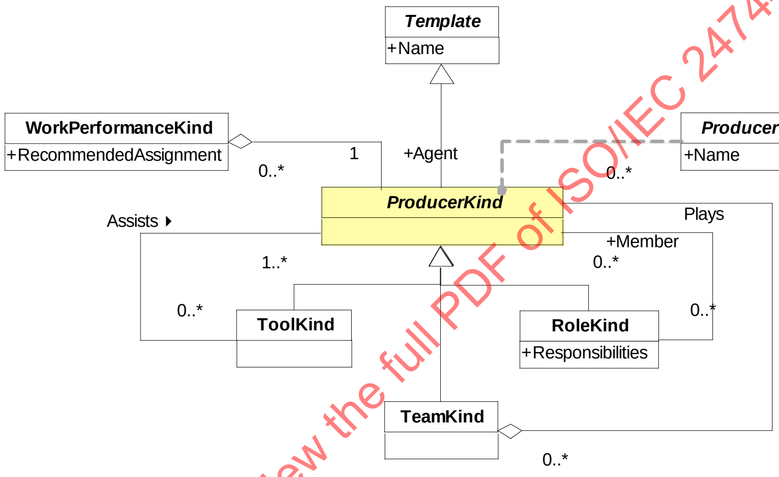
During a certain software development project, John produces some source code and a few XML specifications. A code generator tool reads these specifications and generates some additional code from them. Both John and the tool are producers.

7.1.38 ProducerKind

A producer kind is *a specific kind of producer, characterized by its area of expertise.*

ProducerKind is an abstract subclass of Template, specialized into ToolKind, TeamKind and RoleKind.

This is a producer-related class.



7.1.38.1 Attributes

This class has no attributes of its own.

7.1.38.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Producer	A producer in the endeavour domain is always of some producer kind defined in the methodology domain.
Plays	n/a	RoleKind	A producer kind may be assigned to play one or more role kinds.
IsInvolvedIn-Performance	Agent	Work-PerformanceKind	A producer kind may be involved in a number of work performance kinds.
n/a	Member	TeamKind	A producer kind may be a member of a team kind.
IsAssistedBy	n/a	ToolKind	Producers of a particular kind are assisted by tools of some particular kinds.

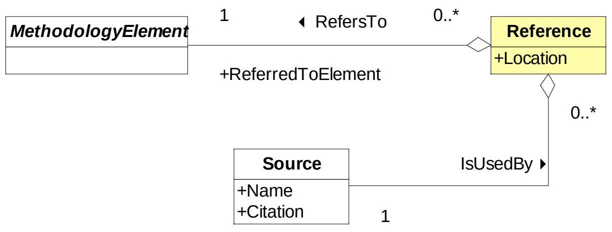
7.1.38.3 Example

In a given methodology, a producer kind “Quality Assurance Team” (a team kind) is defined and linked to a work performance kind that points to the “Quality Assurance” process kind. This is to represent the fact that, when said methodology is enacted, a group of people will have to take the responsibility of performing work units of the “Quality Assurance” kind.

7.1.39 Reference

A reference is *a specific linkage between a given methodology element and a given source*. References implement the many-to-many mapping between MethodologyElement and Source.

This is a support class.



7.1.39.1 Attributes

Name	Type	Semantics
Location	String	The specific location of the information relating to the documented element within the associated source.

7.1.39.2 Relationships

Name	Role	To class	Semantics
RefersTo	n/a	Methodology-Element	The reference refers to a given methodology element.
Uses	n/a	Source	The source providing information for this reference.

7.1.39.3 Example

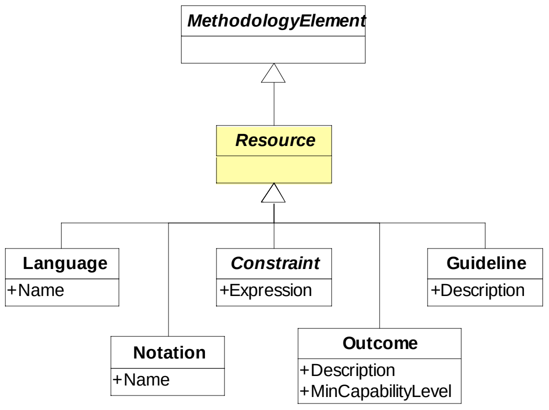
A method engineer decides to incorporate the methodology element "Dialog Design" (a technique kind) to a method fragment repository. Since the description of the technique in the book by Henderson-Sellers, Simons and Younessi (a source, called "HendersonSellers98a") is interesting, the method engineer decides to attach a reference to said technique kind that precisely locates the documented element within the source. This is achieved by creating a reference with ReferredToElement = DialogDesign, Source = HendersonSellers98a and Location = "pages 182-188".

7.1.40 Resource

A resource is *a methodology element that is directly used at the endeavour level, without an instantiation process*. Any methodology element that serves as a reference or guideline during an endeavour is represented by Resource.

Resource is an abstract subclass of MethodologyElement, specialized into Language, Notation, Constraint, Outcome and Guideline.

This is a high-level class.



7.1.40.1 Attributes

This class has no attributes of its own.

7.1.40.2 Relationships

This class has no relationships of its own.

7.1.40.3 Example

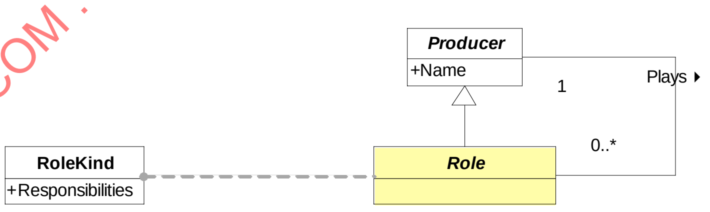
This class is too abstract to give a concrete example. Please see the examples for any of the subtypes of Resource.

7.1.41 Role

A role is *a collection of responsibilities that a producer can take*. Roles are often used to declare what responsibilities must be addressed without deciding on how they will be implemented.

Role is an abstract subclass of Producer.

This is a producer-related class.



7.1.41.1 Attributes

This class has no attributes of its own.

7.1.41.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	RoleKind	A role in the endeavour domain is always of some role kind defined in the methodology domain.
n/a	n/a	Producer	A role is played by a given producer.

7.1.41.3 Example

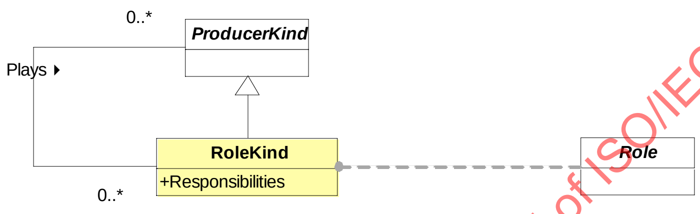
During a certain project, Mary is in charge of writing the user documentation. Mary leaves the project midway and John takes over with the same responsibilities. This collection of responsibilities, which could be called "technical writer", is a role.

7.1.42 RoleKind

A role kind is *a specific kind of role, characterized by the involved responsibilities*. Different role kinds usually address different needs and make use of different skills.

RoleKind is a subclass of ProducerKind.

This is a producer-related class.



7.1.42.1 Attributes

Name	Type	Semantics
Responsibilities	String	The responsibilities of this role kind.

7.1.42.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Role	A role in the endeavour domain is always of some role kind defined in the methodology domain.
n/a	n/a	ProducerKind	A role kind is defined to be possibly played by different producer kinds.

7.1.42.3 Example

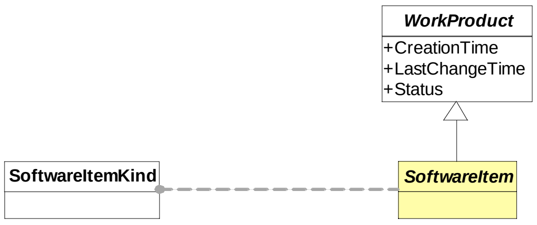
In a given methodology, it is necessary that close contact is maintained with the customers. To capture this independently of any person or group in particular, the method engineer introduces a role kind "Customer Liaison".

7.1.43 SoftwareItem

A software item is *a piece of software of interest to the endeavour*.

SoftwareItem is an abstract subclass of WorkProduct.

This is a product-related class.



7.1.43.1 Attributes

This class has no attributes of its own.

7.1.43.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	SoftwareItem-Kind	A software item in the endeavour domain is always of some software item kind defined in the methodology domain.

7.1.43.3 Example

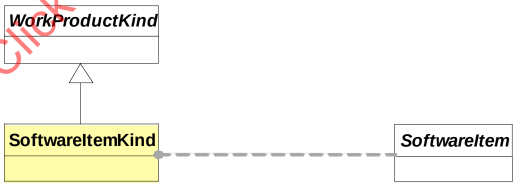
During a software development project, a team of developers (with the help of some tools) produce the executable program. John produces the online help files and Mary creates an initial database dump. The executable program, the online help files and the database dump are software items.

7.1.44 SoftwareItemKind

A software item kind is *a specific kind of software item, characterized by its scope, requirements and features.*

SoftwareItemKind is a subclass of WorkProductKind.

This is a product-related class.



7.1.44.1 Attributes

This class has no attributes of its own.

7.1.44.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	SoftwareItem	A software item in the endeavour domain is always of some software item kind defined in the methodology domain.

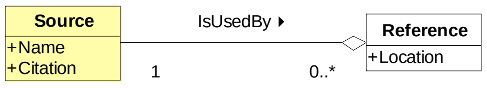
7.1.44.3 Example

In a particular systems development methodology, the software item kind “Operating System” is defined to represent the fact that, when said methodology is enacted, software items of such kind will be used in different ways.

7.1.45 Source

A source is *a source of information, experience or best practices*. This class is often used by method engineers to represent books, articles or other sources of documentation useful to track information related to elements.

This is a support class.



7.1.45.1 Attributes

Name	Type	Semantics
Name	String	The name of the source.
Citation	String	The formal citation of the source, often used with bibliographic sources.

7.1.45.2 Relationships

Name	Role	To class	Semantics
IsUsedBy	n/a	Reference	A source may be used by a number of references.

7.1.45.3 Example

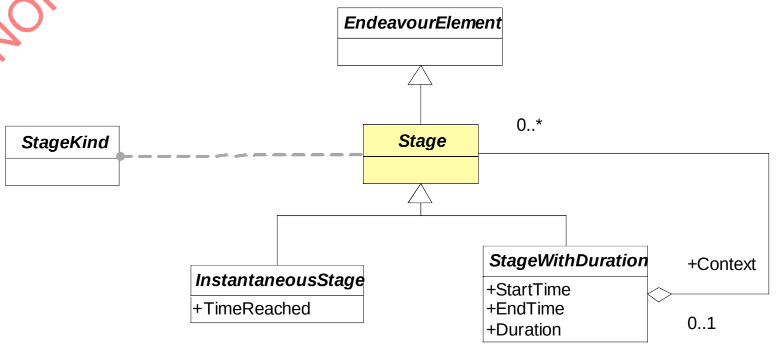
A method engineer decides to incorporate several technique kinds from the book by Henderson-Sellers, Simons and Younessi to a method fragment repository. Since the description of the techniques in the book is interesting, the method engineer decides to create the source "HendersonSellers98a" (with the appropriate citation) and link it to the above mentioned technique kinds via references.

7.1.46 Stage

A stage is *a managed time frame within an endeavour.*

Stage is an abstract subclass of EndeavourElement, specialized into StageWithDuration and Instantaneous-Stage.

This is a process-related class.



7.1.46.1 Attributes

This class has no attributes of its own.

7.1.46.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	StageKind	A stage in the endeavour domain is always of some stage kind defined in the methodology domain.
n/a	n/a	StageWith-Duration	A stage may take place within a stage with duration.

7.1.46.3 Example

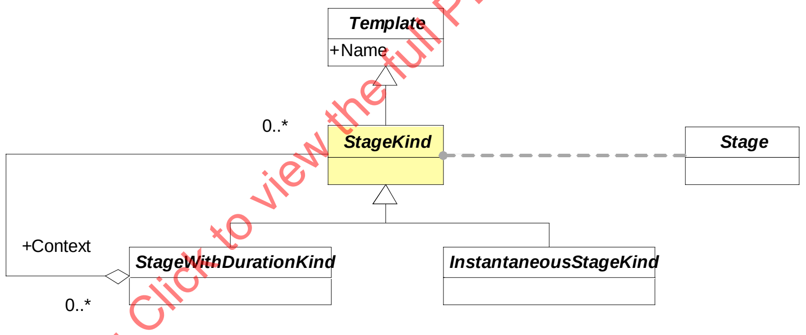
During a certain project, the product to be built is first defined and then incrementally constructed. These two managed time frames are stages (with duration). To manage the transition between definition and construction, a project-managed point in time between these phases is considered. This point in time is another (instantaneous) stage.

7.1.47 StageKind

A stage kind is *a specific kind of stage, characterized by the abstraction level at which it works on the endeavour and the result that it aims to produce.*

StageKind is an abstract subclass of Template, specialized into StageWithDurationKind and Instantaneous-StageKind.

This is a process-related class.



7.1.47.1 Attributes

This class has no attributes of its own.

7.1.47.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Stage	A stage in the endeavour domain is always of some stage kind defined in the methodology domain.
n/a	n/a	StageWith-DurationKind	A stage kind may take place within a number of stage with duration kinds.

7.1.47.3 Example

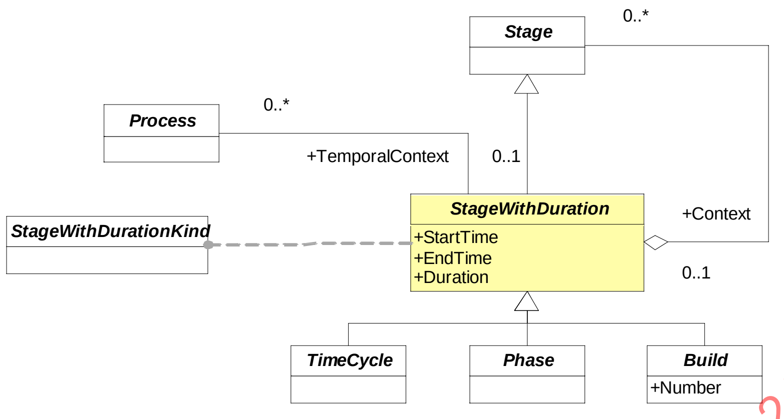
In a given methodology, the product to be built is first defined and then incrementally constructed. To capture this, the method engineer introduces the stage kinds “System Definition” and “System Construction” (phase kinds) separated by the stage kind “System definition is complete and stable” (a milestone kind).

7.1.48 StageWithDuration

A stage with duration is *a managed interval of time within an endeavour.*

StageWithDuration is an abstract subclass of Stage, specialized into TimeCycle, Phase and Build.

This is a process-related class.



7.1.48.1 Attributes

Name	Type	Semantics
StartTime	Timestamp	The point in time at which the stage with duration is started.
EndTime	Timestamp	The point in time at which the stage with duration is finished.
Duration	Timespan	The span of time between the start time and the end time.

7.1.48.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	StageWith-DurationKind	A stage with duration in the endeavour domain is always of some stage with duration kind defined in the methodology domain.
n/a	TemporalContext	Process	A stage with duration is the temporal context in which a set of processes take place.
n/a	Context	Stage	A stage with duration may be the context in which a number of other stages take place.

7.1.48.3 Example

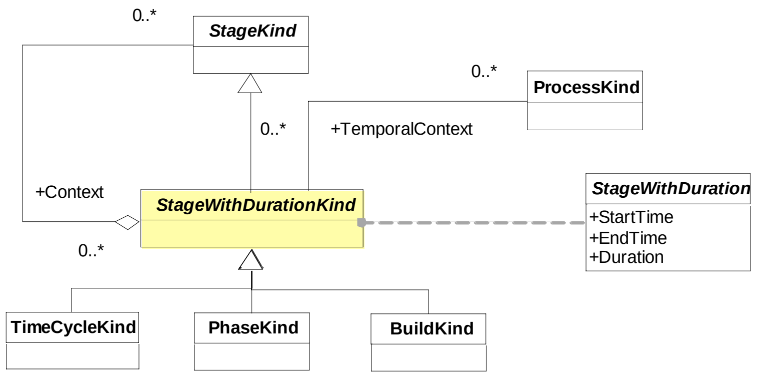
During a certain project, the product to be built is first defined and then incrementally constructed. These two managed time frames are stages with duration (phases). Within the construction stage with duration, a sequence of smaller stages is performed in order to incrementally build the product. Each of these smaller stages is a stage with duration too (a build).

7.1.49 StageWithDurationKind

A stage with duration kind is *a specific kind of stage with duration, characterized by the abstraction level at which it works on the endeavour and the result that it aims to produce.*

StageWithDurationKind is an abstract subclass of StageKind, specialized into TimeCycleKind, PhaseKind and BuildKind.

This is a process-related class.



7.1.49.1 Attributes

This class has no attributes of its own.

7.1.49.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	StageWith-Duration	A stage with duration in the endeavour domain is always of some stage with duration kind defined in the methodology domain.
n/a	TemporalContext	ProcessKind	A stage with duration kind is the temporal context in which a set of process kinds may take place.
n/a	Context	StageKind	A stage with duration kind may be the context in which a number of other stage kinds take place.

7.1.49.3 Example

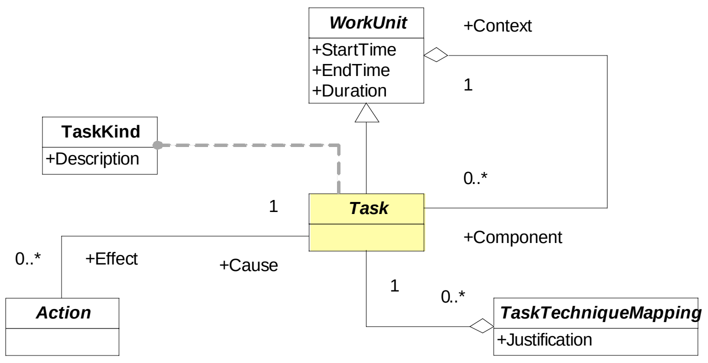
In a given methodology, the product to be built is first defined and then incrementally constructed. To capture this, the method engineer introduces the stage with duration kinds “System Definition” and “System Construction” (phase kinds).

7.1.50 Task

A task is *a small-grained work unit that focuses on what must be done in order to achieve a given purpose.*

Task is an abstract subclass of WorkUnit.

This is a process-related class.



7.1.50.1 Attributes

This class has no attributes of its own.

7.1.50.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	TaskKind	A task in the endeavour domain is always of some task kind defined in the methodology domain.
n/a	Component	WorkUnit	A task always occurs within a particular work unit.
n/a	n/a	TaskTechnique-Mapping	A task may be involved in a number of task-technique mappings.
n/a	Cause	Action	A task may cause actions.

7.1.50.3 Example

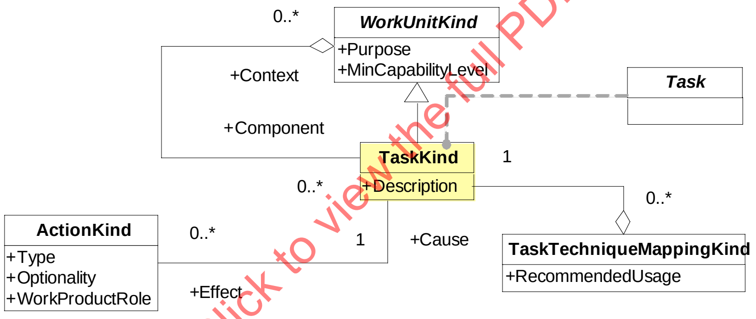
During a software development project, Mary identifies the candidate classes in the system, and then she tries to find the relationships between them, updating the class list in the process. Finally, Mary defines some attributes and operations on the classes. Each of these individual pieces of work is a task.

7.1.51 TaskKind

A task kind is *a specific kind of task, characterized by its purpose within the endeavour*.

TaskKind is a subclass of WorkUnitKind.

This is a process-related class.



7.1.51.1 Attributes

Name	Type	Semantics
Description	String	The description of what is to be done in order to accomplish the purpose.

7.1.51.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Task	A task in the endeavour domain is always of some task kind defined in the methodology domain.
n/a	Component	WorkUnitKind	A task kind is defined to occur within some particular work unit kinds.
n/a	n/a	TaskTechnique-MappingKind	A task kind may be involved in a number of task-technique mapping kinds.
n/a	Cause	ActionKind	A task kind may cause action kinds.

7.1.51.3 Example

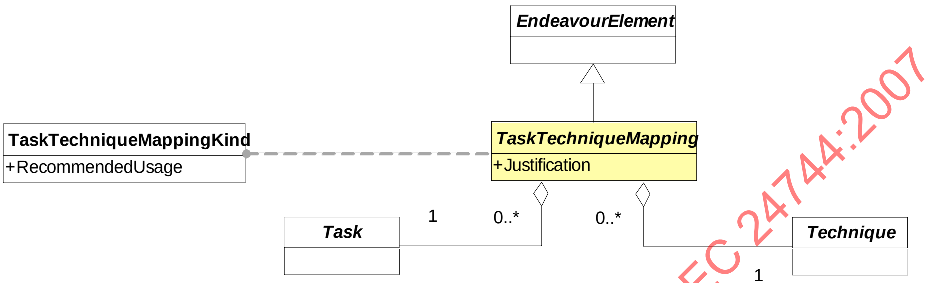
In a particular software development methodology, quality assurance is performed by reviewing each generated product and then comparing the number of defects found against historical data. To capture this, the method engineer defines the task kinds “Review work product” and “Compare defect report to historical data”.

7.1.52 TaskTechniqueMapping

A task-technique mapping is *a usage association between a given task and a given technique.* A task-technique mapping represents the fact that, at the endeavour domain, a given technique is being used to accomplish a given task.

TaskTechniqueMapping is an abstract subclass of EndeavourElement.

This is a process-related class.



7.1.52.1 Attributes

Name	Type	Semantics
Justification	String	The justification why the associated technique is chosen to accomplish the associated task.

7.1.52.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	TaskTechnique-MappingKind	A task-technique mapping in the endeavour domain is always of some task-technique mapping kind defined in the methodology domain.
n/a	n/a	Task	A task-technique mapping maps a particular task.
n/a	n/a	Technique	A task-technique mapping maps a particular technique.

7.1.52.3 Example

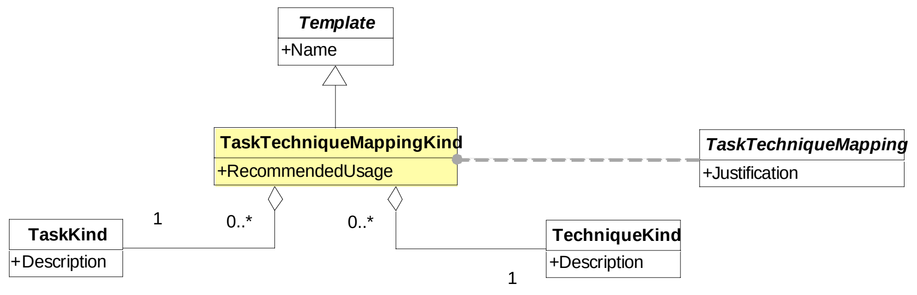
During a software development project, Mary needs to identify the candidate classes in the system. In order to do so, she checks the methodology being used and she sees that two kinds of techniques are recommended for tasks of this kind: “Text Analysis” and “CRC Cards”. Since Mary is working by herself, she thinks it is better to use text analysis. The particular association between the “Identify Classes” task that Mary is performing and the chosen technique “Text Analysis” (together with Mary’s justification for her decision) is a task-technique mapping.

7.1.53 TaskTechniqueMappingKind

A task-technique mapping kind is *a specific kind of task-technique mapping, characterized by the mapped task kind and technique kind.*

TaskKind is a subclass of Template.

This is a process-related class.



7.1.53.1 Attributes

Name	Type	Semantics
Recommended-Usage	DeonticValue	The recommended usage of techniques of the associated kind by tasks of the associated kind. See 7.2.2 for possible values.

7.1.53.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	TaskTechnique-MappingKind	A task-technique mapping in the endeavour domain is always of some task-technique mapping kind defined in the methodology domain.
n/a	n/a	TaskKind	A task-technique mapping kind maps a particular task kind.
n/a	n/a	TechniqueKind	A task-technique mapping kind maps a particular technique kind.

7.1.53.3 Example

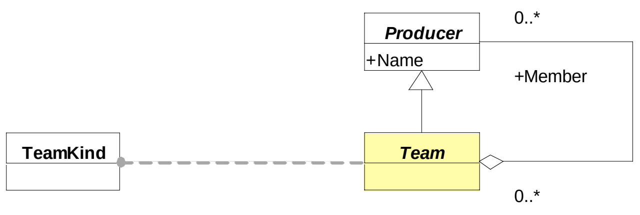
When certain software development methodology is enacted, tasks of kind “Elicit system requirements” can be performed by interviewing stakeholders and, optionally, by organizing focus groups. To capture this, the method engineer introduces the technique kinds “Stakeholder Interviews” and “Focus Groups”, and associates both with the above mentioned task kind via the appropriate task-technique mapping kinds, assigning recommended usage values of Recommended and Optional, respectively.

7.1.54 Team

A team is *an organized set of producers that collectively focus on common work units.*

Team is an abstract subclass of Producer.

This is a producer-related class.



7.1.54.1 Attributes

This class has no attributes of its own.

7.1.54.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	TeamKind	A team in the endeavour domain is always of some team kind defined in the methodology domain.
Context	n/a	Producer	A team is composed of producers.

7.1.54.3 Example

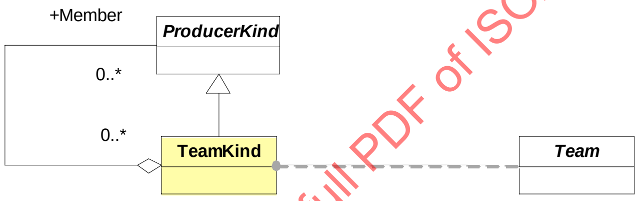
On a certain project, maintaining a close contact with the customer is so important that a single person is not enough for the task. A group of people is organized to deal with this task. This group of people is a team.

7.1.55 TeamKind

A team kind is *a specific kind of team, characterized by its responsibilities.*

TeamKind is a subclass of ProducerKind.

This is a producer-related class.



7.1.55.1 Attributes

This class has no attributes of its own.

7.1.55.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Team	A team in the endeavour domain is always of some team kind defined in the methodology domain.
Context	n/a	ProducerKind	A team kind is composed of producer kinds.

7.1.55.3 Example

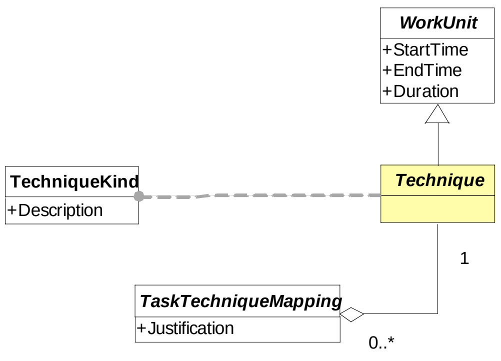
In a certain safety-critical software development methodology, quality assurance needs to be performed by a specially designated separate group of developers. To capture this, the method engineer defines the team kind “Quality Assurance Team”.

7.1.56 Technique

A technique is *a small-grained work unit that focuses on how the given purpose may be achieved.*

Technique is an abstract subclass of WorkUnit.

This is a process-related class.



7.1.56.1 Attributes

This class has no attributes of its own.

7.1.56.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	TechniqueKind	A technique in the endeavour domain is always of some technique kind defined in the methodology domain.
n/a	n/a	TaskTechnique-Mapping	A technique may be involved in a number of task-technique mappings.

7.1.56.3 Example

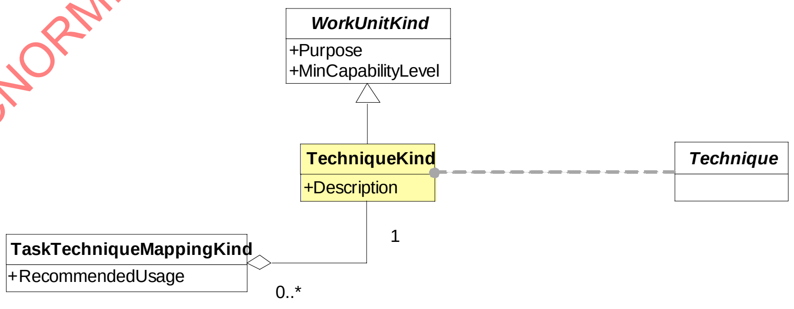
During a software development project, Mary needs to identify the classes in the system. In order to perform this task, she first analyses the text in some documents related to the system, then she organizes a CRC Card session, and finally she brainstorms with her colleagues trying to find additional classes. These three different ways of tackling the same task are techniques.

7.1.57 TechniqueKind

A technique kind is *a specific kind of technique, characterized by its purpose within the endeavour.*

TechniqueKind is a subclass of WorkUnitKind.

This is a process-related class.



7.1.57.1 Attributes

Name	Type	Semantics
Description	String	The description of how to accomplish the purpose.

7.1.57.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Technique	A technique in the endeavour domain is always of some technique kind defined in the methodology domain.
n/a	n/a	TaskTechnique-MappingKind	A technique kind may be involved in a number of task-technique mapping kinds.

7.1.57.3 Example

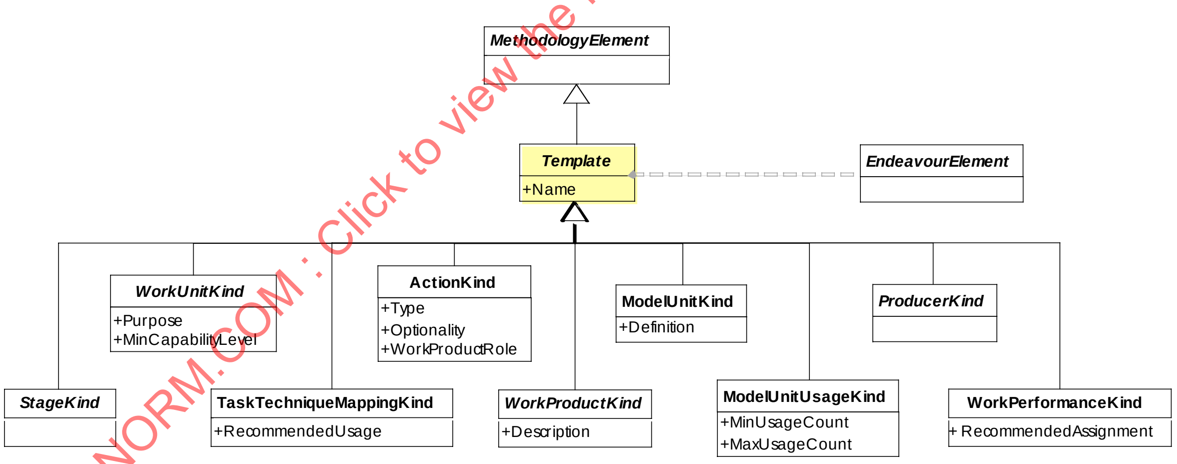
When certain software development methodology is enacted, tasks of kind “Elicit system requirements” can be performed by either interviewing stakeholders or organizing focus groups. To capture this, the method engineer introduces the technique kinds “Stakeholder Interviews” and “Focus Groups”, and associates both with the above mentioned task kind via the appropriate task-technique mapping kinds.

7.1.58 Template

A template is *a methodology element that is used at the endeavour level through an instantiation process*. Any methodology element that acts as a class to be instantiated during enactment as an endeavour element is represented by Template.

Template is an abstract subclass of MethodologyElement, specialized into StageKind, WorkUnitKind, Task-TechniqueMappingKind, ActionKind, WorkProductKind, ModelUnitKind, ModelUnitUsageKind, ProducerKind and WorkPerformanceKind.

This is a high-level class.



7.1.58.1 Attributes

Name	Type	Semantics
Name	String	The name of the class that would be instantiated during enactment.

7.1.58.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	EndeavourElement	An endeavour element is always of some template defined in the methodology domain.

7.1.58.3 Example

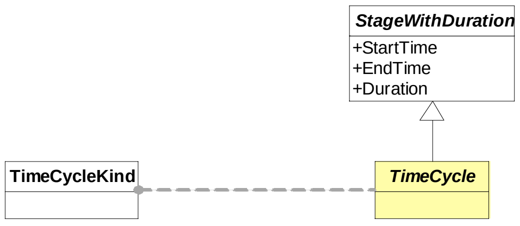
This class is too abstract to give a concrete example. Please see the examples for any of the subtypes of Template.

7.1.59 TimeCycle

A time cycle is *a stage with duration for which the objective is the delivery of a final product or service.*

TimeCycle is an abstract subclass of StageWithDuration.

This is a process-related class.



7.1.59.1 Attributes

This class has no attributes of its own.

7.1.59.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	TimeCycleKind	A time cycle in the endeavour domain is always of some time cycle kind defined in the methodology domain.

7.1.59.3 Example

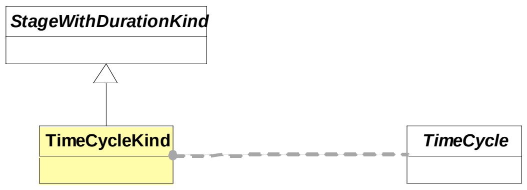
John has been commissioned to develop and deliver an experimental, strategically important product. He decides to organize a product-focused project and, in addition, establish a research and development line to do further investigation. Each of these major time-related arrangements is a different time cycle.

7.1.60 TimeCycleKind

A time cycle kind is *a specific kind of time cycle, characterized by the type of outcomes that it aims to produce*. Different time cycle kinds are usually utilized to account for different needs in the characteristics of endeavours and products.

TimeCycleKind is a subclass of StageWithDurationKind.

This is a process-related class.



7.1.60.1 Attributes

This class has no attributes of its own.

7.1.60.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	TimeCycle	A time cycle in the endeavour domain is always of some time cycle kind defined in the methodology domain.

7.1.60.3 Example

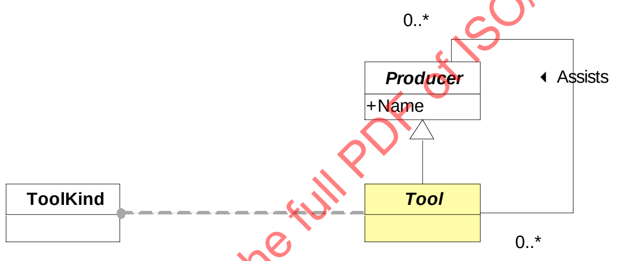
When certain methodology is enacted, final products are delivered by either executing product-focused projects or sustaining research & development lines for some time. To capture this, the method engineer introduces the time cycle kinds “SpecificProject” and “ResearchLine”. Each will comprise different stage kinds.

7.1.61 Tool

A tool is *an instrument that helps another producer to execute its responsibilities in an automated way.*

Tool is an abstract subclass of Producer.

This is a producer-related class.



7.1.61.1 Attributes

This class has no attributes of its own.

7.1.61.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	ToolKind	A tool in the endeavour domain is always of some tool kind defined in the methodology domain.
Assists	n/a	Producer	A tool may assist a set of producers.

7.1.61.3 Example

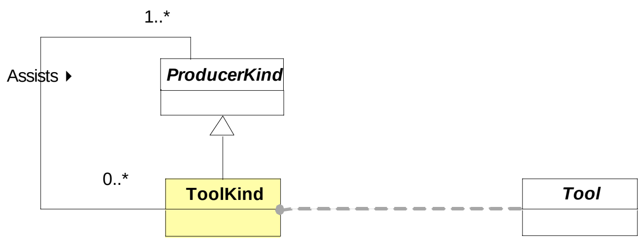
During a certain software development project, Mary writes some code and then compiles it. She then automatically creates unit test stubs for the code and, after completing the skeleton stubs, she unit tests her code. The code editor, the compiler and the automated unit tester are tools.

7.1.62 ToolKind

A tool kind is *a specific kind of tool, characterized by its features*. Different tool kinds are often used to represent different products such as diagram editors, integrated development environments, defect tracking systems, collaboration and messaging systems or code generators.

ToolKind is a subclass of ProducerKind.

This is a producer-related class.



7.1.62.1 Attributes

This class has no attributes of its own.

7.1.62.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	Tool	A tool in the endeavour domain is always of some tool kind defined in the methodology domain.
Assists	n/a	ProducerKind	Tools of a particular kind may assist producers of some particular kinds.

7.1.62.3 Example

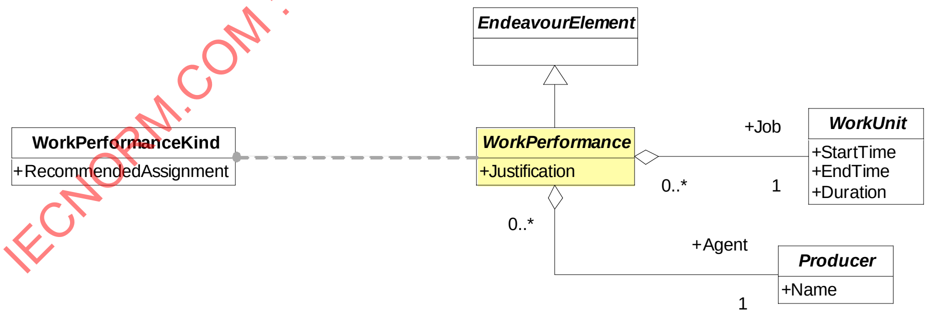
In a particular software development methodology, different kinds of tools can be used. To capture this, the method engineer introduces tool kinds “Compiler”, “.NET Compiler” (a subtype of the former) and “Diagram Editor”.

7.1.63 WorkPerformance

A work performance is *an assignment and responsibility association between a particular producer and a particular work unit*.

WorkPerformance is an abstract subclass of EndeavourElement.

This is a producer- and process-related class.



7.1.63.1 Attributes

Name	Type	Semantics
Justification	String	The justification why the associated work unit is assigned to the associated producer.

7.1.63.2 Relationships

Name	Role	To class	Semantics
InvolvesWork-Unit	n/a	WorkUnit	A work performance involves a particular work unit.

Name	Role	To class	Semantics
Involves-Producer	n/a	Producer	A work performance involves a particular producer.

7.1.63.3 Example

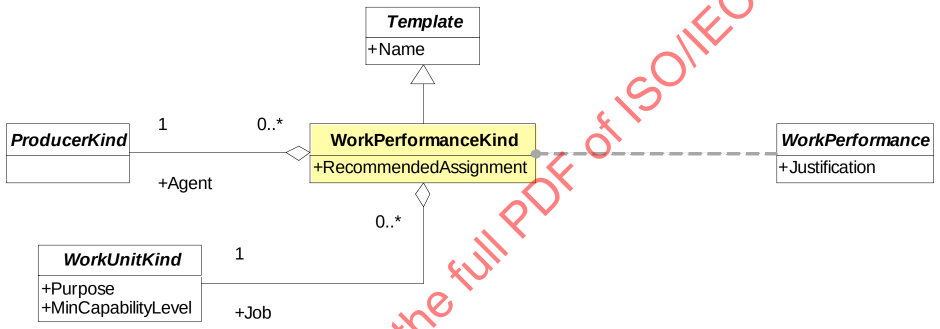
During a certain project, the system documentation team is assigned the task to create a user's manual for the system being developed. Such assignment is a work performance.

7.1.64 WorkPerformanceKind

A work performance kind is *a specific kind of work performance, characterized by the purpose of the inherent assignment and responsibility association.*

WorkPerformanceKind is a subclass of Template.

This is a producer- and process-related class.



7.1.64.1 Attributes

Name	Type	Semantics
Recommended-Assignment	DeonticValue	The recommended assignment of work units of the associated kind to producers of the associated kind. See 7.2.2 for possible values.

7.1.64.2 Relationships

Name	Role	To class	Semantics
InvolvesWork-UnitKind	n/a	WorkUnitKind	A work performance kind involves a particular work unit kind.
Involves-ProducerKind	n/a	ProducerKind	A work performance kind involves a particular producer kind.

7.1.64.3 Example

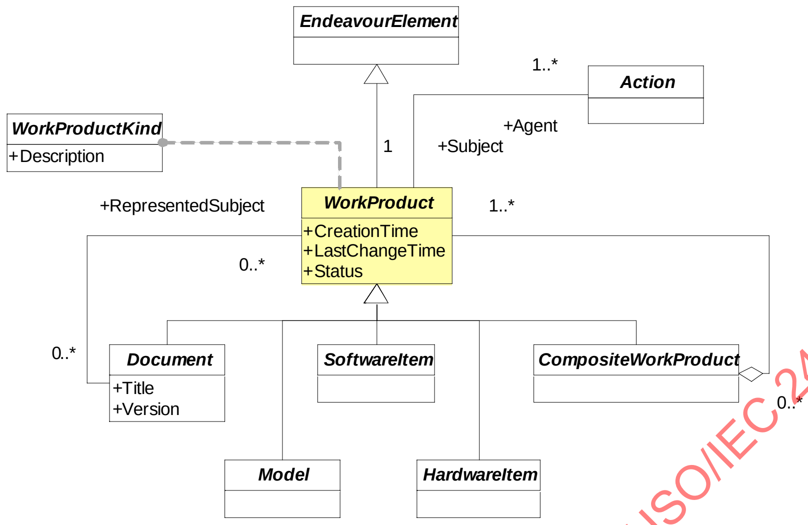
In a given methodology, the "Quality Assurance" process kind must be mapped to the "Quality Assurance Team" team kind and, to a lesser degree, to the "Development Team" team kind. In order to do this, the method engineer creates two work performance kinds associated with the "Quality Assurance" process kind (also a work unit kind): one of them is associated to the "Quality Assurance Team" producer kind and has RecommendedAssignment = Recommended; and the other one is associated to the "Development Team" producer kind and has RecommendedAssignment = Optional.

7.1.65 WorkProduct

A work product is *an artefact of interest for the endeavour*. Work products are usually documents, physical things or information collections that are created, modified or referred to (i.e. used but not changed) during the endeavour.

WorkProduct is an abstract subclass of EndeavourElement, specialized into Document, Model, SoftwareItem, HardwareItem and CompositeWorkProduct.

This is a product-related class.



7.1.65.1 Attributes

Name	Type	Semantics
CreationTime	Timestamp	The point in time at which the work product is created.
LastChangeTime	Timestamp	The point in time at which the work product is last changed.
Status	WorkProductStatus	The maturity status of the work product. See 7.2.3 for possible values.

7.1.65.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	WorkProductKind	A work product in the endeavour domain is always of some work product kind defined in the methodology domain.
n/a	Represented-Subject	Document	A work product may be depicted by different documents.
n/a	Subject	Action	A work product is always the subject of one or more actions.
IsPartOf	Component	CompositeWork-Product	A work product may be part of any number of composite work products.

7.1.65.3 Example

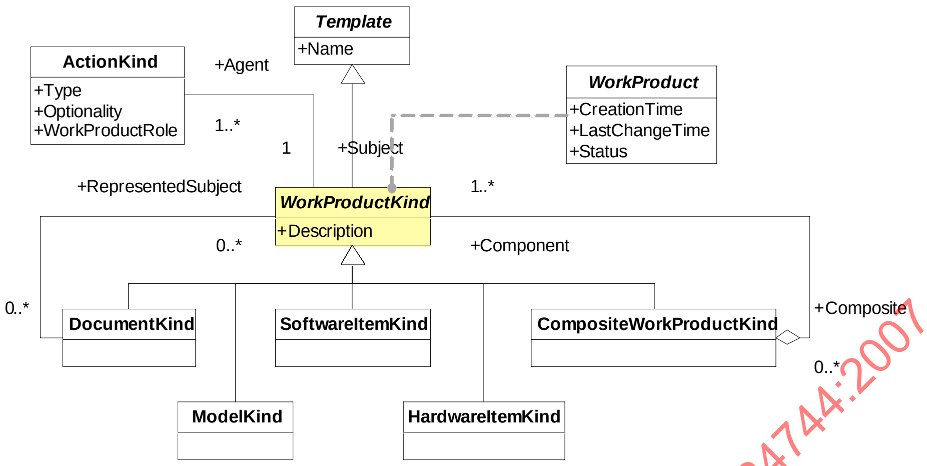
During a certain project, John creates a requirements specification using a needs statement provided by the customers. The requirements specification is then used by other developers as a starting point to design and build the final system. The users' needs statement, the requirements specification and the final system are work products.

7.1.66 WorkProductKind

A work product kind is *a specific kind of work product, characterized by the nature of its contents and the intention behind its usage.* Different work product kinds are usually defined to provide content and/or presentation "templates" that can be applied to the corresponding work products.

WorkProductKind is an abstract subclass of Template, specialized into DocumentKind, ModelKind, Software-ItemKind, HardwareItemKind and CompositeWorkProductKind.

This is a product-related class.



7.1.66.1 Attributes

Name	Type	Semantics
Description	string	The description of the nature of the contents and, optionally, form of representation, of work products of this kind.

7.1.66.2 Relationships

Name	Role	To class	Semantics
Classifies	n/a	WorkProduct	A work product in the endeavour domain is always of some work product kind defined in the methodology domain.
n/a	Represented-Subject	DocumentKind	A work product kind may be depicted by different document kinds.
n/a	Subject	ActionKind	A work product kind is always the subject of one or more action kinds.
IsPartOf	Component	CompositeWork-ProductKind	A work product kind may be part of any number of composite work product kinds.

7.1.66.3 Example

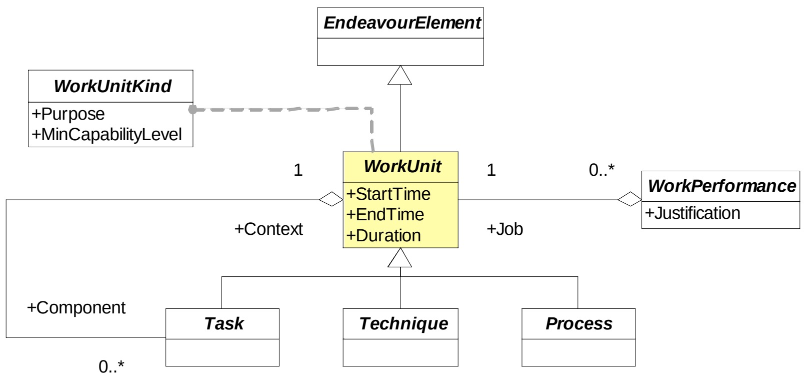
In a given methodology, the work product kind “System Requirements Specification” (a document kind) is defined to represent the fact that, when said methodology is enacted, work products of such kind will be created or used.

7.1.67 WorkUnit

A work unit is *a job performed, or intended to be performed, within an endeavour.*

WorkUnit is an abstract subclass of EndeavourElement, specialized into Task, Technique and Process.

This is a process-related class.



7.1.67.1 Attributes

Name	Type	Semantics
StartTime	Timestamp	The point in time at which the work unit is started.
EndTime	Timestamp	The point in time at which the work unit is finished.
Duration	Timespan	The span of time between the start time and the end time.

7.1.67.2 Relationships

Name	Role	To class	Semantics
IsClassifiedBy	n/a	WorkUnitKind	A work unit in the endeavour domain is always of some work unit kind defined in the methodology domain.
n/a	Context	Task	A work unit is always the context for a given task.
IsInvolvedIn-Performance	Job	Work-Performance	A work unit may be involved in a number of work performances.

7.1.67.3 Example

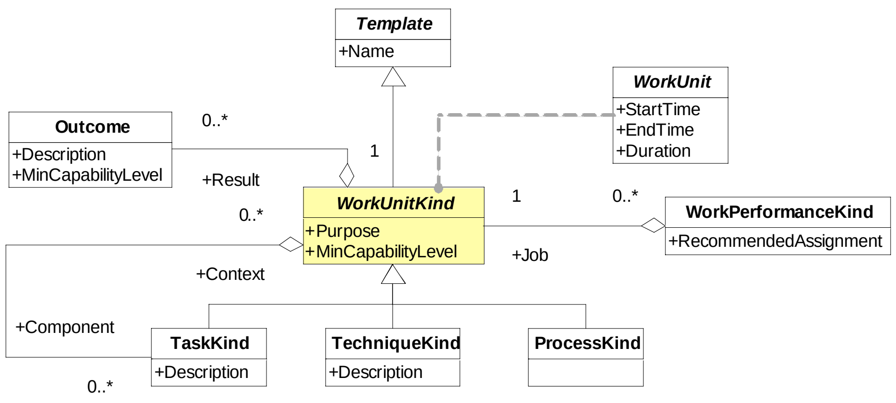
On a certain project, Mary is in charge of quality assurance. In order to do this, Mary's team reviews each generated product and then compares the number of defects found against historical data. Mary's overall responsibility (quality assurance) is a work unit (a process), and each individual piece of work performed by her team (reviewing work products and comparing defect reports to historical data) is also a work unit (tasks).

7.1.68 WorkUnitKind

A work unit kind is *a specific kind of work unit, characterized by its purpose within the endeavour.*

WorkUnitKind is an abstract subclass of Template, specialized into TaskKind, TechniqueKind and Process-Kind.

This is a process-related class.


Template
+Name
WorkUnit
+StartTime
+EndTime
+Duration
Outcome
+Description
+MinCapabilityLevel
0..*
1
+Result
0..*
WorkUnitKind
+Purpose
+MinCapabilityLevel
1
0..*
WorkPerformanceKind
+RecommendedAssignment
+Job
+Context
+Component
TaskKind
+Description
TechniqueKind
+Description
ProcessKind
0..*