INTERNATIONAL STANDARD

ISO/IEC 24744

First edition
2007-02-15

AMENDMENT 1
2010-02-01

Software Engineering — Metamodel for Development Methodologies

AMENDMENT 1: Notation

Ingénierie du logiciel — Métamodèle pour les méthodologies de développement

AMENDEMENT 1: Notation

Reference number
ISO/IEC 24744:2007/Amd.1:2010(E)

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.

Amendment 1 to ISO/IEC 24744:2007 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 7, *Software and systems engineering*.

Software Engineering — Metamodel for Development Methodologies

AMENDMENT 1: Notation

Page iii, Contents

Add the following line after "**Annex B** (informative) **Mappings to Other Metamodelling Approaches**":

Annex C (informative) **Graphical Notation**

Page iv, Table of Figures

After the last entry ("Figure 9 – Support classes") add the following:

Figure C.1 – A lifecycle diagram showing the temporal structure of a complete method

Figure C.2 – A lifecycle diagram showing the content structure as well as the temporal structure of a method

Figure C.3 – An enactment diagram for the "Construction" phase kind of Figure C.2

Figure C.4 – A dependency diagram based on a refinement of Figure C.2

Figure C.5 – A process diagram showing the details of the "Requirements Engineering" and "Requirements Quality Assurance" processes

Figure C.6 – An action diagram showing how requirements-related task kinds interact with requirements-related work products

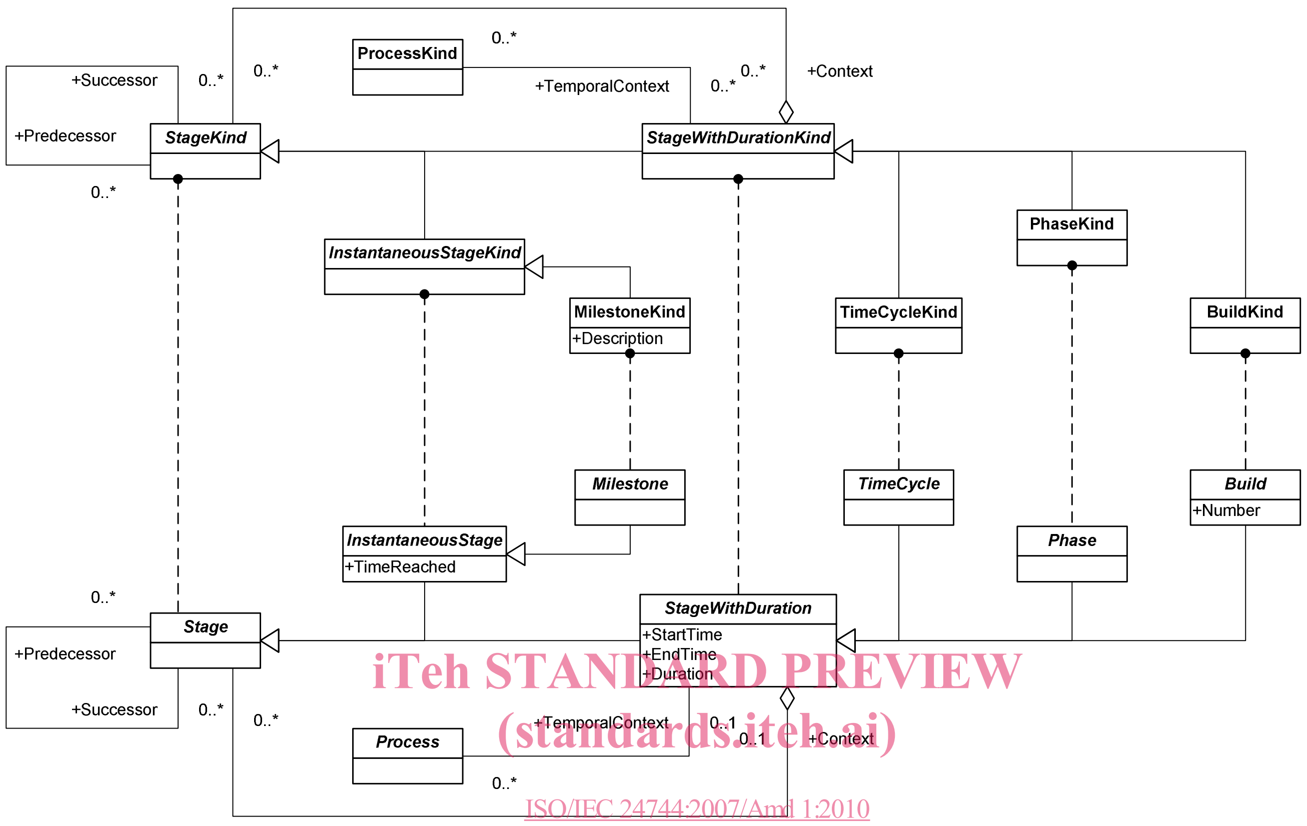
Page vi, Introduction

Add the following paragraph at the end of the Introduction:

This International Standard also presents a proposed notation for the ISO/IEC 24744 standard metamodel. The notation presented here is mainly graphical and supports most of the classes found in ISO/IEC 24744.

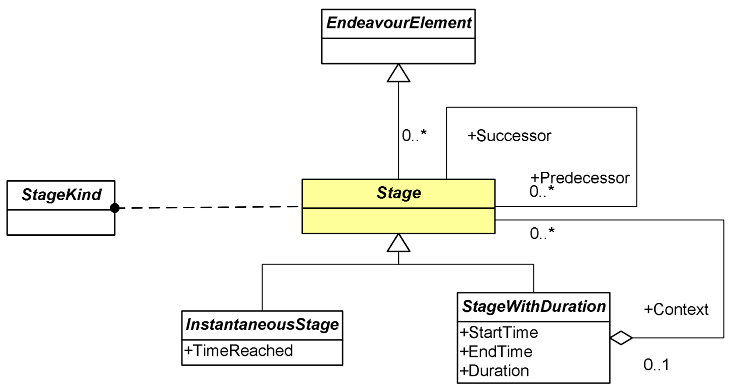
Page 11, Figure 5

Replace the figure with the following:



Page 46, 7.1.46

Replace the diagram with the following:



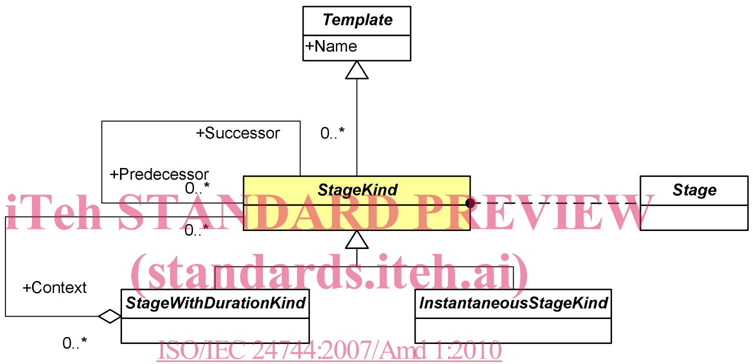
Page 47, 7.1.46.2

Add the following two new rows to the table:

Name	Role	To class	Semantics
OccursAfter	Successor	Stage	A successor stage occurs after some predecessor stages.
OccursBefore	Predecessor	Stage	A predecessor stage occurs before some successor stages.

Page 47, 7.1.47

Replace the diagram with the following:



Page 47, 7.1.47.2

Add the following two new rows to the table:

Name	Role	To class	Semantics
OccursAfter	Successor	StageKind	A successor stage kind occurs after some predecessor stage kinds.
OccursBefore	Predecessor	StageKind	A predecessor stage kind occurs before some successor stage kinds.

Page 78, before the Bibliography

Insert the following new annex:

Annex C
(informative)

Graphical Notation

C.1 Introduction

The metamodel of ISO/IEC 24744 contains classes that represent concepts from the method domain and classes that represent concepts from the endeavour domain. The notation presented here covers mainly the former, although some recommendations are given on how to represent the latter. Using this notation, methodologists or method engineers can represent method fragments and complete methodologies, and project managers can depict endeavours as they progress over time.

This notation has been designed to be easy to draw by hand as well as using a software tool on a computer. Special care has been taken in choosing symbols that convey the underlying concept, at least in most situations and to readers of most cultures and backgrounds. In addition, the symbols adopted by the notation exhibit visual resemblance (based on shapes and colours) to each other that mimic the structural relationships of the underlying concepts in the metamodel, establishing common "visual themes" for closely related concepts. Although colour is extensively used by this notation, since it helps identify symbols and symbol patterns with ease when displayed on a computer display or a colour printout, it is important to note that care has been taken to guarantee that greyscale and black and white versions of the same symbols are perfectly readable and identifiable. In this regard, colour does enhance diagram readability when it is available but, conversely, it can be avoided without too great a loss. Colour specification is done via RGB values in the sRGB (IEC 61966-2-1:1999) colour space.

C.1.1 Abstract Symbols

This notation introduces the concept of "abstract symbols", i.e. symbols that depict instances of abstract classes. In principle, most notations only include symbols to depict instances of concrete classes, since abstract classes do not have direct instances. However, in some scenarios it is convenient to represent an entity in a diagram for which only the abstract type is known. This can be achieved by using so-called abstract symbols. For example, consider the case where a work product kind representing a certain system must be depicted in a diagram. A notation with only concrete symbols would force the diagram author to choose a specific concrete type of work product kind (such as document kind, model kind, software item kind etc.) in order to depict it. This notation includes an "abstract work product kind" symbol that allows the author to depict the above-mentioned system without specifying whether it is a model kind, a document kind, a software item kind etc. Abstract symbols usually consist of the simple shape from which all the concrete symbols in the visual theme are generated.

C.1.2 Notation Coverage

As a general principle, graphic symbols are given in the notation for every concrete (i.e. directly instantiable) class in the metamodel for which a graphical representation is considered to be appropriate. In addition, additional graphic symbols (abstract symbols) are given for every abstract class that is a direct ancestor (i.e. superclass) of said concrete classes. Abstract classes or higher ranks are considered to be too intangible as to be worth representing visually. For example, the DocumentKind class is concrete and suitable for visual representation, so a symbol for it is given in the notation; the WorkProductKind is abstract but a direct ancestor of DocumentKind, so an abstract symbol is given to it; on the contrary, the Template class, superclass of WorkProductKind, is too intangible and no graphical representation is given for it.

Specifically, the proposed notation for ISO/IEC 24744 covers the following classes:

— Stage-related classes, i.e. TimeCycleKind, PhaseKind, BuildKind and MilestoneKind as well as their direct ancestors StageWithDurationKind and InstantaneousStageKind.

— Work unit-related classes, i.e. ProcessKind, TaskKind and TechniqueKind as well as their direct ancestor WorkUnitKind. The related class Outcome is also covered.

— Work product-related classes, i.e. DocumentKind, ModelKind, SoftwareItemKind, HardwareItemKind and CompositeWorkProductKind, as well as their direct ancestor WorkProductKind.

— Producer-related classes, i.e. TeamKind, RoleKind and ToolKind, as well as their direct ancestor ProducerKind. The endeavour-level-only class Person is also covered.

— Constraint classes, i.e. PreCondition and PostCondition.

— “Relationship” classes, i.e. ActionKind, TaskTechniqueMappingKind and WorkPerformanceKind.

— Some support classes, i.e. Conglomerate and Guideline.

Wherever possible, these areas determine a set of “families” of symbols, which roughly correspond to branches in the specialization hierarchy in the metamodel. Within each such family, the shapes and colours of the icons for the subtypes are similar. In addition, notation is given for generic concepts, usually corresponding to relationships and links that may occur in a variety of situations.

The above bulleted list includes mostly method-domain classes. The corresponding endeavour-domain classes are also addressed by this notation. For the sake of coherence, the convention is adopted that an endeavour-domain class is always represented by the same symbol as used for its method-domain counterpart but using a dashed line. For example, the symbol for the MilestoneKind class is a small rotated blue square; this means that the symbol for the Milestone class is a similarly small rotated blue square in a dashed line. By using this convention, a type (method-domain) and its instances (endeavour-domain) are always represented by closely enough symbols but with a clear difference that makes them distinguishable. This convention is assumed throughout the remaining sections of this annex, and therefore explicit notation for endeavour-domain classes is not given.

The metamodel areas not covered by this notation include:

— Language-related classes, i.e. Language, Notation and ModelUnitKind. These classes work together to represent formal languages (plus their visual representations), which are better expressed by class diagrams, EBNF or other formalisms. A graphical representation is therefore considered not appropriate.

— Reference-related classes, i.e. Reference and Source. These classes are designed to allow method engineers to provide reference material about the “method chunks” (Section 5.1) that they create. Structured text is a better representation of this kind of information than graphical diagrams.

The following sections describe the notation in detail, including the symbols used, the syntax for their usage (derived from the metamodel) and the associated semantics (given by the mapping between the graphical symbols and the classes in the metamodel).

C.2 Notation Elements

C.2.1 Stages

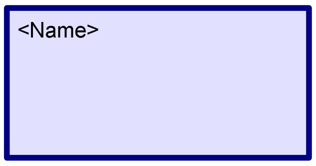
A stage is a managed time frame within an endeavour (Section 7.1.46). Stages are partitioned into stage kinds by the StageKind class according to the abstraction level at which they work on the endeavour and the result that they aim to produce (Section 7.1.47).

Four concrete subtypes of StageKind are covered by this notation: TimeCycleKind, PhaseKind, BuildKind and MilestoneKind. The former three correspond to stages with duration and are, therefore, represented by broad symbols that can contain other elements. A rectilinear theme has been chosen to convey the idea of temporality. MilestoneKind, on the other hand, corresponds to instantaneous stages, and is consequently depicted by a narrower symbol that cannot contain nested elements. Colours for all these symbols belong to the blue-purple range. The name of the stage kind is shown inside the symbol.

C.2.1.1 StageWithDurationKind

A stage with duration is a managed interval of time within an endeavour (Section 7.1.48). Stages with duration are partitioned into stage with duration kinds by the StageWithDurationKind class according to its abstraction level and the result it aims to produce (Section 7.1.49).

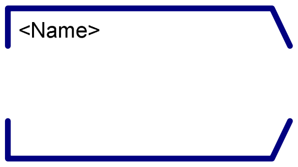
This is an abstract class, depicted by an abstract symbol, a horizontally oriented rectangle. This symbol tries to convey the idea of an empty container, inside which other elements can be shown. Line colour is navy blue (RGB 0, 0, 128) and fill colour is light blue-grey (RGB 225, 225, 255). The name of the stage with duration kind is shown inside the rectangle, in the top left corner, where the “<Name>” placeholder appears in the following figure.



C.2.1.2 TimeCycleKind

A time cycle is a managed interval of time within an endeavour for which the objective is the delivery of a final product or service (Section 7.1.59). Time cycles are partitioned into time cycle kinds by the TimeCycleKind class according to the type of outcomes that they aim to produce (Section 7.1.60).

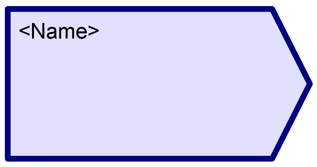
The symbol used to depict a time cycle kind is composed of two horizontal brackets with their right-hand side end bent outwards, simulating a truncated arrow head. These brackets delimit a rectangle within which symbols for other stage kinds can be shown. This symbol tries to convey the meaning that a time cycle kind comprises a collection of other stage kinds, hence the bracket analogy. Line colour is navy blue (RGB 0, 0, 128). The name of the time cycle kind is shown inside the symbol, in the top left corner, where the “<Name>” placeholder appears in the following figure.



C.2.1.3 PhaseKind

A phase is a managed interval of time within an endeavour for which the objective is the transition between cognitive frameworks (Section 7.1.31). Phases are partitioned into phase kinds by the PhaseKind class according to the abstraction level and formality of the result that they aim to produce (Section 7.1.32).

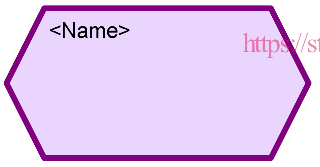
The symbol used to depict a phase kind is a pointed rectangle with the point facing to the right. Line colour is navy blue (RGB 0, 0, 128) and fill colour is light blue-grey (RGB 225, 225, 255). This symbol tries to convey the idea of temporality – hence the point, reminiscent of an arrow head. The name of the phase kind is shown inside the symbol, in the top left corner, where the “<Name>” placeholder appears in the following figure.



C.2.1.4 BuildKind

A build is a managed interval of time within an endeavour for which the major objective is the delivery of an incremented version of an already existing set of work products (Section 7.1.3). Builds are partitioned into build kinds by the BuildKind class according to the type of result that they aim to produce (Section 7.1.4).

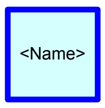
The symbol used to depict a build kind is a double-pointed rectangle with the points facing to the right and left. This symbol tries to convey the idea of sequence, hence the double point, resembling dual arrow heads. Line colour is lilac (RGB 128, 0, 128) and fill colour is light purple (RGB 234, 213, 255). The name of the build kind is shown inside the symbol, on the top left corner, where the “<Name>” placeholder appears in the following figure.



C.2.1.5 InstantaneousStageKind

An instantaneous stage is a managed point in time within an endeavour (Section 7.1.16). Instantaneous stages are partitioned into instantaneous stage kinds by the InstantaneousStageKind class according to the kind of event that it signifies (Section 7.1.17).

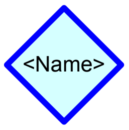
This is an abstract class, depicted by an abstract symbol, a square. This symbol tries to convey the idea of a point in time, hence the similarity with other stage-related symbols (overall rectangular shape) but a smaller area. Since instantaneous stages are points in time rather than time spans, no other symbols can be shown inside this one. Line colour is bright blue (RGB 0, 0, 255) and fill colour is light blue (RGB 213, 213, 255). The name of the instantaneous stage kind is shown inside the square, centred, where the “<Name>” placeholder appears in the following figure.



C.2.1.6 MilestoneKind

A milestone is a managed point in time within an endeavour that marks some significant event in the endeavour (Section 7.1.20). Milestones are partitioned into milestone kinds by the MilestoneKind class according to their specific purpose and kind of event that they signify (Section 7.1.21).

The symbol used to depict a milestone kind is a small square rotated 45 degrees, resembling a diamond shape. This symbol tries to convey the idea of an event-marking point in time, hence the similarity with other stage-related symbols (points facing left and right) but a smaller area. It also resembles the symbol used by many project management software tools to depict milestones. Line colour is bright blue (RGB 0, 0, 255) and fill colour is light blue (RGB 213, 213, 255). The name of the milestone kind is shown inside the square, centred, where the "<Name>" placeholder appears in the following figure.



C.2.2 Work Units

A work unit is a job performed, or intended to be performed, within an endeavour (Section 7.1.67). Work units are partitioned into work unit kinds by the WorkUnitKind class according to their purpose within the endeavour (Section 7.1.68).

Three concrete subtypes of WorkUnitKind are covered by this notation: ProcessKind, TaskKind and TechniqueKind. None of these concepts are involved in whole/part relationships that may need nesting of symbols, so the symbols chosen to depict them are basic shapes and easily resizable to accommodate long names or abbreviations. All shapes are curvilinear. Colours for all these symbols belong to the green range. The name of the work unit kind is shown inside the symbol, centred. The minimum capability level (Section 7.1.68.1) of the work unit kind can be optionally shown inscribed in a small circle, inside the symbol.

In addition, the Outcome class is also covered.

No abstract symbol is provided for WorkUnitKind because no abstract usage of work unit kinds is expected.

C.2.2.1 ProcessKind

A process is a large-grained work unit that operates within a given area of expertise within the endeavour (Section 7.1.35). Processes are partitioned into process kinds by the ProcessKind class according to the area of expertise in which they occur (Section 7.1.36).

The symbol used to depict a process kind is a rounded rectangle or "roundangle". Line colour is olive green (RGB 102, 153, 0) and fill colour is light olive green (RGB 234, 255, 213). The name of the process kind is shown inside the symbol, centred, where the "<Name>" placeholder appears in the following figure. The minimum capability level of the process kind is optionally shown in the top left corner inside the rectangle ("n" in the figure).