Journal of Visual Languages and Computing 23 (2012) 195–212

Contents lists available at SciVerse ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

Evaluating a graphical notation for modelling software development methodologies ☆

Kenia Sousa [a,*], Jean Vanderdonckt [a], Brian Henderson-Sellers [b], Cesar Gonzalez-Perez [c]

[a] Louvain School of Management (LSM), Information Systems Unit (ISYS), Université catholique de Louvain, Louvain-La-Neuve, Belgium
[b] School of Software, Faculty of Engineering and Information Technology, University of Technology, Sydney, Australia
[c] Institute of Heritage Sciences (Incipit), Spanish National Research Council (CSIC), Spain

ARTICLE INFO

Article history:
Received 10 May 2009
Received in revised form
8 March 2012
Accepted 2 April 2012
Available online 26 April 2012

Keywords:
Software development methodologies
Method engineering
Graphical notation
Cognitive dimensions

ABSTRACT

This work aims at evaluating a graphical notation for modelling software (and other kinds of) development methodologies, thus demonstrating how useful the graphical aspects can be for sharing knowledge between the people responsible for documenting information and those responsible for understanding and putting it into practice. We acknowledge the importance of having a common set of symbols that can be used to create, use and disseminate information for a larger audience than is possible today with a variety of alternatives and lack of a common ground. Using a cognitive dimensions framework, we make a standard evaluation of the elements and diagrams of the notation proposed to support the ISO/IEC 24744 methodology metamodel standard, considering the trade-offs between different dimensions. We suggest improvements to this existing notation based on this analysis, in the context of improving communication between creators and users of methodologies.

1. Introduction

Organizations that want to follow a standardised way of working find it suitable to adopt an agreed approach that addresses important organizational issues, such as defining organizational roles, responsibilities and their communication needs. Along with the growing importance of creating and customizing methods according to the organizational context comes the idea of actually modelling the method. The use of metamodels to construct methods, as discussed in [11,17], has demonstrated its usefulness for tool support [33]; standardising knowledge; representing information in an abstract manner [4]; among other advantages. However, as well as using a standardised metamodel, the elements of the method that are conformant to this metamodel need to be visualized in an agreed and standardised way—such a notation is the focus of this paper. The notation analysed here is applicable to a wide variety of methodological domains, not only software but also hardware, systems and numerous engineering application domains. However, since the underpinning metamodel was standardised within the software engineering community [21], we will adopt this as our domain of discourse whilst not forgetting its wider potential applicability.

Both metamodels and models are essentially cognitive constructs i.e. they exist in an individual's mind. To share these concepts successfully, a means of communicating these ideas is necessary. A typical way, and one exploited here, is the creation of diagrams using a predominantly graphical notation. In this context, software engineering *metamodels* are often depicted using Object-Oriented (OO) notational elements such as UML [28] class diagrams. For example, the Software & System Process Engineering

☆ This paper has been recommended for acceptance by Shi Kho Chang.

* Corresponding author. Tel.: +32 010/47.83.91.
E-mail addresses: kenia.sousa@gmail.com, kenia.sousa@usi4biz.com (K. Sousa).
URL: http://usi4biz.com/kenia (K. Sousa).

1045-926X/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jvlc.2012.04.001

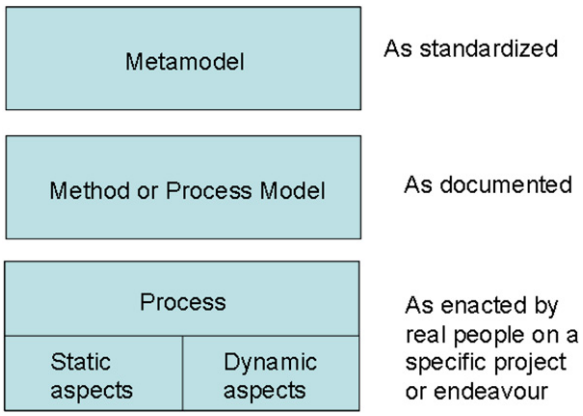

Fig. 1. Terminology of the three domains of interest.

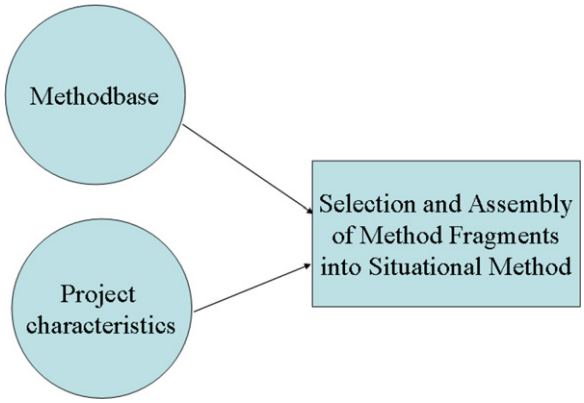

Fig. 2. A situational method is created from elements in the methodbase by the application of characteristics from a specific project or endeavour.

Metamodel (SPEM), a metamodel for defining software development processes and their components [31] uses UML diagrams with extensions using stereotypes. The metamodel of the Object-Oriented Process, Environment and Notation (OPEN) Process Framework (OPF) [7], a process environment to create and configure processes according to project needs, is also presented by class diagrams to represent tasks, activities and their relationships within a lifecycle. In addition, the semantics of these metamodel elements may be given as additional free text (natural language and/or a formal language such as OCL). Of greater interest here is the graphical representation of the process models or methods generated from and concordant with these metamodels. In particular, we investigate the current proposals for a (new) graphical representation of method elements from the ISO/IEC 24744 [21] International Standard, a preliminary version of which is found in [11], the final version being published in 2010 [22].[1]

While the *process* of constructing a method may be aided by graphics such as the MAP approach [30], deontic matrices [12] or a UML activity diagram [32], here we concentrate on the *product* of that construction process i.e. the method (a.k.a. methodology or process model—middle element of Fig. 1) and, to a lesser extent, graphically representing the predominantly static aspect of the enactment of such a methodology (called here a "process") on a specific endeavour (Fig. 1 lower element). The process is created as an instance of the process model, the elements of which are conformant to elements of the metamodel. Concurrently, we also focus on the method construction as exemplified in Situational Method Engineering (SME) [35]—as supported in [21]. SME focuses on the use of method fragments as elementary or atomic pieces of a method, gleaned from existing theory and practice in many ways [29]. Stored in a methodbase or repository, these fragments are then selected and "glued" together by the addition of project-specific characteristics to create a process model highly applicable to a specific software development situation, thus creating a situational method (Fig. 2). The instantiation of such a metamodel into a model representing a method requires a graphical representation that facilitates understanding the goals to be achieved and strategies to be used when applying the method.

Various methods have their particular visual aspect to depict elements and relationships between them, such as a list of activities played by specific roles. Some approaches adopt a formal[2] notation to specify methods, such as UML activity diagrams, while others adopt a strategy that applies simple connectors of activities aimed at allowing anyone to understand the method, such as the Usability Engineering Life cycle [23] and the Usability Design Process [8]. The existence of different approaches to specify methods makes a common understanding more difficult. A shared notation composed of graphical elements to represent methods aims at facilitating the communication between method engineers from different domains (e.g. SE, HCI) or approaches (e.g. traditional, agile). Besides the difficulty of communicating ideas, with a wide variety of ways to design methods, method engineers also face the burden of constantly learning the graphical elements necessary to design a method or tailor existing ones, depending on what the organization uses. A common notation facilitates their work because it improves the quality of discussion since they use a shared set of elements, thus decreasing the learning curve.

Usually methods involve different roles responsible for executing certain activities and producing work products. Furthermore, to discover any strategic information regarding how a method works, it is easier to reason about the method by analysing the models that describe it, as supported in [9]. For instance, it is more efficient to learn the method by reading its models than by observing people working, making questions about certain issues, taking notes; as well as a set of other actions that could be

[1] The ISO/IEC 24744 notation was formally published during the review process for this paper.

[2] Formal can have many meanings. Here we combine the meaning of "having a form" (e.g. [15]) and "having a mathematical/logical basis" (here a metamodel).

repeated each time a new professional comes to an organization.

When method engineers use diagrams to design methods, their main goal is to express knowledge of the method and share it with the IT professionals within an organization. Therefore, the graphical elements and the diagrams that are created need to be easy to be drawn by method engineers, as well as being easy to be understood by and useful for the professionals.

Our main goal, therefore, is to evaluate a formal graphical notation for modelling software development methodologies with an emphasis on its use by method engineers, who are designers of methods and also by IT professionals, who must understand the modelled methods. The notation under study provides a structured approach to describe software development methodologies through models independent of technology, which enlarges its added value for Software Engineering as well as for other disciplines, such as Human–Computer Interaction.

The cognitive dimensions framework [3] to be used here to evaluate the notation proposed for ISO/IEC 24744 supports the analysis of the interactions among (1) the structure of the information; (2) the properties and resources of an environment where the structure can be manipulated and (3) the type of activity the user wants to perform with the notation. The cognitive dimensions framework is further explained in Section 3. Applying this framework to our main goal, the structure of the information (item 1 above) is represented by a graphical notation. This notation can be manipulated through paper and pencil, but there is also the possibility of using a CASE tool or a Microsoft Visio stencil (the latter is used as an example here) that may be used to create diagrams using the notation. However, this aspect is optional since our emphasis is on the actual notation, not on a tool that supports handling it.[3] Clearly, the characteristics of a notation may be implemented differently in different software tools. Since such tools are still in the commercial development stage (by several companies worldwide), in this paper we provide a baseline in terms of our evaluation of the ISO/IEC 24744 notation as *drawn by hand*. This analysis, as presented here, can, in the future, be used for comparison purposes when a similar cognitive dimensions analysis is performed on any one specific software drawing tool.

The cognitive dimensions framework is utilized to evaluate a graphical notation used to model methods and its elements, and to read the created methods. The framework is applied by first understanding the scope of the notation under analysis (e.g. graphical diagrams and elements of a notation) and the tasks under analysis (e.g. exploratory design to create a method and its elements using the notation). The second step is to evaluate the notation considering one dimension at a time. One of the main advantages of this framework is that it brings a set of common terms that can be useful to judge different aspects of the notation. For each dimension, it is important to understand the meaning of the dimension, its relevance to the tasks under analysis and the trade-off of this dimension concerning the other dimensions. Once the dimension is understood, the analysis of the notation for this specific dimension includes detecting the positive and negative factors of the notation. With this analysis, it is possible to understand how well the notation addresses each dimension in order to suggest improvements for the notation in the light of each dimension, when applicable.

The remainder of this work presents the main elements and diagrams of the graphical notation under analysis (Section 2). The cognitive dimensions framework is then presented (Section 3) and used (Section 4) to evaluate the proposed ISO/IEC 24744 notation when used to design methods. A complexity analysis in Section 5 completes the analysis. The main contributions are demonstrated through the proposed solutions for certain issues, leading to some recommended improvements to the standard notation for ISO/IEC 24744, made prior to its publication.

2. Graphical notation of ISO/IEC 24744

ISO/IEC 24744 [21] describes the Software Engineering Metamodel for Development Methodologies (SEMDM). Built on a three-domain architecture (Fig. 3), corresponding to the concepts shown in Fig. 1, it uses powertype patterns [19] within the standardised metamodel. Fig. 4 shows the key concepts, represented as classes, of the SEMDM, all of which reside in the metamodel domain of Fig. 3. Both the metamodel—and its corresponding notation (Fig. 5)—use concepts that can be grouped into the following areas:

Stages (including StageWithDurationKind, TimeCycleKind, PhaseKind, BuildKind, InstantaneousStageKind, MilestoneKind) represent calendar time. Some of the stage elements can contain other elements; therefore, the symbols need to be wide enough to contain elements and also need to be expandable.

Work units (including ProcessKind, TaskKind, TechniqueKind, Outcome) represent jobs to be done. These elements do not contain other elements, so the symbols have basic shapes, although still resizable to accommodate long names.

Work products (including DocumentKind, ModelKind, SoftwareItemKind, HardwareItemKind and CompositeWorkProductKind) are artefacts of interest to the endeavour. These elements are represented with simple symbols since they do not contain other elements.

Producers (including ProducerKind, TeamKind, RoleKind, ToolKind, Person) represent people and tools that execute the work units and utilize the work products. These elements thus represent agents with symbols based on a schematic depiction of a torso.

Relationship concepts (including ActionKind, WorkPerformanceKind) show connections between elements, represented by arcs between pairs of symbols. ActionKind and WorkPerformanceKind use a deontic marker, which is a representation of association between elements that indicates optionality. ActionKind additionally uses

[3] Although for more general interest we do have repeated reference to the use of tools, using Visio as an exemplar only.

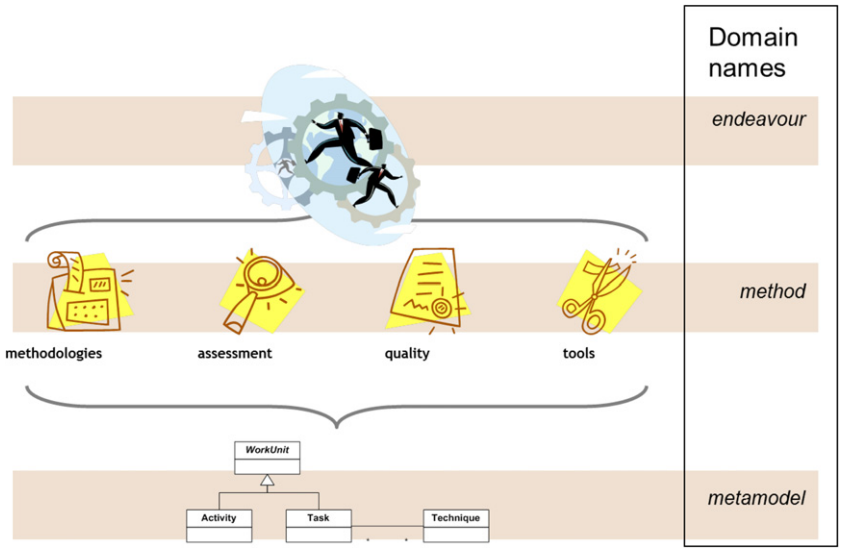

Fig. 3. Three domain architecture employed in ISO/IEC 24744. The endeavour domain is where people work on a project or a set of projects (called endeavours); the method domain contains information on how to develop software; this is all underpinned by a semi-formal metamodel (modified slightly from [18]).

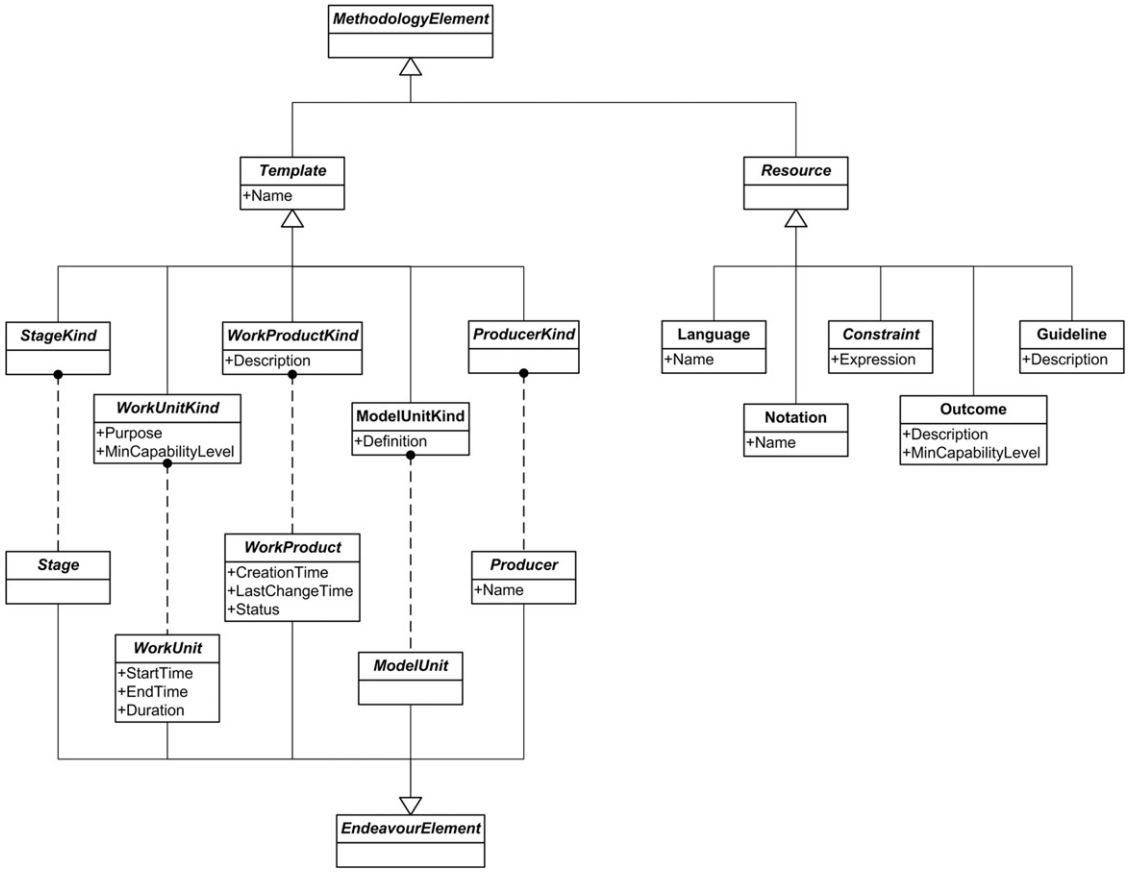

Fig. 4. SEMDM key concepts of the metamodel domain.

an ActionType, which indicates the type of usage of the task on the work product (i.e. ActionType t={C, M, R, D}, where C=create, M=modify, R=read, D=delete).

Constraints (including Constraint, PreCondition, PostCondition). The elements for constraints represent conditions that must hold at a certain point in time and are represented

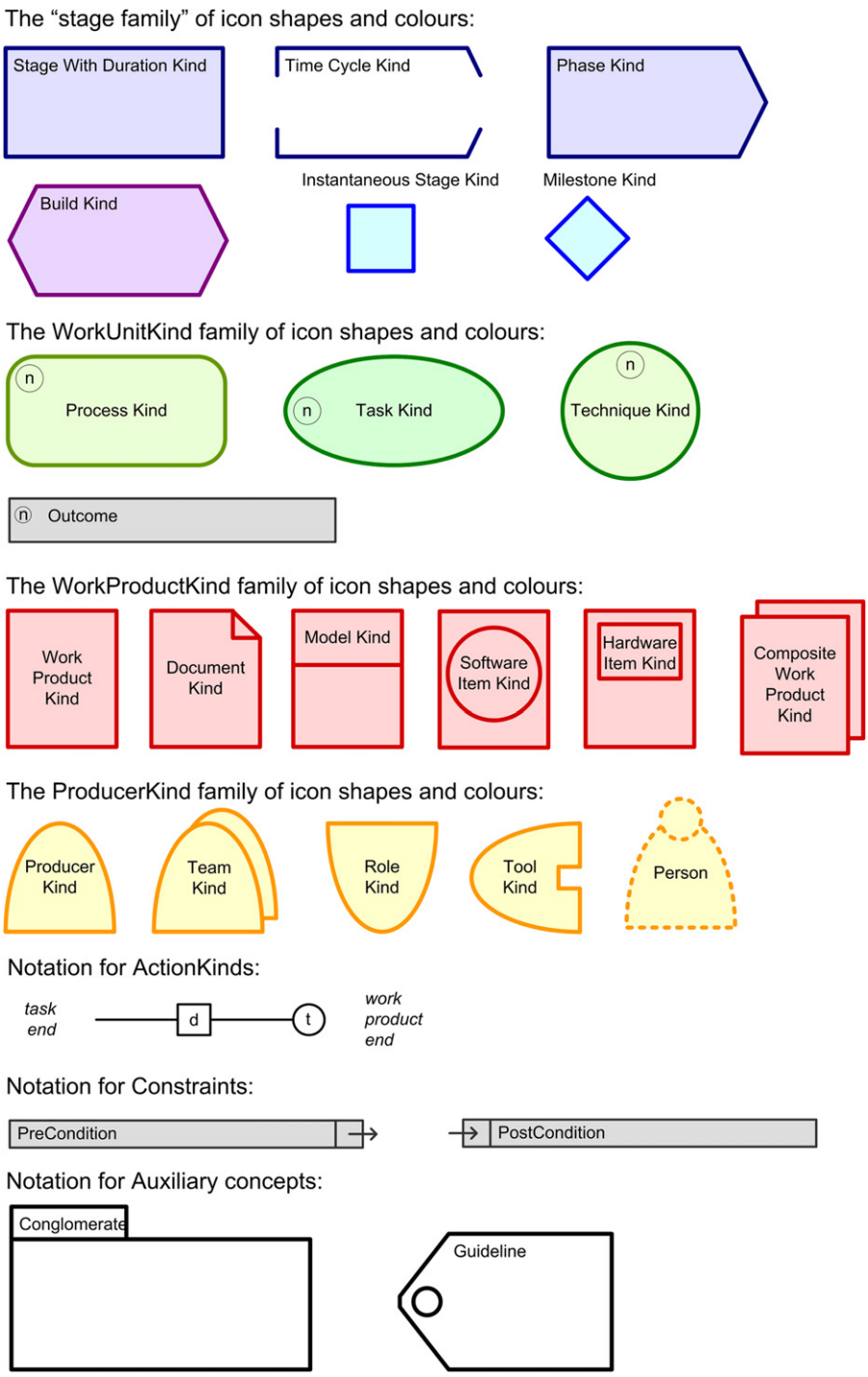

Fig. 5. Icon shapes and colours for all the graphical elements of ISO/IEC 24744. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

by a horizontal rectangle with two compartments in order to depict the type of constraint and an expression.[4]

Auxiliary concepts (Conglomerate and Guideline). A conglomerate is a collection of related methodology elements belonging to the method domain. The symbol is of a container nature. A guideline gives advice on how a set of methodology elements can be used during enactment. The symbol is representative of a luggage tag.

[4] It could be argued, from Tufte's [34] principle of ink economy that there are two visual cues here, when one may be considered sufficient. However, Moody [25] argues the reverse, citing the visual expressiveness of cartographic maps.

To use ISO/IEC 24744, method engineers create instances of the SEMDM concepts, these instances being part of the method domain (Fig. 3) known as method fragments. A full software development method is then constructed by relating these method fragments in such a way as to suit a particular organizational or project context or situation. This project-specific method can then be exercised on calendar-time projects of the endeavour domain (Fig. 3).

These graphical shapes related to the SEMDM concepts of Fig. 4 are collected together in Fig. 5. Each shape is depicted in a particular colour with a darker-coloured outline. The shapes have the colour as a visual cue. Although the notation is fully usable in black and white, here we also evaluate the proposed colour combinations for their usability and overall appropriateness.

It is interesting to note that although most of the shapes represent instances of concrete classes in the metamodel, some refer to abstract classes therein. Although one cannot have direct instances of an abstract metaclass, developers often wish to avoid committing to one particular (concrete) subtype. For this reason, oft-used abstract classes are also allocated their own symbol. For example, the developer may wish to use a StageWithDurationKind but not yet be sure whether this needs to be a TimeCycleKind, a PhaseKind or a BuildKind (three subclasses of StageWithDurationKind).

The various elements shown in Fig. 5 can be used in different diagrams. They are selected based on their usefulness to represent information about the method depending on the diagram, which takes a different perspective of the method. The types of diagrams proposed are the following:

Lifecycle diagram (Fig. 6) represents the overall structure of a method: method-level specification with stage elements and work unit elements.

Process diagram (Fig. 7) describes the details of a process kind or collection of process kinds used in a method. These details may include the relationships between process kinds, the links between process kinds and task kinds, and the producer kinds assigned to the process and task kinds by the appropriate work performance kinds.

Action diagram (Fig. 8) shows the interactions between task kinds and work product kinds.

Enactment diagram (Fig. 9) represents a specific enactment of a method (or part of a method) and its relationship to the method specification. An enactment diagram is composed of: (i) lifecycle diagram elements on the left-hand side, but organised in a vertical fashion; and (ii) a Gantt chart in which the bars represent instances of the elements in the lifecycle diagram, with time flowing from left to right as in a Gantt chart.

The shape and colour of the elements and their placements in the diagrams and the relationships between different elements are analysed in order to ensure that the notation optimally presents information about the method that is easily understood. The approach used to evaluate the notational elements and diagrams introduced above is explained in the next section.

3. Cognitive dimensions

The Cognitive Dimensions framework [3] is an approach to aid in the evaluation of the usability of information artefacts, which comprise two classes: interactive and

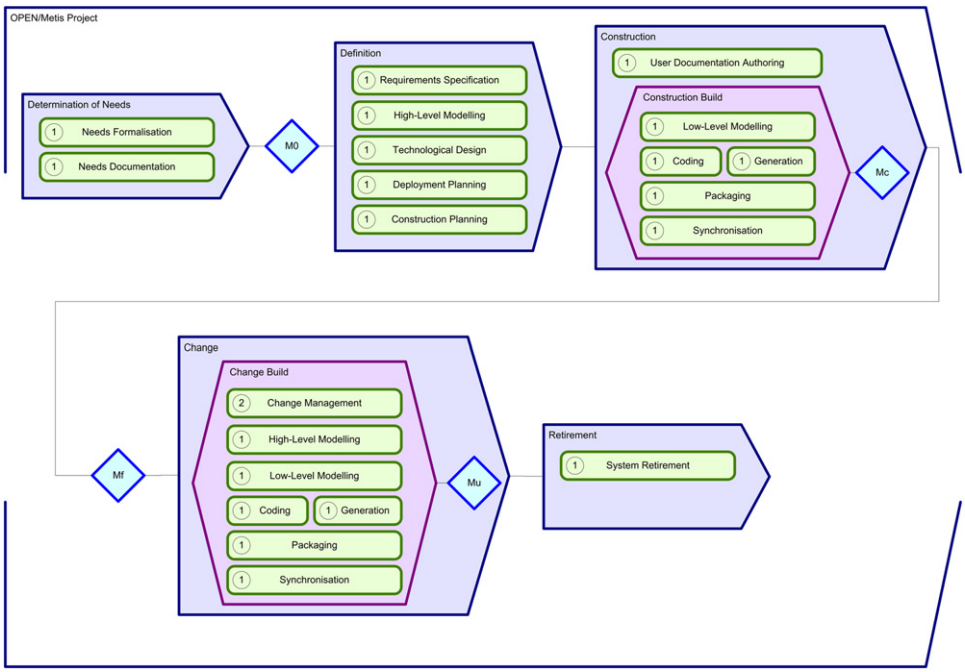

Fig. 6. A lifecycle diagram showing the content structure as well as the temporal structure of a method. Notice how stage kinds can contain both nested stage kinds (such as "Construction Build" inside "Construction") and process kinds ("User Documentation Authoring") (after [20], with kind permission of Springer Science+Business Media).

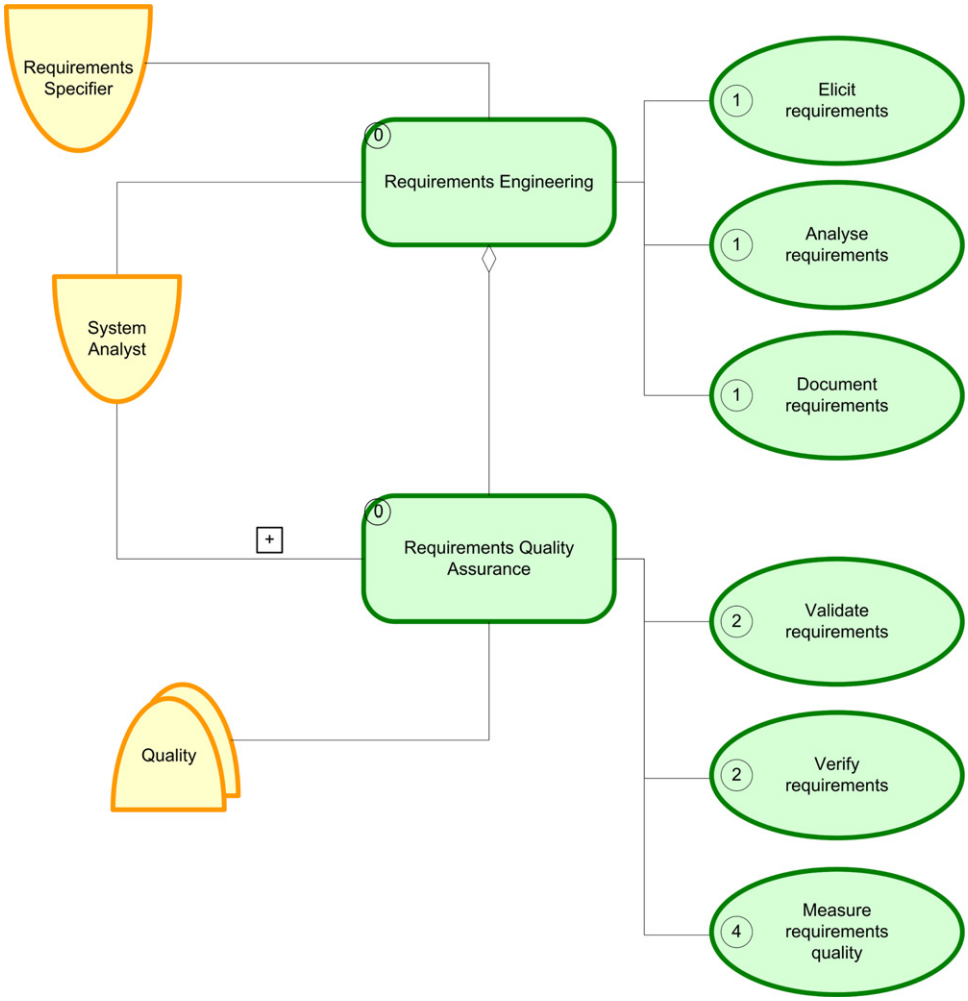

Fig. 7. A process diagram showing the details of the "Requirements Engineering" and "Requirements Quality Assurance" processes (after [20], with kind permission of Springer Science+Business Media).

non-interactive artefacts. Interactive artefacts are tools used to store, manipulate and display information, such as word-processors, mobile phones etc. Non-interactive artefacts are static information artefacts, such as tables, graphs, programming languages etc. In this work, we focus on the evaluation of a non-interactive artefact: a graphical notation, its elements and diagrams.

The Cognitive Dimensions framework gives names to concepts used as discussion tools when evaluating information artefacts. These concepts make it easier for users of this framework to learn it and apply it by communicating with terms that have been pre-defined and agreed upon. In addition, it enables doing trade-off analysis using concrete and familiar concepts. The application of this framework delivers an evaluation of information artefacts along different dimensions. The result of the analysis indicates how the dimensions are addressed by the artefact under analysis.

The Cognitive Dimensions framework identifies six different types of activities in which notations can be used: incrementation, transcription, modification, exploratory design, searching and exploratory understanding. The graphical notation is used for exploratory design, in which the final product needs to be discovered, through sketching or prototyping, for instance. Therefore, during the evaluation, we point out what is the cognitive relevance of each cognitive dimension for exploratory design, ranging from harmful, acceptable, important, useful.

The dimensions apply to the artefact as a whole, which encompasses the notation that is expressed by the perceived symbols that are combined to build an information structure: its elements and diagrams. We evaluate the notation and the act of drawing its symbols using paper and pencil and/or a drawing tool, but our main focus is on the notation's expressiveness and on actions of method engineers when using the notation to create methods.

We point out that whether drawing manually using paper and pencil or with a drawing tool, there are constraints to be respected in both contexts, although a small number of constraints are only applicable to either paper and pencil or to a tool. The reader's attention is drawn to such specificity when it occurs. Manually, there

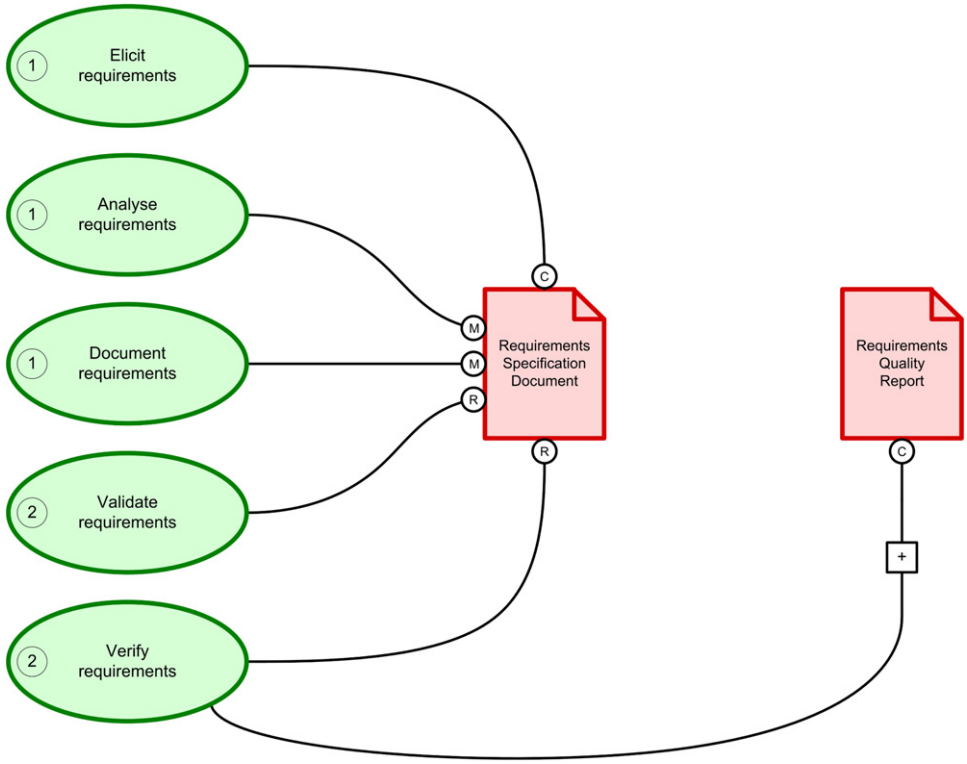

Fig. 8. An action diagram showing how requirements-related task kinds interact with requirements-related work products (after [20], with kind permission of Springer Science+Business Media).

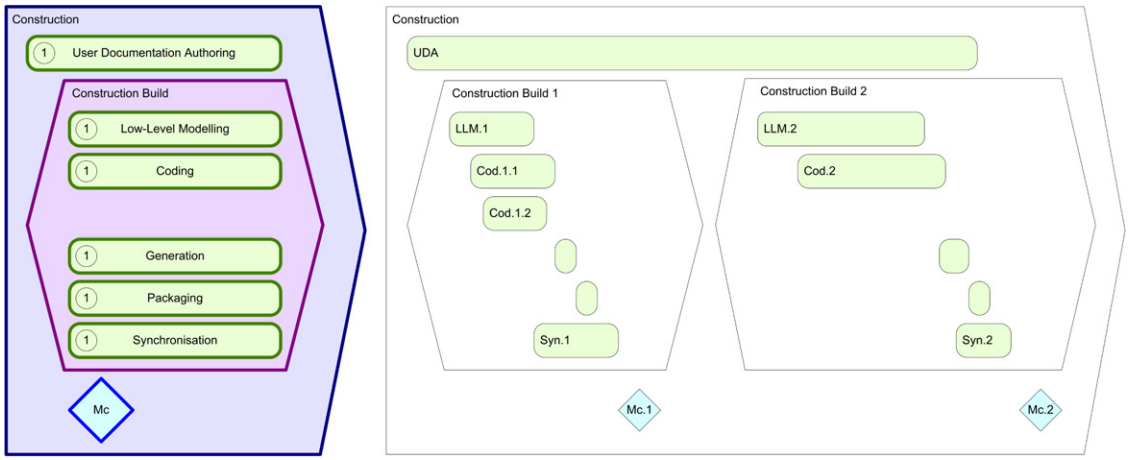

Fig. 9. An enactment diagram for the Construction phase kind of Fig. 6 (after [11]).

is no feedback that could be given by a tool to help satisfy any constraint and it is indeed more difficult to maintain consistency since the responsibility is in the user's hands. The problems in addressing the constraints that arise depend on the number of constraints that need to be remembered by the user, on the type of constraints (e.g. left alignment, right alignment, inclusion) and logical aspect of constraints (e.g. horizontal timeline is logical to users). The constraints are presented for each dimension in the following section, such as: a list of allowed operations on elements, types of possible containment of elements within another element, relationships between pairs of elements, sequence of drawing diagrams, list of abstract symbols and secondary concepts, among others. Here, the term logical means that the structures of the diagram have a direct correspondence with what they represent semantically in order to make their understanding immediate and obvious [16].

The analysis performed in this work explores only a subset of six dimensions, out of the 14 cognitive dimensions. This subset of six dimensions has been selected because only these six dimensions are explored in detail in

the Cognitive Dimensions Framework Tutorial [13,14] and they demand less understanding of cognitive psychology. The selected six dimensions are the following:

(1) Viscosity—resistance to make changes.
(2) Hidden dependencies—degree to which important links between entities are implicit.
(3) Visibility—ability to view components easily.
(4) Premature commitment—the order of notation manipulation actions is constrained.
(5) Abstraction—possibility to create new elements in the notation.
(6) Secondary notation—ability to add information on the notation outside its formal syntax.

These dimensions are used to analyse and discuss different aspects of the notation. The dimensions are more or less appropriate depending on the type of activity under analysis and their relationships, identifying the need to make trade-offs (as demonstrated in Fig. 10) when suggesting new solutions to identified issues.

3.1. Justification

There are other techniques to analyse notations; however, they do not fully address our goal of analysing the cognitive aspects as does the Cognitive Dimensions framework. For instance, the work of Olsen [27] provides a set of criteria to evaluate UI systems or notations beyond the ones covered by usability testing. In summary, it considers the following criteria:

- Importance: Importance of the analysed system/notation to a large population.
- Problem: System/notation provides a solution for a problem not previously solved.
- Generality: System/notation solves problems for different populations.
- Viscosity: System/notation reduces the effort to make changes by providing flexibility, matching problem to the solution, and clearly expressing solutions.
- Participants: Directly involved in the design process specific population who would benefit from the system/notation.
- Combination: A more powerful solution can be created by combining pieces of existing solutions.

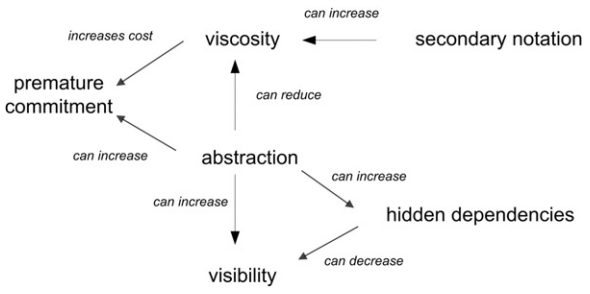

Fig. 10. Trade-offs of cognitive dimensions (modified from [3]).

Despite the importance of these criteria, this technique is rather aimed towards UI systems and has a greater focus on the speed and ease of designing UIs as well as for comparison and evaluation of design solutions using the proposed criteria. From the cognitive dimensions perspective, it covers only viscosity, its other criteria being more related to making the system useful and usable rather than addressing the detailed aspects of designing the solution (i.e. how the elements are created, whether for systems or notations), as we expect for this analysis.

Recently, Moody [25] has gathered together good advice on notation design, viewed from several physical perspectives. Much of his advice comes from Bertin [2] and, although not cited by him, the much earlier work of Constantine and Henderson-Sellers [5,6], which itself is part of the experience and skills set utilized by the designers of the ISO/IEC 24744 notation [22].

4. Evaluating the notation

For each dimension, we present a brief description: a specification of its cognitive relevance for exploratory design; examples of the analysed characteristic in the notation, when it exists; the positive factors of the notation when taking into account that dimension; the negative factors of the notation when the considered dimension is taken into account; the outcome of the evaluation regarding the cognitive relevance of the dimension; and the proposal of solutions to detected issues.

The outcome of the evaluation presents the level of each dimension attained by the elements and diagrams. The ranking of the levels follows a scale, explained as follows:

- *Viscosity*: Low: easy to make changes (few strokes to draw the elements and few components for each element). High: difficult to make changes (high interdependence between composing elements and diagrams).
- *Hidden dependencies*: Low: links between elements are explicit (there are pre-defined types of relationships between pair of elements). High: links between elements are implicit (the types of relationships between elements are unknown).
- *Visibility*: Low: elements are hard to be distinguished (different kinds of element with similar colour and shape, hardly distinguished when placed side by side). High: elements are clearly distinguished (each kind of element has a different colour and shape, easily distinguished when placed side by side).
- *Premature commitment*: Low: the order of drawing the elements is free (the sequence suggested for creating the diagrams admits overlaps and iteration). High: the order of drawing the elements is constrained (the notation enforces a pre-defined order in which to create the diagrams).
- *Abstraction*: Low: difficult to create new elements (the notation does not envision changes on the elements made by users but it does provide abstract symbols when specific types are unknown). High: possible to create new elements (the notation allows the creation of new elements, symbols, relationship types, etc.).

- *Secondary notation*: Low: lack of informal syntax (the notation is designed to depict information using only formalised elements). High: presence of informal syntax (the notation allows using informal notation to add extra information).

4.1. Viscosity

Viscosity means resistance to changes that are made; knock-on viscosity is where one change entails further actions to restore consistency. The cognitive relevance is that viscosity is harmful for exploratory design because it makes it difficult to introduce changes in specifications expressed with the notation.

4.1.1. Types of changes

For the elements in the Lifecycle Diagram and for the Enactment Diagram, there are five main elements: TimeCycleKind, PhaseKind, BuildKind, ProcessKind and MilestoneKind (Fig. 5). The elements in the Process Diagram are ProcessKind, TaskKind and ProducerKind. The elements in the Action Diagram are TaskKind, WorkProductKind and ActionKind. The basic operations that can be performed on each of these elements, independently of the use of any tool or drawing it with paper and pencil, are the following:

(1) Create the element;
(2) Modify the element's name;
(3) Enlarge the element diagonally, vertically or horizontally;
(4) Decrease the size of the element diagonally, vertically or horizontally;
(5) Change the position of the element;
(6) Delete the element.

When substituting an element for another, such as substituting an abstract element for a concrete one or substituting elements that are part of the same family (e.g. TeamKind for RoleKind), the basic operations are:

(1) Delete the existing element;
(2) Create the new element;
(3) Inform the new element's name.

In addition to all the operations common to these elements, ProcessKind, TaskKind and TechniqueKind have additional operations relating to the optional annotation for minimum capability level:

(1) Insert the minimum capability level;
(2) Modify the minimum capability level;
(3) Delete the minimum capability level.

Three important guidelines to consider when drawing are: workload, which should be minimal when drawing; explicit user control and the colour combination. These are particularly relevant to our evaluation of the Positive Factors and Negative Factors sections below.

4.1.2. Positive factors

For the Lifecycle and Enactment diagrams, we can evaluate that the workload varies depending on the capabilities of the adopted tool when there is a need to redraw. For instance, if there is the need to insert a new process, there are two possible solutions: For paper and pencil, the user needs to erase the already-drawn elements, redraw them and place new ones appropriately; this leads to a high workload. Using a CASE tool or a drawing tool, the user only needs to enlarge the termination of the elements (e.g. TimeCycleKind, PhaseKind and BuildKind) in order to include the new element, leading to a low workload.

According to the theories of short term memory coupled with the limits on human capacity for processing information, it is generally regarded as possible for the user to retain between five and nine symbols [24]. Since there are five main symbols in the Lifecycle and Enactment diagrams and less than five on the Process and Action diagrams, to draw them, the required workload is low. The elements are easy to handle and also to change characteristics of the elements, such as name and size.

To draw the Lifecycle and Enactment diagrams, there is a need for explicit user control to define the size of each element. Using paper and pencil, users are free to draw the elements and size them as necessary depending on the number of elements to be inserted inside others, for instance, depending on the number of ProcessKinds inside a PhaseKind. Thus, there is a good level of explicit user control for sizing and placing the elements. Using a drawing tool, the elements can be resized, usually by the edges and that also brings a good level of explicit user control. The Process and Action diagrams, which do not require placing elements inside others, decrease the workload even more since there are less space restrictions.

4.1.3. Negative factors

Changes in the length of the process are difficult in the enactment diagram because the timeline is not visible. Resistance to make changes in elements that are spread across different diagrams is inherent to any diagramming system. Nonetheless, for fairness, some examples of this constraint are expressed as follows: Changes in the process diagram require changes in two more diagrams: lifecycle and enactment diagrams because of a common element in all of them: ProcessKind. Another example of such difficulty happens between the process and action diagrams when changes are made on TaskKinds. Thus, drawing these diagrams presents a high knock-on viscosity, which leads to extra work on the related diagrams to keep consistency and leads to a high level of workload.

4.1.4. Outcome

High knock-on viscosity can lead to deeper reflection and planning before making changes, which is highly recommended for Software Development Methodologies (SDM). On the other hand, high viscosity should generally be avoided. The overall analysis shows that the diagrams have a high knock-on viscosity concerning the interdependence between diagrams, where changes in one lead to specific changes in others that contain the same

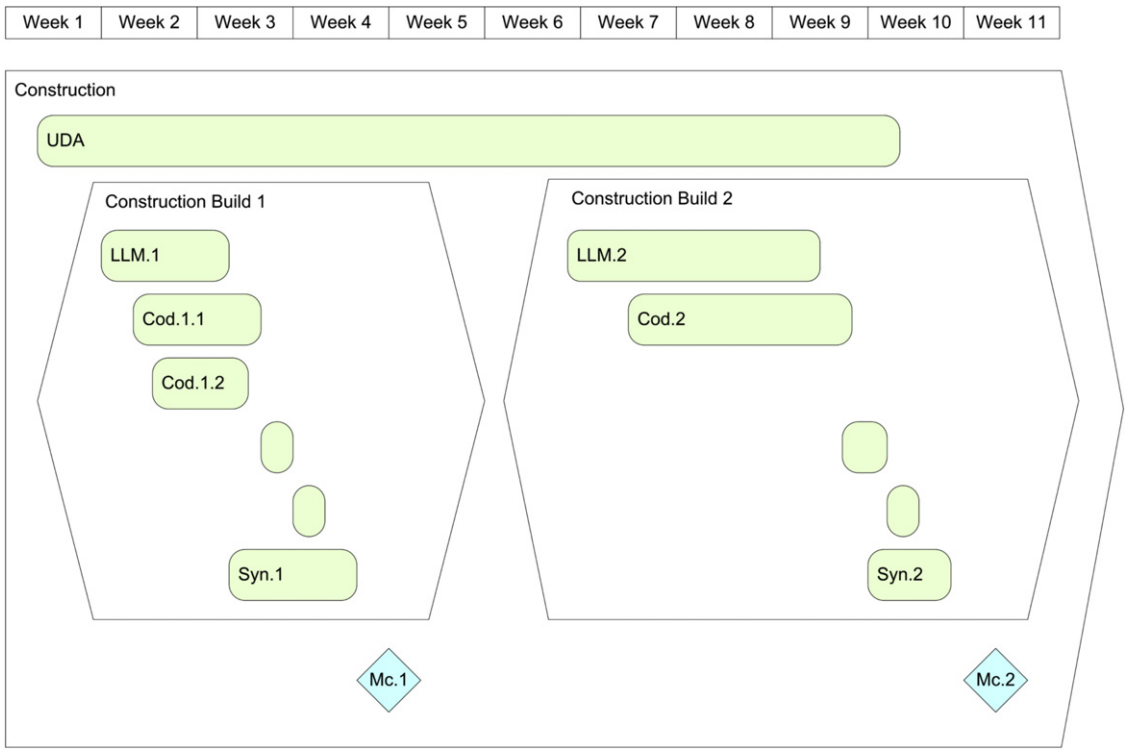

Fig. 11. Enactment diagram with timeline.

elements, although this is inherent to any notation that assumes a diagram as a view on the underlying model. On the other hand, the elements of the various diagrams have a low level of viscosity when changing them.

4.1.5. Suggestions

One suggestion to decrease resistance to make changes is related to an improvement in the enactment diagram. Considering that the enactment diagram shows a specific enactment of a particular method that includes a Gantt chart linked to the symbols of the lifecycle diagram, a visual timeline can be added as a new element that can be optionally used. Even if there is not a project schedule with specific dates, this diagram could at least show a timeline. A horizontal square element could be added in the enactment diagram, which could be placed on top of the diagram. This element could be divided in columns that represent days, weeks, months or years (added together as necessary) to facilitate the identification of the length of ProcessKinds. This element could be visually similar to the Constraint element.

Once this element is included in the Gantt chart, it is more precise if the elements have a rectangular format in order to facilitate locating them within the width of a specific column in the timeline. When there is a need to change the size of a ProcessKind in the enactment diagram and there is no timeline, the method engineer would have no idea of the width of the ProcessKind in relation to other ProcessKinds. The timeline helps scale the elements, and consequently makes the change easier because changing the size of elements is guided by the timeline. As a result of this suggestion, as presented here, the enactment diagram has been updated by the ISO authoring team responsible for this notation to include the timeline, as depicted in Fig. 11.[5]

4.2. Hidden dependencies

Hidden dependencies happen when the relationship between two components is not fully visible. The cognitive relevance of hidden dependencies is acceptable for small tasks in exploratory design, although it is expected to be low in order to make links between elements explicit.

4.2.1. Types of dependencies

Topological containment: Topological containment can occur for two reasons: (i) the subtypes of StageKind elements: TimeCycleKind, PhaseKind and BuildKind correspond to stages with duration (managed timeframe within an endeavour) and can contain other elements; and (ii) a conglomerate can contain other elements.

Relationships: There are some relationships between pair of elements:

- ActionKind is a usage event between TaskKind and WorkProductKind.

[5] We were later informed that this change had been proposed for inclusion in the final version of the ISO/IEC 24744 notation, published in 2010.

- WorkPerformanceKind associates one of the subtypes of ProducerKind and one of the subtypes of WorkUnitKind.
- Whole/Part Link is a representation of whole/part relationships between elements.
- Generic link is a representation of association between elements.

4.2.2. Positive factors

When method engineers learn the main relationships between the elements and use them correctly, they can easily specify the containment properties and relationships between elements within diagrams. The ActionType and Deontic values make the relationship between elements very clearly specified.

4.2.3. Negative factors

The direct relationship between producer kinds and work product kinds does not clearly depict which specific roles are responsible for producing which work products. This example of hidden dependency makes the specification of certain details in methods unclear, thus decreasing the understandability of the designed method.

4.2.4. Outcome

The level of hidden dependencies is low since most relationships between elements in different diagrams are possible to detect and relationships within the diagrams are very clear with the use of relationship concepts.

4.2.5. Suggestions

To solve a matter of expressiveness and show major links between elements in the notation, the Action diagram can be modified adding producer kinds as one of its elements in order to associate producer kinds directly with work product kinds. Having a diagram that has these three elements, producer kinds, task kinds and work product kinds together, facilitates specifying and understanding the method since the method engineer is able to express a wider variety of relationships in the same diagram, resulting in less hidden dependencies and increased visibility. Therefore, we suggest depicting which producer kinds are responsible for which task kinds and work products in the Action diagram, as illustrated in Fig. 12.

Another aspect is that the association of various producer kinds to process kinds and task kinds could be detailed with the specification of goals, where each producer kind may have a different goal towards the same task kind. For instance, associating the producer kinds called Quality Assurance Manager and Project Manager to the task kind "Validate requirements" can be described as follows: the Quality Assurance Manager *performs* the "Validate requirements" task kind and the Project Manager *assists* in the same task kind; where Perform and Assist are types of goal. Different types of goals could be enumerated in an abbreviation table as a set of possible goals that producer kinds may have towards task kinds, as presented in Table 1 and exemplified in the modified action diagram in Fig. 12.

As a consequence of indicating producer goals through abbreviations, it is possible to express which producer kinds have the primary responsibility for which work product kinds. This kind of information decreases another hidden dependency between elements of the notation since abbreviations are useful to differentiate the types of goals that each producer kind has towards the same task kinds since there are several producers that can be linked to the same task kind.

The specification of goals is optional, so one is free not to specify goals for the purpose of decreasing the number of operations done on this diagram. However, as method engineers work on evolving the action diagram and when there is a need to specify the roles more clearly, they could assign a goal for producer kinds. It is true that this suggestion adds more symbols on the notation and, as a consequence, there is added viscosity, but this is important to be more precise on the specification of producer kinds and their roles. In the interest of decreasing viscosity with the use of goals for producer kinds, the suggested abbreviations are logical for users, to the possible detriment of other feasible types of representations, such as using plain or dotted lines or different kinds of edges, which may add even more complexity, especially since the notation adopts plain lines to all edges of symbols for visibility reasons (with the only exception being Person).

4.3. Visibility and Juxtaposition

Visibility is the ability to view components easily and juxtaposition is the ability to place any two components side by side. The cognitive relevance of visibility and juxtaposition is that they are important for exploratory design; therefore users should be able to easily view, distinguish and position the elements of the notation.

4.3.1. Positive factors

For the visibility concerning the combination of colours, it is generally agreed that the number of colours should be between 6 and 8 [1]. The elements in the Lifecycle and Enactment diagrams use seven different fill colours, which are: (1) light blue–grey for StageWithDurationKind, PhaseKind; (2) light purple for BuildKind; (3) light blue for InstanteneousStageKind, MilestoneKind; (4) light olive for ProcessKind and TechniqueKind; (5) light green for TaskKind; (6) very light pink for work products; and (7) light yellow for producers.

Research performed by Murch [26] presents a constraint to guidelines related to colours, which states that the combination between the background colour and the foreground colour should belong to the best colour combination or should not belong to the worst colour combinations. Table 2 depicts the constraint on colour combination that are addressed by the notation.

Concerning juxtaposition, the overall structure of the method can be demonstrated in the lifecycle diagram and in the enactment diagram. Consequently, the method can be understood with these two diagrams with juxtaposition of the elements that show content as well as temporal structure. The repetition in the enactment diagram brings visibility with an extract of the lifecycle

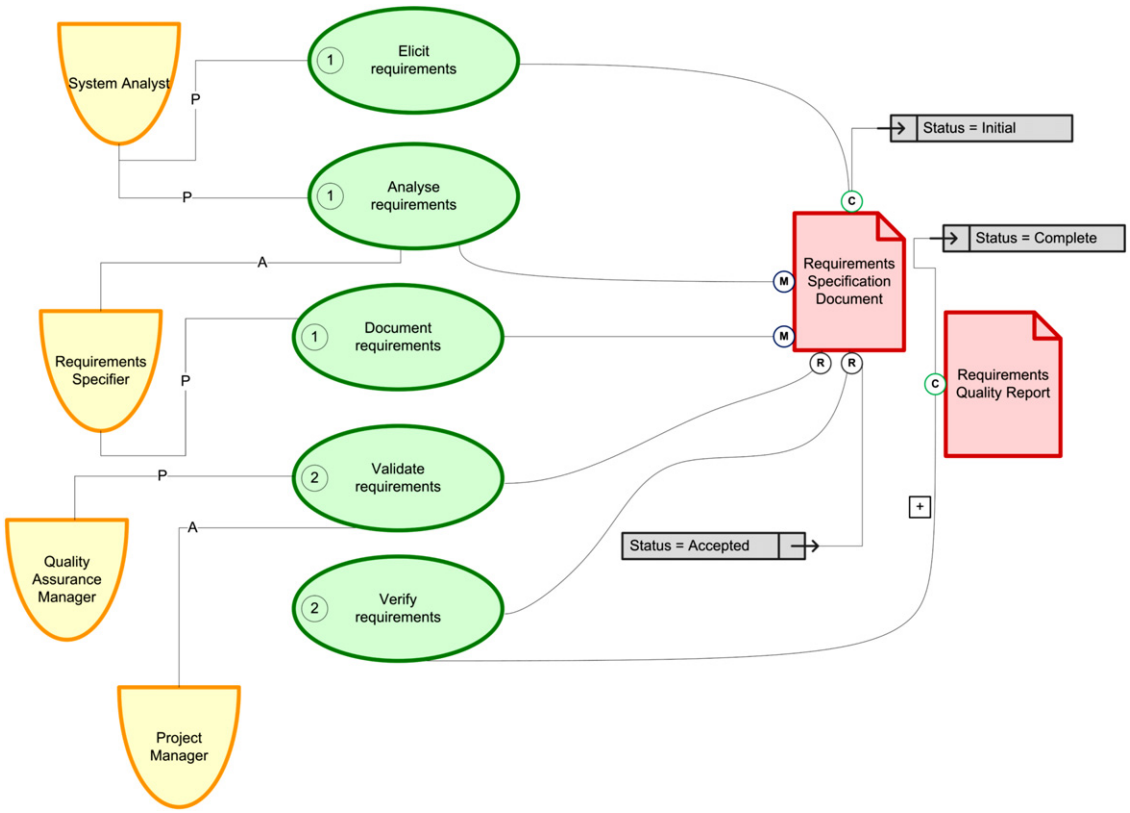

Fig. 12. Modified action diagram with Producer Kinds (cf. Fig. 8).

Table 1
Goals for producer kinds.

Abbreviation	Name	Semantics
P	Performs	A producer of the specified kind performs a task of the specified kind
A	Assists	A producer of the specified kind assists a task of the specified kind
…	…	…

diagram placed just beside the part of the diagram for the Gantt chart.

4.3.2. Negative factors

For the combination of colours, it is not recommended to have too many variations of blue in the same diagram because the eye hardly sees objects which differ by close nuances of blue and this can lead to potential perception problems [26]. We detected two cases that belong to poor colour combinations, as depicted in Table 3. It is important to point out that this research study was conducted by considering colour contrast and not specific values of saturation, hue and brightness.

Visibility issues on the position of elements are related to the inability to directly indicate which producer kinds are responsible for a specific work product. This is indicated only indirectly in both process and action diagrams: the process diagram depicts the association of producer kinds with process kinds, which are, in their turn, linked to task kinds; the action diagram shows the links between these tasks kinds and work products. Nonetheless, when there is more than one producer kind associated to the same process kind, method engineers cannot clearly indicate the association of certain producer kinds to work product kinds. In this scenario, method engineers have difficulty to clearly express relevant information of the method being created. As a result of hidden dependencies between these elements, there is a visibility issue.

4.3.3. Outcome

For the lifecycle and enactment diagrams, the level of visibility is high, even as a consequence of the detected low level of hidden dependencies. For the enactment diagram, the level of juxtaposition is high since it shows the Gantt chart linked to the symbols of the lifecycle diagram. For the process and action diagrams, the level of juxtaposition is low for some elements (e.g. producer kinds and work product kinds) that could be associated, but that are placed in different diagrams.

4.3.4. Suggestions

The suggestion to add more expressiveness to the notation by adding producer kinds in action diagrams (as suggested for hidden dependencies) already improves the level of juxtaposition. To add more clarity to the diagrams, the process diagram could adopt the strategy of

Table 2
Adherence to colour combination.

Elements	Colour Combination	Constraint
MilestoneKind on BuildKind	For bold lines and panels displayed on magenta background (closest to purple on BuildKind)	Blue is a good choice in approximately 50% of the cases
MilestoneKind on TimeCycleKind	For bold lines and panels displayed on white (background of TimeCycleKind)	Blue is a good choice in approximately 63% of the cases
BuildKind on PhaseKind	For bold lines and panels displayed on blue (background of PhaseKind)	Magenta (closest to purple) is a good choice in approximately 31% of the cases

Table 3
Constraints on colour combination.

Elements	Colour Combination	Constraint
ProcessKind on BuildKind	For bold lines and panels displayed on magenta (closest to purple on BuildKind)	Green induces a possible legibility problem in nearly 69% of the cases
ProcessKind on PhaseKind	For bold lines and panels displayed on blue (background of PhaseKind)	Green induces a possible legibility problem in nearly 44% of the cases

using team kind, and the action diagram using role kind, since the action diagram can represent a detailed view of the process diagram.

Most colour combinations are appropriate, with two exceptions: when ProcessKind (green bold lines) is inserted within BuildKind (magenta background) or PhaseKind (blue background), inducing a possible legibility problem. In those cases, the colours that best fit on magenta background using bold lines are: blue (for 50% of the cases), black (for 44% of the cases) and yellow (for 25% of the cases). The colours that best fit on blue background using bold lines are: yellow (for 38% of the cases), magenta (for 31% of the cases) and black (for 31% of the cases).

4.4. Premature commitment

Premature commitment is related to constraints on the order of doing things that force the method engineer to make a decision before the proper information is available. The cognitive relevance of premature commitment is that it is harmful for exploratory design so there should be little constraints to manipulate the notation.

4.4.1. Examples of sequences

The lifecycle diagram can be created before the enactment diagram, which contains a particular method or a method chunk from the lifecycle diagram. The lifecycle diagram can be created before the process diagram because it is important to first specify the main processes in the lifecycle diagram before detailing them in the process diagram. The process diagram can be created before the action diagram, by the same reasoning as the previous one, since it is important to first specify the main tasks in the process diagram before detailing them with links to work products in the action diagram. These examples represent an overall sequence imposed by the metamodel, and which is reflected in the notation by introducing and describing the diagrams in this sequence. This sequence, however, admits a great deal of iteration, as presented in the following section, with some examples of overlaps in the creation of the diagrams.

4.4.2. Positive factors

The sequences listed above are not enforced by the notation; therefore, the method engineer has flexibility in creating the diagrams in any desired order, without being forced to think ahead and make certain fixed decisions first. For instance, the lifecycle and process diagram, as well as the process and the action diagram, can be created in an iterative manner. On the other hand, the enactment diagram can only be created based on the lifecycle diagram, but this constraint does not cause any problem since the method engineer is able to make changes in both diagrams whenever necessary.

The method engineer does not have to make premature commitment on the quantity of elements to include within "nesting" symbols (that is, symbols that contain other symbols) because they are represented by broad symbols that can contain other elements and they are resizable (whether drawing on paper or using a tool, despite the level of difficulty in each case). There are no constraints on specific decisions, such as which deontic values or types of action to use. They are not mandatory at early stages; therefore they can be included in the diagrams when the method engineer has enough information to make this decision.

4.4.3. Outcome

The level of premature commitment is low for the creation of the diagrams. Actually, method engineers make use of the notation to create diagrams but they must follow certain method engineering techniques, which are the aspects that mostly influence whether or not there is the need for premature commitment.

4.5. Abstraction

Abstraction is related to types and availability of abstraction mechanisms. The characteristic of an abstraction from the point of view of cognitive dimensions is that it changes

the notation. The cognitive relevance of abstraction is that it may be harmful for exploratory design, although it should be considered when the creation of new elements is helpful to manipulate and organize elements of the notation, which may result in increased visibility.

4.5.1. Types of abstraction

The following list has abstract classes that use abstract symbols:

- StageWithDurationKind
- InstantaneousStageKind
- WorkProductKind
- ProducerKind
- Constraint

The notation uses abstract symbols to represent an entity in a diagram for which only the abstract type is known. The notation does not envision changes on the elements made by users.

4.5.2. Positive factors

The introduction of these existing abstract classes makes the notation easier to use and avoids method engineers making premature commitments. For instance, method engineers can use the abstract icon of producer kind in initial versions of the process diagram and the abstract of work product kind in initial version of the action diagram, when the specific work product or role is not yet clearly defined, respectively.

4.5.3. Negative factors

Some enumerated types and the enumerators (which provide values) are provided by the standard, but users are allowed to augment either of these through the extension mechanisms provided in the standard. For instance, the enumerated types of actions and deontic values are fixed and this rigidness makes method engineers limited to represent different scenarios. Providing the ability to increase this list does not lead to major changes on the elements or their symbols.

4.5.4. Outcome

The level of abstraction is low since the notation does not enable the definition of new elements. However, there are some elements with abstract symbols, which are helpful to give more flexibility and reduce the viscosity for creating the diagrams.

4.5.5. Suggestions

Using abstractions, it is possible to change the notation by adding new elements that have to be then mastered by subsequent users. Using abstractions can be costly because it requires users to think in abstract terms, which can be difficult for users to learn the notation. Abstractions also require time and effort for creating and maintaining the new elements. Even recognizing those shortcomings, improving the list of possible types of actions and deontic values can increase the visibility of the diagrams that use these concepts. We believe that the lifecycle of the work products can be extended by considering a larger taxonomy of action types.

Examples of action types that can be performed on work products can be: duplicate, analyse, share, browse, search, as well as a collection of different types used simultaneously. Research has created a list of recommended action types, grouping together actions with similar meanings [10]. The result is presented as a taxonomy of action types that can be expanded beyond its present scope and used in the context of actions performed on work products. This suggestion is also applicable if there is an initial list of goals, as suggested for hidden dependencies.

For this suggestion, we must consider the trade-off between guidance and flexibility. A fixed set of action types increases guidance with particular options, but it decreases flexibility without the possibility to add new options. On the other hand, a changeable list of recommended action types increases flexibility, but it decreases guidance since the list could be in constant change. The notation can take not only a balanced approach and support a condensed view with few types to provide guidance but also an optional detailed view that allows extending the initial list with new action types when needed.

4.6. Secondary notation

Secondary notation is related to extra information, which can be available in two main forms. Redundant coding gives a different form for information that is already present in the formal notation. Some examples are indentation and comments in programming and grouping related elements in the same area. Escape from formalism shows extra information, not present in the formal notation, such as annotation on diagrams. The cognitive relevance of secondary notation is that it may be useful (to facilitate the understanding of certain aspects of the notation) or could be harmful (costly to learn both formal and informal syntax and to make it available in tools) for exploratory design, so there must be a trade-off analysis for each case.

4.6.1. Types of secondary notation

Secondary notation is designed to depict concepts from the method domain (Fig. 3) to be used by method engineers when representing method fragments or complete methods. It is composed of formalised elements and abbreviations to be used in relationships between elements. Based on this definition, we can state that currently there is no secondary notation for ISO/IEC 24744.

4.6.2. Positive factors

Once a secondary notation is defined, it should be accepted by all groups of users, otherwise the graphics created could be interpreted differently by each group. Therefore, the main advantage of not having secondary notation is that users of the notation do not have to learn new meanings of how information is shown, thus avoiding any misunderstandings.

4.6.3. Outcome

There is low level of secondary notation since the notation is composed of formalised elements and relationships. The only element that allows including extra information is the Guideline.

5. Complexity analysis

Information Visualization techniques provide abstractions and representations of information in a manner that presents details but, at the same time, it also shows it in a context to facilitate understanding relationships. We have performed a complexity analysis founded on the area of Information Visualization [36]. Table 4 presents a complexity analysis of the main elements (icons) of the notation, hereafter called elements.

Complexity Analysis demonstrates the physical complexity of drawing the elements using this notation. The goal of using this analysis is to add a perspective concerning the act of drawing besides the cognitive aspects already analysed using Cognitive Dimensions.

Each of the criteria used to undertake the complexity analysis of the elements are expressed separately in the table in order to explicitly describe all elements considering each criteria and keeping consistency. For the criteria 'Inclusion', S represents Simple Inclusion and C represents Complex Inclusion.

Each of the criteria used to analyse the elements are described as follows:

Number of components: *The number of components that compose an element.* All of the elements are composed of only one component.

Number of strokes: *The number of lines delineated to draw an element.* Each element has its specific number of strokes to be drawn with paper and pencil. This numbering does not apply when using a drawing tool when you only drag and drop the elements.

Specific orientation: *The state of an element of being placed in its proper position.* All elements have a specific and fixed orientation as they are visually designed, since their visualization cannot be changed. If their orientation changes, it could even cause some elements to be confused with another one, as is the case of ProducerKind and RoleKind. For example, if the orientation of RoleKind was changed vertically, it would have the same orientation as ProducerKind, thus the user would not know the difference between these two elements.

Simple inclusion: *The state of an element that contains only one level of subordinate elements.* The only element for which this applies is BuildKind, which can contain ProcessKind and MilestoneKind, elements that do not contain other elements.

Complex inclusion: *The state of an element that contains subordinate elements, which can also contain other subordinate elements.* There are several examples of this in Table 4. StageWithDurationKind is complex since this abstract class can be any of the subtypes of StageKind. It inherits the characteristics of the element that has the highest level of complexity, that is, with more elements inside other elements, which is TimeCycleKind. TimeCycleKind has three levels of inclusion since it can contain (i) MilestoneKind (alone); (ii) PhaseKind (which can contain only ProcessKind); and (iii) PhaseKind (which can contain BuildKind, which in its turn can also contain other elements inside it: ProcessKind and MilestoneKind).

Table 4
Analysis of elements in lifecycle diagram.

Name	Nr. components	Nr. of strokes	Specific orientation	Inclusion	Juxtaposition	Intersection	Sequence
StageWithDurationKind	1	4	X	C	X		X
TimeCycleKind	1	6	X	C			X
PhaseKind	1	5	X	C	X		X
BuildKind	1	6	X	S			X
InstantaneousStageKind	1	4	X		X		X
MilestoneKind	1	4	X		X		X
ProcessKind	1	1	X		X		X
TaskKind	1	1	X		X		X
TechniqueKind	1	1	X		X		X
WorkProductKind	1	4	X		X		
DocumentKind	1	6	X		X		
ModelKind	1	5	X		X		
SoftwareItemKind	1	5	X		X		
HardwareItemKind	1	8	X		X		
CompositeWorkProductKind	1	8	X		X		
ProducerKind	1	2	X		X		
TeamKind	1	4	X		X		
RoleKind	1	2	X		X		
ToolKind	1	6	X		X		
Person	1	4	X		X		
PreCondition	1	7	X		X		
PostCondition	1	8	X		X		
Conglomerate	1	7	X	C	X		
Guideline	1	6	X		X		

PhaseKind has two levels of inclusion since it can contain (i) only processKind; and (ii) BuildKind with ProcessKind and MilestoneKind inside it. Conglomerate has three levels of inclusion since it can contain any other symbol, similar to the TimeCycleKind.

Juxtaposition: *The state of elements being placed side by side.* Almost all elements are relevant here. The elements StageWithDurationKind, PhaseKind, InstantaneousStageKind, MilestoneKind are placed side by side on the lifecycle diagram. Producers, work units and guidelines are placed side by side on the process diagram. Work units, work products and constraints are placed side by side on the action diagram. Conglomerate can be placed side by side.

Intersection: *A point where a line of an element crosses the line of another one when they are used in diagrams.* There are no intersections in any of the elements.

Sequence: *The order of succession of a same kind of element in a diagram.* Nine elements can be placed in a sequence:

Stage with Duration Kind—This element (if repeated in the same diagram) could be organised in a horizontal sequence (from left to right) in the lifecycle diagram.

TimeCycleKind—This element (if repeated in the same diagram) could be organised in a horizontal sequence (from left to right) in the lifecycle diagram.

PhaseKind—This element is organised in a horizontal sequence (from left to right) in the lifecycle diagram.

BuildKind—This element is organised in a horizontal sequence (from left to right) in the Gantt chart of the enactment diagram.

InstantaneousStageKind—This element is organised in a horizontal sequence (from left to right) in the lifecycle diagram and in a vertical sequence in the enactment diagram.

MilestoneKind—This element is organised in a horizontal sequence on the right of the element in the lifecycle diagram. This element is organised in a vertical sequence below the previous element in both parts of the enactment diagram (lifecycle and Gantt chart).

ProcessKind—This element is organised in both horizontal and vertical sequences in the lifecycle diagram. This element is organised in a vertical sequence below the previous element in both parts of the enactment diagram (lifecycle and Gantt chart) and in the process diagram.

TaskKind—This element is organised in a vertical sequence below the previous element in the process and action diagrams.

TechniqueKind—This element is not organised in sequence, but it is placed closer to the task associated to it in the process diagram.

The remaining elements and their subtypes are not able to be organised in a sequence; they are placed closer to the element associated with them.

In summary, the elements are simple and easily drawn considering different criteria: (i) they are composed of only one component; (ii) they have a fixed and specific orientation; (iii) most elements do not contain subordinate elements, only one has a simple inclusion and four with complex inclusion; (iv) there are no intersections as the lines of an element do not cross the line of other elements since they are mostly placed side by side and they can be organised in a horizontal or vertical sequence to make it easier to read the diagrams.

6. Conclusion

This work has presented an analysis of a graphical notation for ISO/IEC 24744 [21] for modelling software development methodologies in which we considered both the point of view of method engineers and of IT professionals. The cognitive dimensions were used to facilitate the analysis and discussions with other experts. The main contributions of this evaluation are: (i) improving the enactment diagram to facilitate organizing processes in accordance with a visual timeline; (ii) improving the action diagram by adding producer kinds that can better express responsibilities in a detailed view; (iii) specifying the types of goals of producer kinds towards task kinds and work product kinds; (iv) making existing tables more flexible to add the potential of increasing the list of possible types of actions, deontic values and producer goals; (v) identification of certain colour combinations that may induce to legibility problems in the lifecycle diagram. In general, our main focus is to apply the different suggestions in this notation in order to facilitate designing and understanding methods in the daily routine of software projects.

These suggestions were communicated to the ISO SC7 Working Group responsible for the development of the notation for IS 24744. We were advised that they influenced discussions at the Plenary Meeting held in Berlin in May 2008 and many of our suggestions were incorporated into the final version.

Acknowledgements

We acknowledge the support by the Program Alban, the European Union Program of High Level Scholarships for Latin America, under scholarship number E06D103843BR. This is Contribution number 09/04 of the Centre for Object Technology Applications and Research at the University of Technology, Sydney.

References

[1] W.W. Banks, W.E. Gilmore, H.S. Blackman, D.I. Gertman, Human Engineering Design Considerations for Cathode Ray Tube-Generated Displays, vol. II, document NUREG (Contractor's reference: CR-3003/ EGG-2230), U.S. Dept. of Energy, Idaho National Engineering Laboratory, Idaho, 1983.

[2] J. Bertin, Semiology of Graphics: Diagrams Networks, Maps. Univ. Wisconsin Press, Madison, WI, USA, 1983.

[3] A.F. Blackwell, T.R.G. Green, Notational systems—the cognitive dimensions of notations framework, in: J.M. Carroll (Ed.), HCI Models Theories and Frameworks, Morgan Kaufmann, San Francisco, 2003, pp. 103–134.

[4] S. Brinkkemper, Formalisation of Information Systems Modelling, Ph.D. Thesis, University of Nijmegen, Thesis Publishers, Amsterdam, 1990.

[5] L.L. Constantine, B. Henderson-Sellers, Notation matters. Part 1. Framing the issues, Report on Object Analysis and Design 2 (3) (1995) 25–29.

[6] L.L. Constantine, B. Henderson-Sellers, Notation matters. Part 2. Applying the principles, Report on Object Analysis and Design 2 (4) (1995) 20–23.
[7] D.G. Firesmith, B. Henderson-Sellers, The OPEN Process Framework. An Introduction, Addison-Wesley, Harlow, Herts, UK, 2002.
[8] B. Goransson, J. Gulliksen, I. Boivie, The usability design process—integrating user-centered systems design in the software development process, Software Process: Improvement and Practice 8 (2) (2003) 111–131.
[9] C. Gonzalez-Perez, B. Henderson-Sellers, Modelling software development methodologies: a conceptual foundation, Journal of Systems Software 80 (11) (2007) 1778–1796.
[10] J.M. Gonzalez Calleros, J. Vanderdonckt, J. Muñoz Arteaga, A structured method for designing usable 3D Web applications, in: T. Spiliotopoulos (Ed.), Usability Engineering for Designing the Web Experience, IGI Global Inc., Hershey, PA, 2009.
[11] C. Gonzalez-Perez, B. Henderson-Sellers, Metamodelling for Software Engineering, John Wiley & Sons, Chichester, UK, 2008.
[12] I. Graham, B. Henderson-Sellers, H. Younessi, The OPEN Process Specification, Addison-Wesley, UK, 1997.
[13] T.R.G. Green, Cognitive dimensions of notations, in: A. Sutcliffe, L. Macaulay (Eds.), People and Computers V, Cambridge University Press, UK, 1990 (Chapter 32).
[14] T.R.G. Green, A.F. Blackwell, Cognitive Dimensions of Information Artefacts: A Tutorial. Version 1.2, October 1998.
[15] G. Guizzardi, Ontological Foundations for Structural Conceptual Models, Enschede, The Netherlands, 2005.
[16] C.A. Gurr, Aligning syntax and semantics in formalisations of visual languages, IEEE Symposia on Human-Centric Computing Languages and Environments 20 (2001) 60–61.
[17] B. Henderson-Sellers, Process metamodelling and process construction: examples using the OPEN Process Framework (OPF), Annals of Software Engineering 14 (2002) 341–362.
[18] B. Henderson-Sellers, Method engineering: theory and practice, in: Karagiannis, D., Mayr, H.C. (Ed.), Information Systems Technology and its Applications, 5th International Conference ISTA 2006 Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, 2006 (Chapter 1).
[19] B. Henderson-Sellers, C. Gonzalez-Perez, Connecting powertypes and stereotypes, Journal of Object Technology 4 (7) (2005) 83–96.
[20] B. Henderson-Sellers, C. Gonzalez-Perez, Standardizing methodology metamodelling and notation: an ISO exemplar, in: R. Kaschek, C. Kop, C. Steinberger, G. Fliedl (Eds.), Information Systems and e-Business Technologies Lecture Notes in Business Information Processing, Springer, Berlin, 2008, pp. 1–12. (Chapter 1).
[21] ISO/IEC: 24744: Software Engineering—Metamodel for Development Methodologies. International Organization for Standardization/International Electrotechnical Commission, Geneva, 2007.
[22] ISO/IEC: 24744 Software Engineering—Metamodel for Development Methodologies Annex A Notation. International Organization for Standardization/International Electrotechnical Commission, Geneva, 2010.
[23] D. Mayhew, The Usability Engineering Lifecycle—A Practitioner's Handbook for User Interface Design, Morgan Kaufmann Publishers, San Francisco, CA, 1999.
[24] G.A. Miller, The magical number seven, plus or minus two: some limits on our capacity for processing information, Psychological Review 63 (1956) 81–97.
[25] D.L. Moody, The physics of notations: toward a scientific basis for constructing visual notations in software engineering, IEEE Transactions on Software Engineering 35 (6) (2009) 756–779.
[26] G.M. Murch, Colour graphics—blessing or ballyhoo the visual channel, in: R.M. Baecker, W.A.S. Buxton (Eds.), Readings in Human-Computer Interaction—A Multidisciplinary Approach, Morgan Kaufmann Publishers, Los Altos, 1987 (Chapter 7).
[27] D.R. Olsen Jr., Evaluating user interface systems research, in: 20th ACM Symposium on User Interface Software and Technology (2007) 251–258.
[28] OMG, Unified Modelling Language Specification, Version 1.4, OMG Documents Formal/01-09-68 Through 80 (13 Documents), 2001. Available at 〈http://www.omg.org〉.
[29] J. Ralyté, Towards situational methods for information systems development: engineering reusable method chunks, in: O. Vasilecas, A. Caplinskas, W. Wojtkowski, W.G. Wojtkowski, J. Zupancic, S. Wrycza (Eds.), Proceedings of 13th International Conference on Information Systems Development, Advances in Theory, Practice and Education, Vilnius Gediminas Technical University, Vilnius, Lithuania, 2004.
[30] J. Ralyté, C. Rolland, V. Plihon, Method enhancement by scenario based techniques, in: M. Jarke, A. Overweis (Eds.), 11th International Conference, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1999 (Chapter 9).
[31] OMG, Software Process Engineering Metamodel Specification, v2.0, February 2007.
[32] V. Seidita, J. Ralyté, B. Henderson-Sellers, M. Cossentino, N. Arni-Bloch, A comparison of deontic matrices, maps and activity diagrams for the construction of situational methods, in: J. Eder, S.L. Tomassen, A.L. Opdahl, G. Sindre (Eds.), CAiSE'07 Forum, Proceedings of the CAiSE'07 Forum at the 19th International Conference on Advanced Information Systems Engineering, CEUR-WS Org, 2007 (Chapter 23).
[33] J.P. Tolvanen, Incremental Method Engineering with Modeling Tools, Dissertation, Jyvaskyla Studies in Computer Science, Economics and Statistics, vol. 47, University of Jyvaskyla, Finland, 1998.
[34] E.R. Tufte, The Visual Display of Quantitative Information, Graphics Press, 1992, pp. 1–197.
[35] K. Van Slooten, S. Brinkkemper, A method engineering approach to information systems development, in: N. Prakash, C. Rolland, B. Pernici (Eds.), Information Systems Development Process, Proceedings IFIP WG8.1, Elsevier Science Publishers B.V., North-Holland, 1993 (Chapter 10)
[36] C. Ware, Information Visualization: Perception for Design, Morgan Kaufmann, San Francisco, CA, 2004.