Beyond Methodologies: Keeping up with Information Systems Development Approaches through Dynamic Classification

Juhani Iivari  
Department of Information Processing Science  
University of Oulu  
P.O. Box 333  
FIN-90570 Oulu, Finland  
iivari@rieska.oulu.fi

Rudy Hirschheim  
College of Business Administration  
University of Houston  
Houston, TX 77204-6282  
rudy@uh.edu

Heinz K. Klein  
School of Management  
SUNY Binghamton  
Binghamton, NY  
hkklein@binghamton.edu

Abstract

There are hundreds of information systems development methodologies (ISDMs) which differ significantly from each other. Yet, the number of ISDMs that an organization can consider for adoption or adaptation must be quite small for practical reasons. In order to make an informed choice for the selection of preferred ISDMs a hierarchical classification is needed that separates fundamental or “essential” features from “accidental” details. The principal purpose of this paper is to propose such a classification that would allow academics and practitioners alike keeping up with continuing growth of the “methodology jungle”. According to this structure, an ISDM is merely an instantiation of one or more information systems development approach (ISDA). ISDAs comprise the essential features, which are then inherited by the ISDMs belonging to that class. One important implication of organizing the field of ISDMs into a comprehensible number of ISDAs is to shift the discussion from individual ISDMs to the features of more general ISDAs, as expressions of the essences of their instance methodologies. The condensed presentations of ISDAs make it possible to broaden the methodological repertoire of systems developers. They allow flexible, situation-specific “methodology engineering” in which an ISDM is adapted to a specific ISD project through an instantiation of an appropriate ISDA.

1. Introduction

Even though a universally accepted, rigorous and concise definition of an information systems development methodology (ISDM) is lacking, the core idea is clear: a systematic procedure for completing either a system or one of several major stages of the systems development life cycle. An ISDM consists of goals, principles, specific methods and tools, which are selected on the basis of an underlying rationale or systems development philosophy (cf. [37] for an introduction to related terms). The number of different ISDMs is truly remarkable. Jayaratna [21] estimates that there are over a 1000 ISDMs. This proliferation can be expected to continue ([10], [19]).

Given the effort directed to the development of ISDMs one would expect a lively discussion on how to select methodologies in practice either for direct application or adaptation (“reengineering”) and on how to catalogue them as a prerequisite to study the merits of various features and identify the most useful ones. However, neither seems the case – even though feature analyses were started as early as in the 1970’s (cf. [35], [36]) and later showcased in one of the CRIS conferences ([27]-[29]). Basically, their level was too detailed in two respects. Firstly, ISDMs seem to represent a too detailed level of granularity for a meaningful comparison (see section 2.1). Secondly, clearly not all features are equally central to an ISDM and this suggests focusing on a few fundamentals. One wonders how an informed choice can be made for the initial selection of the preferred ISDMs without a good, hierarchical classification that separates fundamental or “essential” features from related details.

The principal purpose of this paper is to propose such a classification that would allow academics and practitioners alike to keep up with the continuing growth of the “methodology jungle”. Our proposal is based on distinguishing “essential” from “accidental” features – metaphorically applying Brooks’ [4] distinction between ‘essences’ and ‘accidents’ of software products. It is hypothesized that the number of “essential” features by which ISDMs can differ is relatively small. As was noted in [11], a complete ISDM necessarily includes several accidental features. While these are important for completeness, in practical application, they do not define its essence. Hence the choice between fundamentally different, i.e. “contrasting” ISDMs can be based on a more parsimonious catalogue of features.

The organization of the paper is as follows. Section 2 summarizes the basic ideas upon which the arguments of this paper build. Of critical importance in this section is the distinction between information systems development
 approaches (ISDAs) and ISDMs. Section 2 illustrates this distinction with a framework that provides a conceptual roof for the spectrum of currently known approaches. The classification introduced in section 2 is, however, “static” and would therefore become quickly obsolete. Hence section 3 proposes a procedure on how to insert a new ISDM in the classification structure. Section 4 offers two examples of how to use the procedure. Section 5 reflects upon the implications of the ideas proposed in this paper.

2. Literature review and background

2.1 The need for ISDM classification

The proliferation of ISDMs has led to the need to analyze, compare and reflect upon alternative ISDMs [34]. At least two principal lines of research can be distinguished in the previous work on ISDM comparisons. One has mainly focused on a so-called ‘feature analysis’ (e.g. [35]; [7]; [9]; [23], [27]; [8]). Another, more recent research approach, uses formal meta-modeling (e.g. [14]; [31]). The feature analyses research stream has varied using a more or less arbitrary set of features (e.g. [36], [3]; [24]) to more theory-based frameworks (e.g. [35]; [22]) and reference models for an information system (e.g. [26]; [16]) or the ISD process (e.g. [20]; [21]). One of the most ambitious attempts to systematically compare information systems and software development methods is offered by [34]. We shall refer to their work more explicitly below.

The key problem with this prior work is that most of the existing comparisons have taken place at the level of specific ISDM, because it focussed on whole methodologies as the unit of analysis. In distinction to this, Fitzgerald’s study [11] of how seasoned practitioners benefit from knowing an ISDM suggests that they “frame it at a higher level of granularity”. He concludes “that the multiplicity of manuals which accompany many methodologies and prescribe in a very detailed fashion the exact processes in the ISDA. They help to ensure that the ISDM instances of a particular approach form coherent wholes.” (p. 207) Based on other, more theoretical work there is reason to believe that this study has revealed a very typical phenomenon of how experienced experts “recode” the details of specific ISDMs into larger “chunks” of knowledge. They are thereby able to remember relevant methodological principles and use them effectively. The most fundamental implication of this insight is to shift the discussion from alternative ISDMs and tools as a unit of analysis to a higher level of granularity, at which the IS community’s thinking can benefit from comparing and contrasting features at the level of ISDAs rather than ISDMs.

To this end this paper proposes groupings of ISDMs, called “approaches” (ISDAs), as a more explicit and central concept to structure the field. It has been common to implicitly group ISDMs by their similarity into families or clusters. Examples of such groupings are the family of structured ISDMs and object-oriented (OO) analysis and design ISDMs. The tenet of the paper is that one should move beyond strict ISDMs to focus on more general ISDAs (“A” as in “Approach”), which may comprise a number of more specific ISDMs (M as in “Methodology”) as their instances. These general ISDAs may be conceived of as prototypical classes that share a number of common features with their member ISDMs. The general ISDAs defining the essences of their member ISDMs may be conceived of as templates which help to generate more detailed ISDMs “on the fly”, possibly combining existing “ISDM fragments” [13]. The key issue then is how to identify the most general features of ISDMs that can be used to group them into “approaches”. The benefits of this can be illustrated with some typical numbers. One can estimate that current ISDMs of the order 1000 can be classified into a more comprehensible number (perhaps 20 to 30) of ISDAs – a reduction of 30 to 50 times. In this way, ISDAs provide condensed forms of knowledge representation for alternative ways of conducting systems development. They can facilitate comparisons between ISDMs by directly focussing on essential features rather than accidental ones.

2.2. A framework for ISDM classification

Initially, the proposed framework will have two levels: At the bottom are concrete ISDMs (as may be documented in manuals or texts like [38]) that at the next higher level are related to approaches. As will become clear in sections 3 and 4, more intermediate levels can be constructed as needed to keep the classification up to date. An ISD approach (ISDA) is interpreted as a class of specific ISDMs sharing a set of related features. In defining the features, Song and Osterweil [34] were a source of inspiration, in particular with their notion of ‘concept’ in their “type hierarchy”. They define a ‘concept’ as an idea that influences the design of a software development method, and distinguish concepts such as ‘problem’, ‘principle’, ‘guideline’, ‘criterion’ and ‘measure’. In building on these ideas, we identified four features. They seemed to be identifiable in all ISDMs that we studied (for illustrations of these features see table 1). Therefore we took them to be essential building blocks of ISDAs, because they drive interpretations and actions in IS development. An ISDA’s goal specifies its general purpose. Guiding principles and beliefs form the common “philosophy” (cf. [1]) of the ISDA. They help to ensure that the ISDM instances of a specific approach form coherent wholes. Fundamental concepts largely define the nature of an IS implicit in the approach, the focus and unit of analysis in ISD. Principles of the ISD process express essential aspects of the ISD process in the ISDA.
Table 1, illustrates the features of an ISDA further. In col. 1, “to provide maintainable software” is an example of a goal related to SA/SD [38]; “the design of modules with high cohesion and weak coupling”, one of its guiding principles. “Data flow” and “cohesion” are two of its fundamental concepts; finally, “to apply a situation-dependent process model” is one of its principles of the ISD process. The framework proposed by Song and Osterweil [34] helps to illustrate those definitions further by noting similarities and differences. They provide examples such as to produce change able programs by noting similarities and differences. They provide Osterweil [34] helps to illustrate those definitions further.

ISDMs as its instances. As an example of the first case (i.e. zero ISDM instances), consider the interactionist approach (cf. [19]), which to our knowledge has not been developed into any specific ISDM. On the other hand, the OO approach comprises a number of ISDM instances. Our claim is that the concept of an ISDA makes it meaningful to compare various approaches, which may be in quite different stages of their development in terms of the number of their ISDM.

This line of reasoning leads to a hierarchical structure - our framework -- depicted in Figure 1. At the bottom, we identify examples of specific ISDMs, which include detailed concepts, notations, activities, and techniques, which may be accidental features (such as notations) or idiosyncratic to the specific ISDM (e.g. the concept of qualifier in [32], among the OO methodologies). The top of Figure 1 describes eleven ISDAs. Table 1 summarizes five of these eleven in more detail (see [18], for a more detailed introduction). The framework includes an inheritance structure in which each ISDM inherits the features of its ISDA (goals, guiding principles and beliefs, fundamental concepts, and principles of the ISD process).1

Our interpretation of the five ISDAs is not necessarily unambiguous. It would, of course, be preferable if the developers of the ISDMs had themselves articulated the essences of the ISDA and ISDM they proposed. Because this was not the case, we were obliged to rely on our own interpretation and judgement. One should also note that one could have a more refined analysis, distinguishing sub-approaches within the main approaches listed in fig. 1. Perhaps this would have allowed for a more unambiguous interpretation. However, this is beyond the scope of this paper (see for example [15] and [30], for reviews of different semantic information/data models underlying the IM approach, and [25], and [12], for a review of the STD approach).

Because the stock of approaches and methodologies is continuously changing, the proposed classification structure would become irrelevant in a short time unless it can easily be updated. The reason for this is that an ISDM may represent a new type of ISDA that is not yet represented in the classification structure. Hence the classification has to be made “dynamic” in the following sense: An ‘insertion’ of a new ISDM may prompt a generalization of existing classes. The main purpose of this paper is to propose a procedure for doing this. The procedure will assure that any new methodology M can be either be "assimilated" by associating it with an existing class (approach) or "accommodated" by redefining the classification structure in such a way that M will fit.

3. A procedure for classifying ISDMs

1 Observe that inheritance in our context does not only concern ISDA’s attributes (goals, guiding principles and beliefs, fundamental concepts and principles of the ISD process) but also their values (i.e. the specific positions an approach takes with regard to the above attributes).
3.1 An outline of the procedure

The basic idea of the following procedure is to identify those ISDAs of which the new ISDM is an instance (case 1) or which underlie the new ISDA or ISDM (case 2). In case 2, an ISDM may represent a new ISDA that is not yet represented in the classification structure or the 'inser
whether it fits under an existing class (case of assimilation). This judgment would be based on analyzing the correspondence of the features of the new M and the existing classes. If that fails, then one needs to check if an existing class can somehow be modified or generalized to assimilate M. This can be done by selecting probable candidate classes and generalizing them by deleting or modifying some of their features. Again the features of the generalized or modified class must correspond to the features of M.

If the checks so far do not succeed at accommodating M (or the new approach), then a new class with multiple inheritance has to be formed (for details see pass 3, steps 12-21 below). In this case the ISDM must be abstracted into a separate approach (step 12). After that the procedure follows the logic of the single hierarchy as described above. First, the procedure (steps 14-15) checks, whether any of the existing candidate classes shares a strict subset of the features that the new class abstracted from M. If it does, then the new class is inserted as its subclass. Next the procedure checks whether any of the candidate classes can be generalized (steps 16-18) or modified (steps 21) so that the resultant class shares a strict subset of features of the new class formed from M. If so, the new class formed from M is inserted as its subset. This pass ends when all the features of the ISDM (or the new class formed from it) are inherited in the resultant class structure (steps 15, 18 and 21) or modified (steps 19-21) or the set of candidate classes has been exhausted (step 13).

Two cases have to be distinguished. In case A selecting the relevant features from the existing classes can form a new class. In this case the new class formed from M forms an 'intersection' approach which inherits all its features from its superclasses. It is unique, however, in the sense that none of its superclasses includes all its features. Case B is that the new M proposes some features that are not yet contained in any existing class. In this case the features will remain in the class formed from M.

### 3.2 Semi-formal specification

The following procedure INSERT(M) describes more precisely the classification process for the insertion of a specific methodology M (or IDSA) in the existing classification structure than the general outline presented in section 3.1.

1. Insert M as an instance of the class ‘ISD methodologies’. Analyze the features \( F = F_M \) i.e. goals, guiding principles, fundamental concepts and principles of the ISD process underlying M.

2. Consider whether there are candidate classes \( C = \{C\} \) for M.\(^3\) If ‘no’ form a class \( C' \) with M as its instance.\(^4\) Insert \( C' \) as an instance of the class ‘ISD approaches’ and go to step 23.

3. Select a candidate class \( C \in C \). If all candidate classes analyzed, go to step 5.

4. Analyze whether the features of C and those underlying M are equivalent, i.e. \( F(C) = F(M) \). If ‘yes’ insert M an instance of C, insert C as an instance of ISD approaches (if not already done) and go step 24. If ‘no’ go to step 3.

5. Select a candidate class \( C \in C \). If all candidate classes analyzed, go to step 12.

6. Consider whether C can be generalized to a class \( C' \) so that \( C' \) includes a strict subset of immediate and inherited features of C (i.e. \( F' = F(C') \subset F(C) \)) and that the features \( C' \) are equivalent to features underlying M, i.e. \( F(C') = F(M) \). If ‘no’, go to step 9.

7. Form a class \( C'' \) with a subclass C. Associate the subset \( F' \) of (immediate and inherited) features of \( C (F' \subset F(C)) \) to be generalized with \( C' \). The features of \( C' \) not to be generalized remain in C (i.e. \( F(C) = F(C') - F' \)). Insert \( C'' \) as a subclass of all the classes of which C was a subclass. Delete from the C’ those features that it inherits from its superclasses. Insert M as an instance of \( C' \).

8. If \( C' \) inherits features not-to-be-generalized (i.e. \( F \in F(C) \)), search the class \( C* \) from which the feature was inherited. Do \( CLEAN(C*, F) \). Repeat step 8 until no \( F \in F(C) \) is inherited by \( C' \). If any of the not-to-be-generalized features is inserted in \( C' \), delete them. Insert \( C' \) as an instance of ISD approaches. Go to step 22.

9. Consider whether a subset \( F* \subset F(C) \) of the immediate and inherited features of C can be modified to features \( F' = \{F'\} \) so that \( C \) and \( C''' \), in which \( C''' \) includes the modified features \( F' \), are subclasses of \( C' \), and the features of \( C''' \) and M are equivalent (i.e. \( F(C''') = F(M) \)). If ‘no’, go to step 5.

10. Associate the not-to-be-modified features of C with \( C' \) (i.e. \( F(C') = F(C) - F' \)). The to-be-modified features remain in \( C \) (i.e. \( F(C) = F' \)). Associate the modified features with class \( C''' \) (i.e. \( F(C''') = F' \)). Insert C and \( C''' \) as subclasses of \( C' \). Insert \( C' \) as a subclass of all the classes of which C was a subclass.\(^3\) I.e. classes the features of which are close to those underlying M. The features include both inherited and immediate features of the class in question.

\(^4\) \( C'' \) with M as its instance shares the goals, guiding principles, fundamental concepts and principles of the ISD underlying M.
Delete from the C’ those features that it inherits from its superclasses. Insert M as an instance of C’’.

11. If any of the to-be-modified features (i.e. F ∈ F(C) - F*) is inherited by C’, search the class C* from which the feature was inherited. Do CLEAN(C*,F). Repeat step 11 until no modified features are inherited by C’. If any of the to-be-modified features is inserted in C’, delete them. Insert C’ as an instance of ISD approaches. Go to step 22.

M and multiple inheritance

12. Form a class C’’ with M as its instance. Associate all M and multiple inheritance approaches. Go to step 22.

13. Select a candidate class C ∈ C. If all candidate classes analyzed, go to step 22.

14. Analyze whether C shares a subset of features of C’’, i.e. F(C) ⊂ F(C’’). If ‘yes’ insert C’’ as a subclass of C. If ‘no’ go to step 16.

15. If all the features of C’’ are inherited in the class structure, go to step 22. Otherwise go to step 13.

16. Consider whether C’ can be generalized to a class C so that C’ includes a strict subset of immediate and inherited features of C (i.e. F(C’) ⊂ F(C)). Insert C’ as a subclass of all the classes of which C was a subclass. Delete from the C’ those features that it inherits from its superclasses. Insert C’’ as a subclass of C’.

17. Form a class C with a subclass C. Associate the subset F’ of (immediate and inherited) features of C (F’ ⊂ F(C)) to be generalized with C’. The features of C not to be generalized remain in C (i.e. F(C) = F(C) - F’). Insert C’ as a subclass of all the classes of which C was a subclass. Delete from the C’ those features that it inherits from its superclasses. Insert C’ as a subclass of C’.

18. If C’ inherits features not-to-be-generalized (i.e. F ∈ F(C)), search the class C* from which the feature was inherited. Do CLEAN(C*,F). Repeat step 18 until no F ∈ F(C) is inherited by C’. If any of the not-to-be-generalized features is inserted in C’, delete them. If all the features of C’’ are inherited in the class structure, go to step 22. Otherwise go to step 13.

19. Consider whether a subset F* ⊆ F(C) of the immediate and inherited features of C can be modified to features F’= {F’} so that C and C’’’, in which C includes to-be-modified features F’ and C’’’, includes the modified features F’, are subclasses of C, and C’’’ shares a subset of features of C’’ (i.e. F(C’’’) ⊂ F(C’’)). If ‘no’, go to step 13.

20. Associate the not-to-be-modified features of C with C’ (i.e. F(C’’’) = F(C’’’) - F*). The to-be-modified features remain in C (i.e. F(C) = F*). Associate the modified features with class C’’’ (i.e. F(C’’’)) = F’). Insert C and C’’’ as subclasses of C’. Insert C’ as a subclass of all the classes of which C was a subclass.

21. If any of the to-be-modified features (i.e. F ∈ F(C) = F*) is inherited by C’, search the class C* from which the feature was inherited. Do CLEAN(C*,F). Repeat step 21 until no modified features are inherited by C’.

22. Consider which of the new classes C’ and C’’ are meaningful as potential ISD approaches. Insert them as instances of the class ‘ISD approaches’ (if not already done).

23. The classification has been performed successfully.

Procedure CLEAN(C,F):

1. Delete the feature F from C and insert it to all subclasses of C.

2. If C becomes empty (i.e. does not include any immediate features), insert all its subclasses as subclasses of all its superclasses and delete C.

3. If C had instance methodologies insert them using the procedure INSERT(M).

Even though the above procedure INSERT looks like an computer algorithm, its execution requires human judgment, especially step 1, analysis of the features (i.e. goals, guiding principles, fundamental goals and principles of the ISD process) underlying the ISDM to be inserted, and step 2, selection of the candidate classes. Step 2 does not state criteria for the selection of the candidate classes but obviously genealogical dependencies of ISDMs should be considered in addition to sharing of common features. Also steps 6, 9, 16 and 19 include significant elements of human judgment. Finally, step 22 is essentially human judgment.

The above procedure includes three “passes”. Pass 1 (steps 3 and 4) checks whether a new ISDM is an instance of any of the candidate classes. If it is, it is simply inserted as an instance of the class in question. Pass 2 (steps 5-11) checks whether an ISDM can be considered an instance of some class when an existing candidate class is generalized by deleting or modifying some of its features. Pass 3 (steps 12-21) deals with the situation of multiple inheritance. In this case the ISDM must be abstracted into a new approach (step 12). The third pass ends when all the features of the ISDM (or the new class formed from it) are inherited in the resultant class structure (steps 15, 18 and 21) or the set of candidate classes have been analyzed (step 13).

The application of the above procedure may include renaming of existing classes to better capture the new classification structure. This aspect will be illustrated below when the procedure is applied.
4. Two examples

To illustrate the procedure, we shall insert Object Modeling Technique (OMT [32]) and Multiview [2] into the framework of Figure 1. OMT is used to illustrate the re-vision of the class hierarchy in the case of the object-oriented approach. Multiview on the other hand illustrates multiple inheritance.

4.1 OMT as an OO approach

To classify OMT [32] requires a careful analysis of its goals, guiding principles, fundamental concepts and principles of the ISD process (step 1). Even though OMT has influences from ‘Information modelling’, the ‘SA/SD approach’ and the ‘OO approach’, we will treat the last one as a relevant candidate class in its case (step 2).

Let us assume that the ‘OO approach’ considers features listed in Table 1 as fundamental. An essential feature of OMT is that it does not adhere to encapsulation in the OO analysis ([33], [17]). Let us call its encapsulation principle “weak encapsulation”. Therefore OMT cannot be considered an instance of the ‘OO approach’ as interpreted in Table 1, assuming that encapsulation refers to “strict encapsulation”, i.e. encapsulation throughout the OO development process (step 4).

The next question is whether OMT could be considered an instance of the class resulting from the deletion of some features of the class ‘OO approach’ (steps 5 and 6). The possible feature to be deleted would be “strict encapsulation”. Because OMT adheres to “weak encapsulation”, the total deletion of the encapsulation principle does not work in this case. So we can skip steps 7 and 8.

Let us analyze whether OMT can be inserted as an instance of the class ‘OO approach’ (steps 5 and 6). The possible feature to be deleted would be “strict encapsulation”. Because OMT adheres to “weak encapsulation”, the total deletion of the encapsulation principle does not work in this case. So we can skip steps 7 and 8.

4.2 Multiview and multiple inheritance

Multiview [2] is an ISDM that explicitly attempts to reconcile ideas from several ISDAs, most notably SSM, sociotechnical design, structured analysis and information modelling. Therefore it provides an excellent case to illustrate multiple inheritance.

Table 2 summarizes the characterizing features underlying Multiview. The characterizations in Table 2 follow closely the vocabulary of Multiview in order to preserve the authenticity of its interpretation.

<table>
<thead>
<tr>
<th>Goals</th>
<th>Multiview provides a methodology for exploration into IS development (i: SSM).</th>
<th>More specifically, it helps in providing answers to the following questions:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. How is the IS supposed to further the aims of the organization? (i: SSM)</td>
<td>2. How can it be fitted into the working lives of the people in the organization who are going to use it? (i: STD)</td>
</tr>
<tr>
<td></td>
<td>3. How can the individuals concerned best relate to the computer in terms of operating it and using the output from it?</td>
<td>4. What information processing function is the system to perform?</td>
</tr>
<tr>
<td></td>
<td>5. What is the technical specification of a system that will come close enough to doing the things that you have written down in the answers to the other four questions?</td>
<td></td>
</tr>
</tbody>
</table>

Guiding principles and beliefs

To develop an information system, which is complete in both technical and human terms, requires multiple viewpoints comprising the viewpoints of human activity, information analysis, sociotechnical aspects, human-computer interface and technical design.

The multiple viewpoints should be combined in a reasonably coherent “methodology framework”

Analysis of human activity system: To search for a particular worldview, Weltanschauung, to form the basis for describing system requirements (i: SSM)
Information analysis: To analyze the entities and functions of the system, independent of how the system will eventually develop

Analysis and design of the sociotechnical aspects: To produce a ‘good fit’ design, taking into account people and their needs and the working environment on the one hand, and the organizational structure, computer systems and necessary work tasks on the other (i: STD)

Design of the human-computer interface: The dialogue should be related to who will be using the system.

Design of technical aspects.

Fundamental concepts

- Analysis of Human Activity System (Human Activity System, Weltanschauung, root definition, relevant system, conceptual model (i: SSM), rich picture)
- Information analysis: Functional analysis (function, event, dataflow)
- Information analysis: Data analysis (entity, relationship, attribute)
- Socio-technical design (technical objectives, social objectives, technical alternatives, social alternatives (i: STD))
- Human-computer dialogue
- Technical aspects (processing application, information retrieval, database maintenance, control, recovery, monitoring)

Principles of the ISD process

- Flexibility of the process within Multiview
- Five stages of IS analysis and design
- Analysis of human activity system: A modification of the seven-stage model of [5]
- Information analysis: Top-down decomposition of functions based on the primary task conceptual model; construction of data flow models; verification of functional and entity models
- Analysis and design of the sociotechnical aspects: Sociotechnical design, user participation (i: STD)

One can easily identify SSM and STD as clear candidate classes in the sense of step 2 of the above procedure. A rudimentary comparison of the features of the IM approach (Table 1) with Table 2 suggests that the latter does not form a reasonable candidate class for Multiview. Multiview has only adopted a few specific techniques (such as the ER model) from IM without other features. The relationship with the SA/SD approach is more complicated. Multiview has clearly adopted dataflow diagrams. Although not stated explicitly, one could also insist that it implicitly adheres to the goals and most of the guiding principles of the SA/SD approach. Despite this possibility, we will focus on only two candidate classes, SSM and STD.

Referring to steps 3 and 4 of the above procedure, it is clear that Multiview cannot be inserted as an instance of either of the two candidate classes. Neither can the candidate classes be generalized by removing features of the original class nor their features modified in such a way that Multiview could be regarded as an instance of the generalized or modified class (step 5 and step 9). So, let us proceed to step 12 of the above procedure.

Step 12 implies that Multiview forms an ISDA, the features of which have been summarized in Table 3. Let us analyze SSM first as a candidate class (step 13). It is clear that the features of SSM as listed in Table 2 do not form a subset of features of Multiview in Table 3 (in particular the stream of cultural analysis is missing in Multiview). Generalization of SSM by removing the stream of cultural analysis (step 16) leads to a class of features which form a subset of the features of Multiview. Let us name the new class ‘SSM-core’ and the original class ‘SSM-1990’.

‘SSM-1990’ forms a subclass of ‘SSM-core’, inheriting all the features of ‘SSM-core’. The only non-inherited feature of ‘SSM-1990’ is the stream of cultural analysis. Let us insert Multiview as an ISDA a subclass of ‘SSM-core’.

Let us return to STD as a candidate class. According to our interpretation Multiview does not specifically emphasize guiding principles such as minimal critical specification and open-ended design, concepts such as variance and unit operation and the evolutionary nature of the ISD process. This leads again in step 16 to a generalization in which a class ‘STD-reduced’ is formed. It includes all the features of the original STD approach except the features mentioned above. The original STD approach, renamed ‘STD-complete’ includes the omitted features and as a subset of the class ‘STD-reduced’ inherits all its features. The STD ideas of Multiview can be interpreted to cover the features of the ‘STD-reduced’. Therefore Multiview can be inserted as a subset of ‘STD-reduced’.

When finalizing, there is the question whether the new classes, ‘Multiview’, ‘SSM-core’ and ‘STD-reduced’ can be considered reasonable ISDAs (step 22). Because ‘SSM-core’ corresponds to the version of SSM by [5], it can be regarded as an ISDA. To our knowledge, ‘STD-reduced’ does not have any instance methodologies. Therefore, let us take a position that it does not form an ISDA of itself. To summarize, the above procedure leads us to consider Multiview as an ISDA that inherits from ‘SSM-core’ and ‘STD-reduced’. The inherited features of Multiview are indicated by i:SSM and i:STD in Table 2, respectively. Additionally, Multiview includes a number of other features, e.g. the modification of the 7-stage model of [5].
and rich pictures from the SSM origin. Neither of these was interpreted as essential features of SSM in section 3.2.

The above example indicates that insertion of a new ISDM does not necessarily lead to the most “elegant” class structure. The resultant class structure would have been more elegant, had the initial structure more faithfully recorded the genealogy of SSM as an approach. In fact, one could have identified in section 2.2 two versions (sub-approaches) of SSM: ‘SSM-1981’ and ‘SSM-1990’, corresponding to the two books ([5],[6]) of Checkland. The latter corresponds to a class with the same name above. ‘SSM-1981’ would have included the seven-stage process model as one of its principles of the ISD process. Both these would be subclasses of ‘SSM-core’ as defined above. In this situation Multiview could have been inserted directly as a subclass of ‘SSM-1981’.

5. Conclusions

In essence, we have provided concepts for moving the discussion from ISDMs as principal units of analysis to a higher level of granularity. The number of levels is flexible and easily adjusts if progress is reflected in higher levels of abstraction for discussing the “features” of approaches. This is the key point in which this paper goes beyond earlier work published. Keeping the hierarchical feature inheritance structure “dynamic” and thereby up to date, should help to focus any future discussion on the essential rather than accidental features of ISDMs and ISDAs. To illustrate these ideas that paper introduces two exemplars on how to identify and summarize ISDA features. Of course, as any language evolves, so the representation scheme may be extended in future work.

The first and more obvious implication of the paper is that it organizes the field of methods and tools of ISD in a more comprehensible way through the development of a structure that distinguishes ISDMs and ISDAs. In this way, hundreds of ISDMs can be reduced to a few dozens ISDAs characterized by their essential features. The reduction does not go at the expense of comprehensiveness or accuracy, because the underlying categories are “leveled” (allowing for the representation of all details if needed at the bottom level) and can easily be restructured to reflect new principles and tools of ISD to reflect new knowledge.

This brings up a more subtle implication, which relates to language and knowledge dissemination. As noted above, existing research on ISDM use in practice has consistently found that ISDMs, as far as they are used, are applied adaptively customizing the ISDM to fit the organization and project in question. However, when ISD is taught, this is not reflected in the way texts and teachers “represent” the principles of ISD. Typically the focus is on the specific details of one or two “widely practiced” ISDMs such as structured analysis or a specific object oriented methodology. We think that a major obstacle for overcoming this has been the lack of a suitable language that affords to talk about principles of ISD in general without resorting to motherhood statements and trivialities. The terms in this paper constitute a proposal for such a language, a starting point from which seasoned analysts (and text book writers) can take off to develop a new vocabulary that encourages novices and their trainers to separate essentials from details and accidentals. The dynamic version of the proposed framework allows us to continually capture the expert knowledge and insights embedded in the continually emerging ISDM base. We therefore should think of it as a dynamic knowledge representation scheme. Currently, the collective knowledge about ISD, which has been built since the late fifties with the emergence of the first life cycle methodologies, remains elusive and difficult to assimilate because it is too widely dispersed in hundreds of ISDMs.

Hence the condensed presentations of ISDAs at the core of this paper can be brought to bear on specific IS development project situations. In addition to the intellectual contributions, this has also practical implications because, at best, only a few ISDMs are likely to be mastered in practice by any one individual. The simplifying concepts of an ISDA may allow practitioners to widen their “methodology” repertoire.

References