SADT/IDEF0 for Augmenting UML, Agile and Usability Engineering Methods

David A. Marca

The University of Phoenix, Online School College of Information Systems and Technology, 3157 East Elwood Street, Phoenix, Arizona 85034, U.S.A.
dmarca@email.phoenix.edu

Abstract. Many experts state that: a) specifying "all the small parts of a system" and b) correct expected system usage, can make Agile Software Development more effective. Unified Modeling Method (UML) addresses the former; Usability Engineering addresses the later. Taken together, they create a systems de-velopment framework, capable of: a) specifying functions, data, behavior and usage, b) rapid prototyping, and c) verifying system usability and correctness. All three of these methods focus first on the system, while secondarily trying to ascertain system context. Correct and complete context requires domain modeling. Structured Analysis and Design Technique (SADT/IDEF0) is a proven way to model any kind of domain. Its power and rigor come from: a) a synthesis of graphics, natural language, hierarchical decomposition, and relative context coding, b) distinguishing controls from transformations, c) function activation rules, and d) heuristics for managing model complexity. This paper explains how SADT/IDEF0 domain modeling can bring correct and complete context, to today's commonplace disciplines of the Unified Modeling Language (UML), Agile System Development, and Usability Engineering methods.

Keywords: Domain Modeling, General Systems Theory, UML, Agile Development, Usability Engineering, SADT, IDEF0, Domain Driven Design.

1 Introduction

Commercial software engineering disciplines have come a very long way since their post World War II origins. Three of the more commonplace disciplines of today are: a) Unified Modeling Language – UML [5], Agile Software Development [2], and Usability Engineering [29]. When used in combination, these methods have a strong track record for developing software for many kinds of problems and domains. Over the last ten years, a large amount of research has been done on the shortcomings of these methods, and a collection of this research is presented in this paper. Taken as a whole, this research suggests that many shortcomings arise because domain modeling is not at the core of these methods. Therefore, one way to bolster today's common-place software development methods is to augment with a proven domain modeling method, such as SADT/IDEF0. Domain modeling is at the core of SADT/IDEF0, and when properly used, the method can produces holistic domain models that can address any level of complexity or abstraction. To explain:

M.J. Escalona, J. Cordeiro, and B. Shishkov (Eds.): ICSOFT 2011, CCIS 303, pp. 38–55, 2013.

1.1 Domain Modeling Is Not the Core of Current Methods

The first, and very important, aspect about modeling with today's commonplace methods is that UML [5], Agile Software Development [2], and Usability Engineering [29] have their origins rooted in software systems. In other words, their focus is on the software system. Thus, their principles, languages and practices were invented for creating software. While each includes a component for domain modeling, that component is not at the core of the method. For example: UML's core is software system specification, Agile's core is rapid software deployment, and UE's core is evaluation of the software system during its use. For these disciplines, domain modeling is just a first step to getting to the core work.

1.2 Domain Modeling Is at the Core of SADT/IDEF0

In contrast, SADT/IDEF0 is rooted in general systems theory [35]. Its focus is any kind of system. Interestingly, when it was first introduced, many in the commercial world confused it for being a method that could just describe either software systems or manufacturing processes. While it can describe these two kinds of systems, its strength is its focus on systems in general. Therefore, it has unique principles, simple language, and special practices for describing any real-world phenomenon – domain modeling is its core! When used correctly, SADT/IDEF0 can produce a set of very concise, small models, with tightly connected context and content. This paper will illuminate the often misunderstood potential of SADT/IDEF0 as a contributor to, and not a replacement for, today's software development methods (see Figure 1).

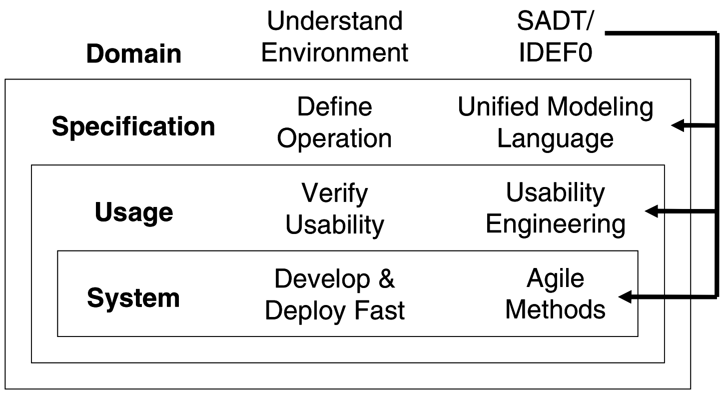


Fig. 1. Where SADT/IDEF0 augments UML, agile and usability engineering methods

1.3 The Use of SADT/IDEF0 Produces Holistic Domain Models

The distinguishing, unique aspect of SADT/IDEF0 is its ability to holistically describe an entire domain to any desired low level of detail, and to describe its context to any desired high level of abstraction. It is thus a story-telling discipline with very rigorous engineering syntax (i.e. boxes and arrows) and semantics (e.g. box sides for Input, Control, Output, and Mechanism – ICOM – and corresponding implicit off-page connectors), plus heuristics for managing model complexity (e.g. hierarchic decomposition, 3 to 6 boxes per decomposition, single purpose and viewpoint per model, model "call" syntax). For example, it distinguishes controls from inputs, and advocates stopping model decomposition when the model's purpose has been fulfilled. It is the aspect of simple, concise, complete, context-rich, holistic description that is the primary contribution of SADT/IDEF0 to the other aforementioned methods [25].

1.4 SADT/IDEF0 Can Address any Level of Complexity or Abstraction

The statement often arises: "My method is effective at domain modeling, so I do not need another method." My response: "yes and no," and here is why: For very simple domains or systems, it is easy to "get one's head around the problem." Thus, no other method is needed to understand the system's immediate context. However, for very complex problems (e.g. enterprise-wide solutions, large weapons such as a submarine, aircraft sheet metal fabrication), no single software developer can understand the whole problem. At the extreme case (e.g. well log interpretation, disaster recovery, decision making, strategy formulation), no domain expert may know or be able to articulate consistently and accurately, the entire domain. Such situations require a context map [52] to documented an understanding of the domain that must then drive the solution design [54] [53]. SADT/IDEF0 has an extremely simple graphic language and a model creation technique that, from the same starting point of any particular subject, can describe: a) all details (i.e. decompose complexity), b) the context of that subject (i.e. context modeling).

2 Why Consider SADT/IDEF0?

Since the 1970's, SADT/IDEF0 has been used to successfully describe a vast number and variety of domains. The reason for this success is best described by Doug Ross in his seminal paper [35]. To paraphrase: SADT/IDEF0 incorporates any other language; its scope is universal and unrestricted. It is concerned only with the orderly and well-structured decomposition of a subject. Model decomposition is sized to suit the modes of thinking and understanding of the viewpoint of the model and the intended audience of readers. Units of understanding (i.e. boxes and their data) are expressed in a way that rigorously, precisely and consistently represents domain interre-lationships. Decomposition is carried out to the required degree of depth, breadth, and scope while still maintaining all of the above properties. Thus, SADT/IDEF0 increases the quantity and quality of understanding that can be beyond the limitations inherently imposed by other kinds of natural or formal languages. Some details:

2.1 Vast Experience in a Wide Variety of Domains

SADT/IDEF0 has over 35 years of domain modeling experience, across a vast number of problems involving systems ranging from tiny to huge, in a wide variety of industries [24]. It has been used in commerce, government and military around the world. It has been used by small of privately held companies (e.g. 1-2 person start-ups) to some of the largest of largest organizations in the world (e.g. the U.S. Air Force), and on some of the largest initiatives (e.g. the U.S.A.F. Sheet Metal Fabrication Project [55]). This widespread success is due to its very strong set of domain modeling concepts, principles and features. A summary of these "features" is given in the Appendix of this paper, and is organized into two tables. Table 1 summarizes the box and arrow syntax and semantics, and the rules for how they interconnect. Table 2 completes Table 1 and summaries the reference language used to identify model elements. None of the aforementioned methods can claim the array of features, the richness of graphic semantics, the number of in-context supplements, or the longevity of success across so many industries and problems *for domain modeling*.

2.2 Strong Conceptual Underpinnings for Modeling

Since the 1970s, experts have agreed that deep understanding of the domain is vital for successful and effective software engineering [44]. The conceptual underpinnings of SADT/IDEF0 continue to be cited as being very strong for domain modeling. Most notably: a) tightly managed multiple views at the architecture level for complex systems [19], b) support for aspect-oriented modeling by being able to modularize cross-cutting concerns [17], c) defining boundaries essential for specifying objects, system scope, human-computer interaction [40], d) hierarchical exposition of detail for very large domains without loss of context and without making errors when going to next levels of detail/abstraction [31], e) specifying strong versus weak influence that each datum has on its functions [40], and f) "calls" (i.e. just like a software subroutine call) a model from another model to maximize reuse [20].

2.3 SADT/IDEF0 Features Are for Domain Modeling

Back in the 1970s, we did not have the universally understood notion of "ontology" as we know it today in the software engineering field. Nonetheless, since 1977, SADT™ has had a complete ontology for domain modeling [38]. Back then, the components of the ontology were called "features." Three of its core features are: context, model, and viewpoint [35]. With these three features, a core modeling principle was constructed: one model = one subject, described from one viewpoint [18]. This is, in effect, what we commonly call today the "system boundary." This demarcation point allows SADT/IDEF0 to consider a domain to be the whole context within which a system operates (e.g. the enterprise for a financial system as well as the business environment around that enterprise, the submarine for a defensive weapon system, the building for a thermostat control system). Also, SADT/IDEF0 has a simple box-and-arrow graphic language with associated semantics that make it ideal for capturing domain knowledge and reviewing it with end-users [10].

2.4 Preservation of Context

Probably the most important aspects of a domain modeling method are: a) simple syntax within which domain-specific language can be embedded, b) powerful semantics for representing the various roles information play in a domain, c) rigorous decomposition rules to support the detailing of highly complex subjects as well as the abstraction (to any level) of the context around any given system boundary, and d) consistent subject boundary management so that no context is lost when you move "into" the details of the subject. Together, these aspects provide a means by which context is always preserved. Figure 2 provides an example of context preservation. Notice how the data (arrow) inputs (I), controls (C), outputs (O), and mechanisms (M) that cross the functional boundary (box) are tied directly to the arrows on the diagram that details the function through the use of ICOM coding. With ICOM codes, you can never loose your way when you decompose a subject or create an abstraction that represents the context of a subject. Thus, since context preservation is crucial for domain modeling, SADT/IDEF0 has merit for augmenting the system development methods [25] such as the ones given earlier in this paper.

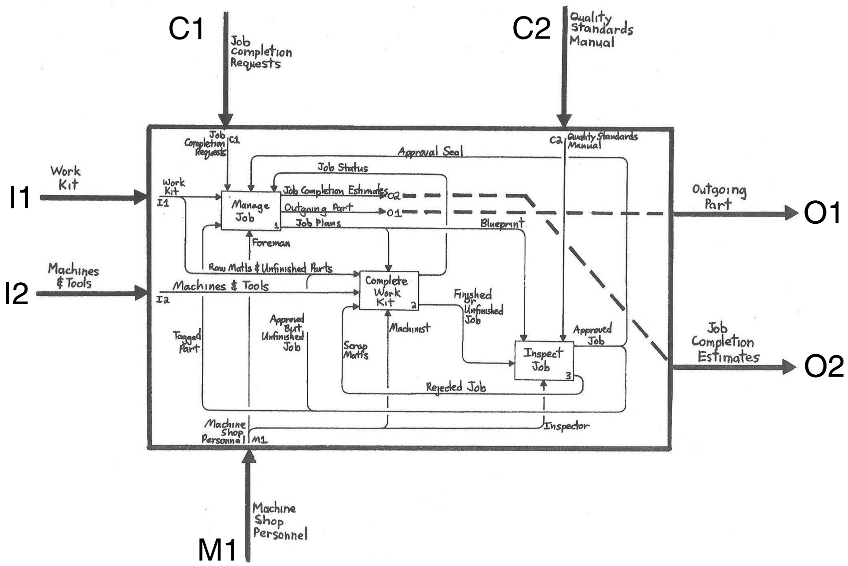


Fig. 2. An example of how SADT/IDEF0 models preserve context [24]

3 Augmentation Approach

The augmentation approach given in this paper is based on over 10 years of research by many practitioners and researchers. They point out particular shortcomings in the domain modeling portions of the aforementioned methods. They identified particular domain knowledge that, if it were available, could improve the results generated by the methods. So, given that particular domain knowledge required by UML, Agile and Usability Engineering methods. In short, correct, comprehensive and consistent specifications of domain knowledge are needed. Not only can SADT/IDEF0 correctly, comprehensively and consistently describe an entire domain – and not just the immediate context of a software system – it can describe that domain in rich and varied ways using carefully designed in-context supplements [38]. To explain:

3.1 Domain Knowledge Required by Other Methods

When practiced correctly, SADT/IDEF0 compresses a wealth of domain knowledge into a manageable set of small models. SADT/IDEF0 models and the special supplements created from the diagrams in those models, describe particular domain knowledge which the aforementioned methods depend upon: a) for UML: the system interface with its environment, decisions around manual versus automated function realization, functional scope, important objects in the domain, data dictionary, control data distinct from transactional data, overarching rules (often expressed as policies or doctrine), domain events and responses to those events (often called scenarios), and common versus special case scenarios; b) for Agile: same as for UML; and c) for Usability Engineering: the users' work, the context of that work, the tasks for accomplishing the work, the systems users need, and system usage scenarios.

3.2 Strong Specifications of Domain Knowledge

This paper briefly looks at some of the ontology of SADT (see Appendix), plus some additional features added after 1977, and explain how they can be used to augment the domain modeling portions of UML, Agile, and UE. Section 4 explains through figures and tables how SADT/IDEF0 models create stronger specifications of domain knowledge than the method it is augmenting. For example: a) knowledge that would have been missed by the other method, b) knowledge that would have been very hard to identify or describe by the other method, c) knowledge that needs to appear in all three methods that does not now do so, and d) how knowledge can be traced through all three methods. It is important to repeat that the domain modeling portions of the aforementioned methods are not bad; they just have shortcomings that over 10 years of practice have identified and documented (see References). SADT/IDEF0 can support the improvement recommendations in that documentation.

3.3 Knowledge Specification Using In-Context Supplements

The basic "unit of specification" of SADT/IDEF0 is the diagram, and a collection of diagrams comprises a model. However, SADT/IDEF0 has additional means by which domain knowledge is specified. To explain, the SADT/IDEF0 modeling process gives a person much more information than what is put on the basic diagram [24]. For example: a) terminology definitions, b) properties of functions and data, c) in-context narratives about the domain, d) particular situations (e.g. control flows, work flows) and special circumstances (e.g. mutually constraining functions) that occur in the domain, and e) rules by which functions activate and data must or must not interact with each other. Figure 3 gives an example of one basic diagram plus its supplemental pages, each identified with a letter corresponding to a-e above.

SADT/IDEF0 uses diagram supplements to capture this information, usually just after a basic diagram is approved by the domain experts that were interviewed by the systems analyst who authored the diagram. The supplements are: i) glossary page, ii) for exposition only (FEO) page, and iii) text page [38]. A glossary page defines terminology. A text page succinctly describes the operation of each box on the diagram. FEO pages contain closely related figures or pictures, or they annotate the basic diagram with: property labels, highlighted boxes and arrows, or box activation rules. Each supplement is derived directly from only its basic diagram, and thus these specifications of domain knowledge are always inside the context of one, well-bounded subject. Thus, SADT/IDEF0 supplements are fully consistent with each other.

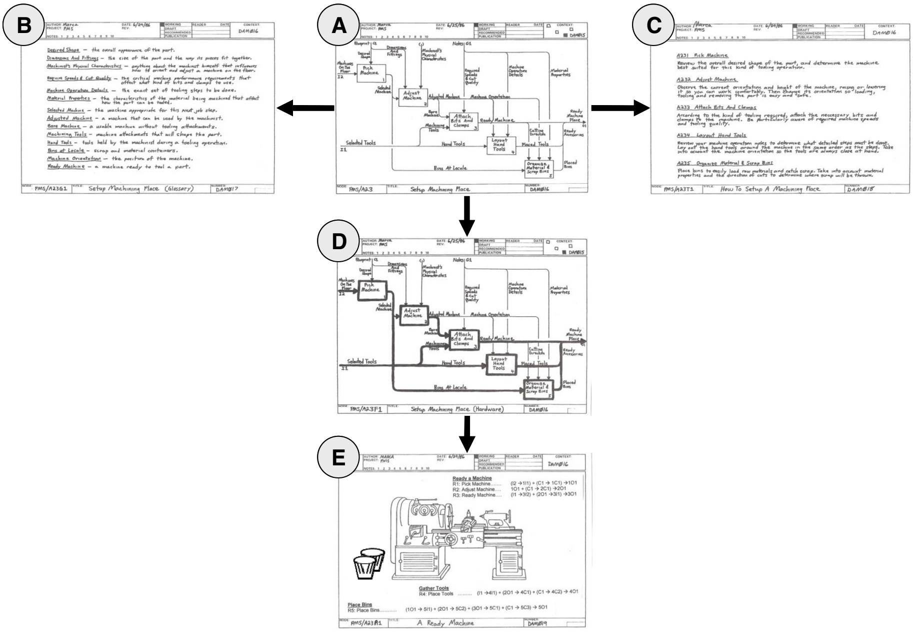


Fig. 3. A supplement set for a single SADT/IDEF0 diagram [24]

3.4 SADT/IDEF0 Ontology and Model Supplements Enable Augmentation

Figure 3 shows that the supplements developed directly from a single diagram comprise a rich specification of one bounded subject in the domain. Return to Table 1 and 2 in the Appendix, and identify all the ontology elements that go into these supplements: you will see the depth of SADT/IDEF0 for representing domain knowledge. As Section 2.4 says, the ontology is power enough for describing any system to any level of detail and any level of abstraction without loosing context. Thus:

Hypothesis 1: A set of SADT/IDEF0 diagrams and supplements that correctly and completely describe the domain in which a software system will operate, has content that is essential for augmenting the UML, Agile, and Usability Engineering methods.

Hypothesis 2: The content of those SADT/IDEF0 models and diagram supplements can be extracted and organized so that it can become useful input to the UML, Agile, and Usability Engineering methods, and without altering those methods.

4 Augmentations for UML, Agile and Usability Engineering

The proposed augmentation approach centers on comprehensive, correct and consistent specifications of domain knowledge. When used properly, SADT/IDEF0 can create such specification of an entire domain, not just the immediate context, for a software system. And it can describe that domain in rich and varied ways using in-context supplements which contain the: language, beliefs, assumptions, human organization, human work, work tasks and tools, and system usage expectations, that are vital to the successful application of UML, Agile, and Usability Engineering methods. This section summarizes shortcomings and corresponding improvement recommendations, based on over 10 years of experience with the aforementioned methods. The combination of shortcomings, recommendations, and the representational power of SADT/IDEF0 diagrams and supplements led to this approach.

4.1 Benefits to UML

UML Shortcomings. Experts have consistently noted that object-oriented code design methods are better at specifying software than they are at modeling domains [16]. For domain modeling, UML considers the domain to be the entities that touch the software system [27], and that is what UML "domain model" specify. The only other outward-facing UML model, the "business model," specifies how the software system will be used [56] [57]. These models can define a software system's boundary, provided they are complete and accurate. But assuring completeness and accuracy without context is risky. Since modeling languages optimized for software systems are less effective at modeling the software system's environment, augmentations have been proposed to attach more domain knowledge to UML software specifications.

SADT/IDEF0 Feature	Benefits to UML
Activation Rule	In-context specification of business rule or decision-making rule.
Annotation -- Graphic (Diagram Highlights)	In-context system use case specifications are created by telling a story based on just the highlighted boxes and arrows.
Annotation -- Text (Diagram Notes)	A well-written paragraph for each box can turn into formal descriptions of the domain.
Context Diagram, Context Model	In-context general background knowledge: a) to any highest level of abstraction, b) to any lowest level of detail.
Control Versus Input	Separation of concerns: an accurate & complete model of the control system independent from an accurate & complete model of the transaction system.
Coupling/Cohesion (assessment)	Apply these concepts to a completed model to assess pathologies in the domain. For example: "we always did it that way" becomes immediately apparent.
Data Dictionary (i.e. "glossary")	In-context domain terminology, from which an ontology for the domain can be created,
Decomposition -- Stopping Heuristic	When decomposition stops when a function is all manual or all automated, then you have defined the human/system interface points.
Feedback Loop (output-input, output-control)	Useful for understanding: a) domain pathologies, b) interaction scenarios, c) architectural constraints.
Model Tie (i.e. "model call") Encoding	In-context formal description of "aspect," permits faster identification of cross-system common functionality.
Small, Multiple Models	Identification of key objects in the domain. Specify object functions independent of object modes/states.
Why, What, How (i.e. levels of abstraction)	Separation of concerns: distinct models for why (rationale), what (function), and how (mechanism) = modular understanding of context at different levels of abstraction.

Fig. 4. How UML can benefit from SADT/IDEF0 domain modeling

UML Augmentation. The augmentations that suggest strengthening UML's ability to define a software system's environment advise doing domain modeling using some other language or tool, and then linking captured knowledge to UML software specifications. For example: a) domain ontology database [6], b) general background knowledge base with reasoning logic [43], c) in-context identification, specification and validation of business rules [16] [47], and decision-making rules [58], d) how and why people do the work that they do [23], and e) formal descriptions of the domain [7]. Taken together, these augmentations suggest: a) that SADT/IDEF0 models of a domain contain knowledge that can benefit UML specifications, and b) efficacy can be achieved if domain modeling is a activity distinct from software specification. Figure 5 shows how SADT/IDEF0 diagrams and supplements can augment UML.

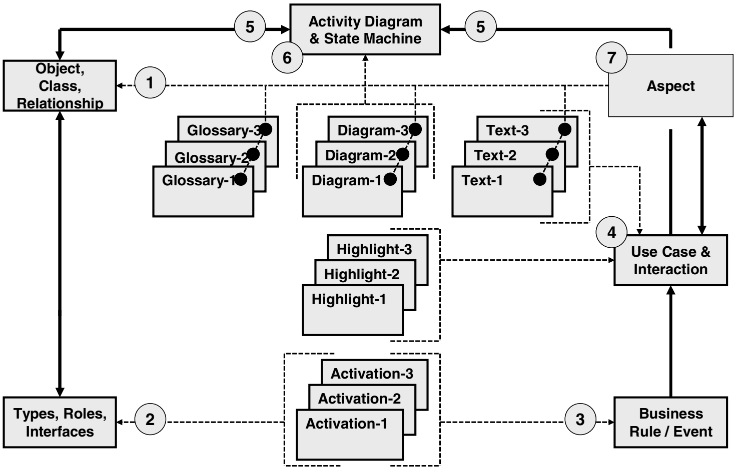


Fig. 5. The Step-by-step use of SADT/IDEF0 diagram and supplement content to augment the development of UML specifications

4.2 Benefits to Agile

Agile Shortcomings. One component of the Agile Manifesto advocates working software over comprehensive documentation [2]. Not surprisingly, traditional domain modeling methods have not heretofore been recommended for augmenting Agile software development efforts. However, "small method" augmentations have been recommended since Agile was first purported. These suggestions carefully distinguish "comprehensive" from "essential" documentation. Yes, comprehensive documentation can, when taken to the extreme, merely adds time and cost to projects without adding value to the software system. But taken to the other extreme, a lack of documentation altogether often creates gaps in verified understanding between users and software developers, and leaves no rationale behind for those who maintain or wish to reuse the resulting software system. Clearly, a middle ground of specification (i.e. for domain, analysis and design) would seem to benefit all parties, so long as those specifications are efficient and effective [3]. Figure 6 summarizes the benefits.

SADT/IDEF0 Feature	Benefits to Agile Software Development
Activation Rule	Complete hierarchy of rule cause-and-effect: Highest-level rule activation causes lower-level rule activations (traceability).
Annotation -- Graphic (Diagram Highlights)	In-context system use case specifications are created by telling a story based on just the highlighted boxes and arrows.
Annotation -- Text (Diagram Notes)	In-context informal descriptions of software activations (include in prototype wrapper documentation).
Context Diagram, Context Model	In-context general background knowledge: a) to any highest level of abstraction, b) to any lowest level of detail.
Control Versus Input	Understand how to make, and then keep fixed, major object architecture decisions.
Coupling/Cohesion (assessment)	Apply these concepts to a completed model to assess pathologies in the domain. For example: "we always did it that way" becomes immediately apparent.
Data Dictionary (i.e. "glossary")	Quickly understand the user's language, and the context for language usage.
Decomposition -- Stopping Heuristic	When decomposition stops when a function is all manual or all automated, then you have defined the human/system interface points.
Feedback Loop (output-input, output-control)	Document domain knowledge using self-reflection to uncover and assess tacit knowledge and fundamental assumptions.
Model Tie (i.e. "model call") Encoding	In-context formal description of "aspect," permits faster identification of cross-system common functionality.
Small, Multiple Models	Identification of key objects in the domain. Specify object functions independent of object modes/states.
Why, What, How (i.e. levels of abstraction)	Separation of concerns: distinct models for why (rationale), what (function), and how (mechanism) = modular understanding of context at different levels of abstraction.

Fig. 6. How agile can benefit from SADT/IDEF0 domain modeling

Agile Augmentation. Past recommendations have suggested augmenting the informal artifacts of Agile, and advocate for practices that focus on the domain to explain why: a) people need the system, b) will use the system in particular ways, c) they expect to see certain menus, displays, interactions, and functionality, and d) they are investing their time in the software development project. For example: a) documenting domain knowledge using JAD [14], self-reflection [14], and Wikis [33] [11], b) making and keeping fixed major object architecture decisions which enable parallel development by many Agile teams in support of very large projects [34] [3], c) documenting system design knowledge with informal specifications [37], and informal tools [8], d) making explicit tacit design assumptions with Total Quality methods [12] and self-reflection [36], and e) publishing (including vital documentation) competing prototypes to the wider community for evaluation and selection a best solution for reuse [50]. Taken together, these recommendations point to an interesting line of augmentation (Figure 3) by using traditional modeling methods such as SADT/IDEF0. Figure 7 shows how SADT/IDEF0 diagrams and supplements can be used to augment Agile.

4.3 Benefits to Usability Engineering

Usability Engineering Shortcomings. Practitioners and researchers have already shown: a) how Usability Engineering can be combined with Agile [15] [59], b) that particular combinations can enable effective design space exploration [32], c) and that the prototypes from those explorations can be systematically evaluated and augmented to create best-in-class production software [49]. However, such outcomes rely on augmenting the traditional usability engineering methods with very good knowledge acquisition methods and very good modeling tools [44] [41]. To explain, Usability Engineering has had a tradition of employing the concepts and methods of participatory design [42] to obtain optimal understanding of a domain and especially the tacit knowledge of domain inhabitants. So, traditional Usability Engineering methods have employed ethnographic techniques, which have traditionally relied on hand-written field notebooks and not on formal models. But, adding formal modeling to ethnographic practices can add value [25]. Also, with the advent of Computer Aided Software Engineering (CASE) tools, the creation and review of formal models can happen much more quickly than in the days of purely manual drawing, copying, distributing copies, the recording of feedback, and so on.

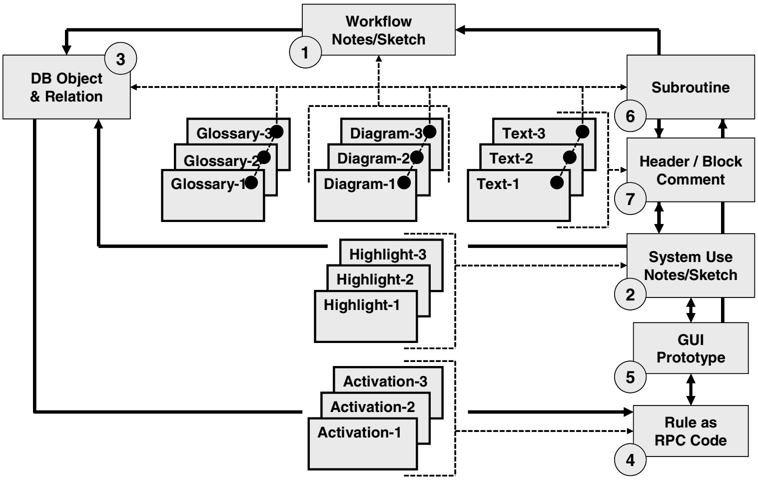


Fig. 7. The Step-by-step use of SADT/IDEF0 diagram and supplement content to augment the artifacts of agile software development

SADT/IDEF0 Feature	Benefits to Usability Engineering
Activation Rule	In-context interaction posibilities (patterns) and their rationale, plus associated potential implications (claims).
Annotation -- Graphic (Diagram Highlights)	In-context specification of work tasks. Context provides background and rationale for the users' work.
Annotation -- Text (Diagram Notes)	Text for all manual boxes becomes an in-context description of people's work.
Context Diagram, Context Model	Formalize and limit "context," noting how relevant information differs from context to context.
Control Versus Input	Distinguish which user generated artifacts are simply material for the next step in processing from those artifacts than govern subsequent workflow steps.
Coupling/Cohesion (assessment)	Apply these concepts to a completed model to assess pathologies in the domain. For example: "we always did it that way" becomes immediately apparent.
Data Dictionary (i.e. "glossary")	Quickly understand the user's language, and the context for language usage.
Decomposition -- Stopping Heuristic	When decomposition stops when a function is all manual or all automated, then you have defined the human/system interface points.
Feedback Loop (output-input, output-control)	Use to create test cases to evaluate software prototypes (in-context cases).
Model Tie (i.e. "model call") Encoding	Create patterns by unifying the often scattered aspects (i.e. usage behaviors) by constructing themes (i.e. relationship rules among aspects).
Small, Multiple Models	Create a context-based, generalized navigation "space" model, and then use it to create a UI presentation model.
Why, What, How (i.e. levels of abstraction)	Distinguish local dynamics from global dynamics from contextual dynamics .

Fig. 8. How usability engineering can benefit from SADT/IDEF0 domain modeling

Usability Engineering Augmentation. Many augmentations to Usability Engineering have been suggested, and most have been centered on incorporating ethnographic concepts and field work. Some of the most noteworthy augmentations are: a) models that distinguish local dynamics from global dynamics from contextual dynamics [45], b) a context-based, generalized navigation space model that is used that model to create a UI presentation model [22], c) formalized context that shows how information differs from context-to-context [28], d) UI design trade-offs via patterns – in-context problem-solution pairs – and claims – implications of design decisions [1] e) a claims library that enables UI design reuse [60], e) patterns that unify the highly scattered aspects of usage behavior via a set of themes that define relationship rules for aspects [4], and f) domain models that have syntax and semantics that enable consistency across architectural, design, structural, behavioral models [13] [26].

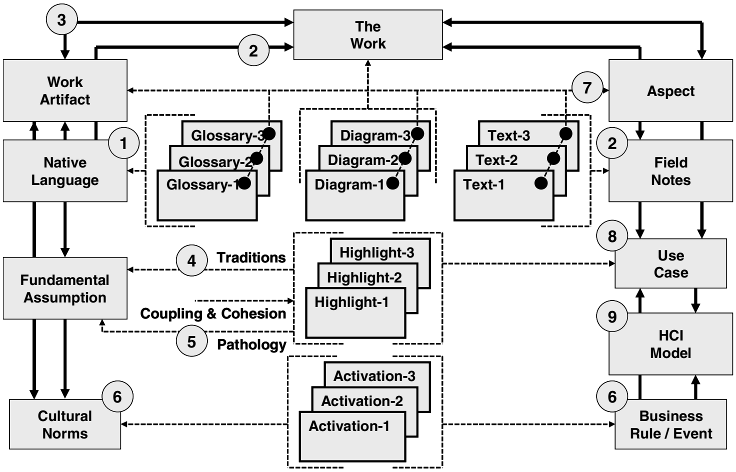


Fig. 9. The Step-by-step use of SADT/IDEF0 diagram and supplement content to augment usability engineering

5 Summary, Conclusions and Future Work

This paper has taken an approach to providing benefits to UML, Agile, and Usability Engineering methods by using SADT/IDEF0: a) for domain modeling, and b) in particular ways based on over 10 years of experience with these methods by a variety of practitioners and researchers. These experiences were selected based on their: a) advocating specific augmentations to the aforementioned methods, and b) showing how those augmentations could benefit: i) the software development process advocated by the method, ii) any software or non-software prototypes generated by the method, and iii) the reuse and maintenance of the final specifications generated by the method. The recommended shortcomings and corresponding improvement recommendations were used to develop the proposed augmentation approach.

The approach centers on comprehensive, correct and consistent specifications of domain knowledge. When used properly, SADT/IDEF0 can create such specifications of an entire domain, not just the software system's immediate context. And it can describe that domain in rich and varied ways using in-context supplements which contain the: language, beliefs, assumptions, human organization, human work, work tasks and tools, and system usage expectations, vital to the successful application of UML, Agile, and Usability Engineering methods. Figure 10 summarizes the role SADT/IDEF0 plays in the augmentation process. It also shows how the SADT/IDEF0

Author Reader Cycle [24]) can augment Agile by providing domain experts time to think about the knowledge already given to software developers to ensure facts are consistent and correct with the current common understanding.

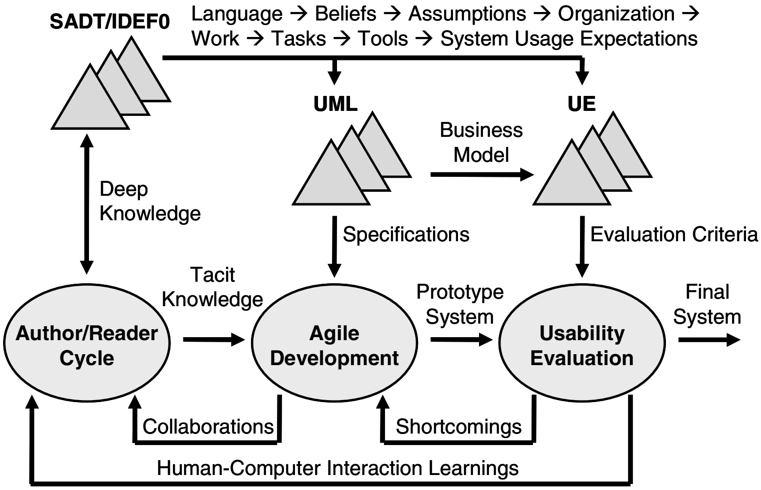


Fig. 10. Augmenting UML, agile and usability engineering with SADT/IDEF0 models and the author/reader review cycle

The combination of over 10 years of experience by practitioners and researchers, their recommendations for improving upon the shortcomings they discovered, and the ability of the SADT/IDEF0 to support those recommendations, led the author to conclude that there is also merit for further elaboration and demonstration of the approach's viability by extending a commercial SADT/IDEF0 tool. Specifically, such a CASE tool could be extended by: a) enhancing its existing ontology of SADT/IDEF0, b) integrating that ontology with UML tools, c) creating an interface to a domain knowledge reasoning system and a formal specification system, and d) building a component for the automatic generation of a deep human-system interaction model that includes patterns and claims [1]. A proposal for future work is underway.

Acknowledgements. The author wishes to first and foremost acknowledge the late Douglas T. Ross for his contributions to the fields of industrial engineering and computer science, and for his inventions of APT, PLEX and SADT™. In alphabetical order, acknowledgements go to: Michael Connor, Melvin Dickover, Patricia Duran, Clarence Feldman, Al Irvine, Clement McGowan, Robert Munck, Chuck Patrick, Kenneth Schoman, Michelle Stowe, and the late Daniel Thornhill – the people whose early work with SADT™, and later with its public domain version IDEF0, greatly furthered: its ontology, the correct understanding of its principles, and how to correctly practice its methods. Lastly, recognition is given to the insightful researchers and practitioners cited in this paper for experiences with the UML, Agile and Usability Engineering methods, and their work in clarifying shortcomings and related practical issues surrounding the domain modeling embedded in those software development methods.

References

1. Abraham, G., Atwood, M.: Patterns or claims: do they help in communicating design advice? In: Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group (2009)
2. Abrahamsson, P., et al.: Agile Software Development Methods: Review and Analysis. VTT Publications, Number 478, Kaitovayla (2002)
3. Alleman, G.B.: Agile Project Management Methods for ERP: How to Apply Agile Processes to Complex COTS Projects and Live to Tell about It. In: Wells, D., Williams, L. (eds.) XP 2002. LNCS, vol. 2418, pp. 70–88. Springer, Heidelberg (2002)
4. Baniassad, E., Clarke, S.: Theme: An Approach for Aspect-Oriented Analysis and Design. In: Proceedings of the 26th International Conference on Software Engineering (2004)
5. Booch, G., et al.: The Unified Modeling Language User Guide. Addison-Wesley, Boston (1999)
6. Brockmans, S., Haase, P., Hitzler, P., Studer, R.: A Metamodel and UML Profile for Rule-Extended OWL DL Ontologies. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 303–316. Springer, Heidelberg (2006)
7. Bryant, B., et al.: From Natural Language Requirements to Executable Models of Software Components. In: Proceedings of the Monterey Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation (2003)
8. Bryant, S., et al.: Pair programming and the re-appropriation of individual tools for collaborative software development. In: Proceedings of the Conference on Cooperative Systems Design (2006)
9. Calvary, G., Thevenin, D.: A Unifying Reference Framework for the Development of Plastic User Interfaces. In: Little, M.R., Nigay, L. (eds.) EHCI 2001. LNCS, vol. 2254, pp. 173–192. Springer, Heidelberg (2001)
10. Congram, C., Epelman, M.: How to describe your service: An invitation to Structured Analysis and Design Technique. International Journal of Service Industry Management 6(2) (1995)
11. Decker, B., et al.: A framework for Agile reuse in software engineering using Wiki Technology. In: Proceedings of the Knowledge Management for Distributed Agile Processes Workshop (2005)
12. Dingsøyr, T., Hanssen, G.K.: Extending Agile Methods: Postmortem Reviews as Extended Feedback. In: Henninger, S., Maurer, F. (eds.) LSO 2003. LNCS, vol. 2640, pp. 4–12. Springer, Heidelberg (2003)
13. Egyed, A., Medvidovic, N.: A Formal Approach to Heterogeneous Software Modeling. In: Maibaum, T. (ed.) FASE 2000. LNCS, vol. 1783, pp. 178–192. Springer, Heidelberg (2000)
14. Friedrich, W., van der Poll, J.: Towards a Method to Elicit Tacit Domain Knowledge from Users. Interdisciplinary Journal of Information, Knowledge, and Management 2 (2007)
15. Granollers, T., et al.: Usability Engineering Process Model. Integration with Software Engineering. In: Proceedings of the HCI International Conference 2003 (2003)
16. Halpin, T.: Augmenting UML with fact-orientation. In: Proceedings of the 34th Annual Hawaii International Conference on Systems Sciences (2001)
17. Hilliard, R.: Using Aspects in Architectural Description. In: Moreira, A., Grundy, J. (eds.) Early Aspects 2007 Workshop. LNCS, vol. 4765, pp. 139–154. Springer, Heidelberg (2007)
18. Hilliard, R.: Aspects, Concerns, Subjects, Views, ... In: OOPSLA 1999 Workshop on Multi-Dimensional Separation of Concerns in Object-Oriented Systems (1999)
19. Hilliard, R.: Views and Viewpoints in Software Systems Architecture. In: First Working IFIP Conference on Software Architecture (WICSA 1) (1999)
20. Hilliard, R., et al.: The architectural metaphor as a foundation for systems engineering. In: Proceedings of the 6th Annual International Symposium of the International Council on Systems Engineering (1996)

21. Iachello, G., Abowd, G.: From privacy methods to a privacy toolbox: Evaluation shows that heuristics are complementary. ACM Transactions on Computer-Human Interaction 15(2) (2008)
22. Koch, N., et al.: The Authoring Process of the UML-based Web Engineering Approach. In: Proceedings of the 1st International Workshop on Web-oriented Software Technology (2001)
23. Larsen, G.: Designing component-based frameworks using patterns in the UML. CACM 42(10) (1999)
24. Marca, D.: IDEF0 and SADT: A Modeler's Guide, 3rd edn. OpenProcess, Inc., Boston (2006)
25. Marca, D.: Augmenting SADT to develop computer support for cooperative work. In: Proceedings of the 13th International Conference on Software Engineering (1991)
26. Medvidovic, N., et al.: Round-Trip Software Engineering Using UML: From Architecture to Design and Back. In: Proceedings of the 7th European Conference on Software Engineering (1999)
27. Menard, R.: Domain modeling: Leveraging the heart of RUP for straight through processing. IBM Developer Works (2003), `http://www.ibm.com/developerworks/rational/library/2234.html` (retrieved on March 17, 2011)
28. Najar, S., et al.: Semantic representation of context models: a framework for analyzing and understanding. In: Proceedings of the 1st Workshop on Context, Information and Ontologies, CIAO 2009 (2009)
29. Nielsen, J.: Usability Engineering. Academic Press, London (1993)
30. Normantas, K., Vasilecas, O., Sosunovas, S.: Augmenting UML with decision table technique. In: International Conference on Computer Systems and Technologies: CompSys-Tech 2009 (2009)
31. Ng, J., et al.: The development of an enterprise resources planning system using a hierarchical design pyramid. Journal of Intelligent Manufacturing 9(5) (1996)
32. Paelke, V., Nebe, K.: Integrating Agile methods for mixed reality design space exploration. In: Proceedings of the 7th ACM Conference on Designing Interactive Systems, DIS 2008 (2008)
33. Rech, J., et al.: Riki: A System for Knowledge Transfer and Reuse in Software Engineering Projects. In: Lytras, M., Naeve, A. (eds.) Open Source for Knowledge and Learning Management: Strategies Beyond Tools. IGI Global Publishers (2007)
34. Reifer, D., et al.: Scaling Agile Methods. IEEE Software (July/August 2003)
35. Ross, D.: Structured Analysis (SA): A Language for Communicating Ideas. IEEE Transactions on Software Engineering 3(1) (1977)
36. Salo, O., Kolehmainen, K., Kyllönen, P., Löthman, J., Salmijärvi, S., Abrahamsson, P.: Self-Adaptability of Agile Software Processes: A Case Study on Post-iteration Workshops. In: Eckstein, J., Baumeister, H. (eds.) XP 2004. LNCS, vol. 3092, pp. 184–193. Springer, Heidelberg (2004)
37. Scacchi, W.: Is Open Source Software Development Faster, Better, and Cheaper than Software Engineering? In: Proceedings of the 2nd ICSE Workshop on Open Source Software Engineering (2002)
38. Schoman, K., Ross, D.: Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering 3(1) (1977)
39. Seffah, A., et al.: HCI, Usability and Software Engineering Integration: Present & Future. In: Human-Centered Software Engineering: Integrating Usability in the Software Development Lifecycle. HCI Series, vol. 8 (2005)
40. Siltala, M.: Modeling Contracting Procedure and the Concept of the Service Portfolio for Finnish Municipalities using SADT. Nordic Journal of Surveying and Real Estate Research 1 (2009)

41. Sitou, W., Spanfelner, B.: Towards requirements engineering for context adaptive systems. In: 31st Annual International Computer Software and Applications Conference, COMPSAC 2007, vol. 2 (2007)
42. Spradley, J.: Participant Observation. Holt, Rinehart and Winston, London (1980)
43. Süß, J., Leicher, A.: Augmenting Domain Specific UML Models with RDF. In: Proceedings of the 3rd Workshop in Software Model Engineering, Lisbon (2004)
44. Sutcliffe, A.: Applying small group theory to analysis and design of CSCW systems. In: Proceedings of the Workshop on Human and Social Factors of Software Engineering, HSSE 2005 (2005)
45. Sutcliffe, A.: On the effective use and reuse of HCI knowledge. ACM Transactions on Computer-Human Interaction (TOCHI) 7(2) (2000)
46. Sutcliffe, A.: The Domain Theory for Requirements Engineering. IEEE Transactions on Software Engineering 24(3) (1998)
47. Skersys, T., Gudas, S.: The Enhancement of Class Model Development Using Business Rules. In: Bozanis, P., Houstis, E.N. (eds.) PCI 2005. LNCS, vol. 3746, pp. 480–490. Springer, Heidelberg (2005)
48. Wahid, S.: Investigating design knowledge reuse for interface development. In: Proceedings of the 6th Conference on Designing Interactive Systems, DIS 2006 (2006)
49. Verlinden, J., Horva, I.: Analyzing opportunities for using interactive augmented prototyping in design practice. In: Artificial Intelligence for Engineering Design, Analysis and Manufacturing. Cambridge University Press (2009)
50. Lethbridge, T.C., Laganiére, R.: Object-Oriented Software Engineering: Practical Software Development Using UML and Java. McGraw-Hill, London (2001)
51. Winckler, M., et al.: Tasks and scenario-based evaluation of information visualization techniques. In: Proceedings of the 3rd Annual Conference on Task Models and Diagrams, TAMODIA 2004 (2004)
52. Novak, J., Cañas, A.: The Theory Underlying Concept Maps and How to Construct Them, Technical Report IHMC CmapTools 2006-01 Rev 01-2008, Florida Institute for Human and Machine Cognition (2008)
53. Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J., Neema, S.: Developing applications using model-driven design environments. IEEE Computer 39(2) (2006)
54. Hruby, P.: Ontology-based domain-driven design. In: OOPSLA Workshop on Best Practices for Model-Driven Software Development, San Diego, CA, USA (2005)
55. Wikipedia: Integrated Computer-Aided Manufacturing (2011), http://en.wikipedia.org/wiki/Integrated_Computer-Aided_Manufacturing (retrieved March 20, 2011)
56. Coste, P., et al.: Multilanguage Design of Heterogeneous Systems. In: CODES 1999 (1999)
57. Stuikys, V., Damasevicius, R.: Relationship Model of Abstractions Used for Developing Domain Generators. Informatica 13(1) (2001)
58. Vasilecas, O., Normantas, K.: Decision table based approach for business rules modelling in UML/OCL. In: Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop (2010)
59. Seffah, A., et al.: HCI, Usability and Software Engineering Integration: Present & Future. In: Human-Centered Software Engineering: Integrating Usability in the Software Development Lifecycle. HCI Series, vol. 8 (2005)
60. Brel, C., Renevier-Gonin, P., Occello, A., Déry-Pinna, A.-M., Faron-Zucker, C., Riveill, M.: Application Composition Driven By UI Composition. In: Bernhaupt, R., Forbrig, P., Gulliksen, J., Lárusdóttir, M. (eds.) HCSE 2010. LNCS, vol. 6409, pp. 198–205. Springer, Heidelberg (2010)

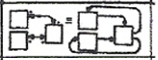
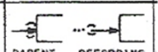
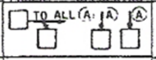
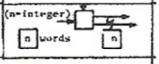
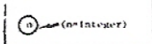
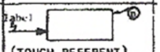
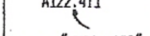
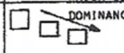
Appendix

1 "Features" of the SADT Ontology

Table 1. SADT "features" published in 1977 by Douglas Ross [35]

		PURPOSE	CONCEPT		MECHANISM	NOTATION	
Context	1	BOUND CONTEXT	INSIDE/OUTSIDE		SA BOX	NAME	
Arrow	2	RELATE/CONNECT	FROM/TO		SA ARROW	LABEL	
Transform	3	SHOW TRANSFORMATION	INPUT-OUTPUT		SA INTERFACE	INPUT OUTPUT	
Control	4	SHOW CIRCUMSTANCE	CONTROL		SA INTERFACE	CONTROL	
Means	5	SHOW MEANS	SUPPORT		SA MECHANISM	MECHANISM	
Verbs	6	NAME APTLY	ACTIVITY HAPPENINGS	DATA THINGS	SA NAMES	ACTIVITY VERB	DATA NOUN
Nouns	7	LABEL APTLY	THINGS	HAPPENINGS	SA LABELS	NOUN	VERB
Path	8	SHOW NECESSITY	I-O	C-O	PATH		
Dominance	9	SHOW DOMINANCE	C	I	CONSTRAINT		
Relevance	10	SHOW RELEVANCE	ICO	ICO	ALL INTERFACES		
Omissions	11	OMIT OBVIOUS	C-O	I-O	OMITTED ARROW		
Branches	12	BE EXPLICIT WITHOUT CLUTTER	PIPELINES, CONDUITS, WIRES		BRANCH	A A A	
Joins	13				JOIN	A A A	
	14	BE CONCISE AND CLEAR	CABLES, MULTI-WIRES		BUNDLE	A B C(=AUB)	
	15				SPREAD	C=(AUB) A B	
OR	16	SHOW EXCLUSIVES	EXPLICIT ALTERNATIVES		OR BRANCH	A OR B A B	
AND	17				OR JOIN	A B A OR B	
Boundary	18	SHOW INTERFACES TO PARENT DIAGRAM	PARENT CHILD ARROWS PENETRATE		SA BOUNDARY ARROWS (ON CHILD)	NO BOX SHOWN	
Parent ICOM	19	SHOW EXPLICIT PARENT CONNECTION	NUMBER CONVENTION FOR PARENT, WRITE ICOM CODE ON CHILD BOUNDARY ARROWS		I C M O	c1 c2 o2 (ON CHILD)	
	20	SHOW UNIQUE DECOMPOSITION	DETAIL REFERENCE EXPRESSION (DRE)		C-NUMBER OR PAGE NUMBER OF DETAIL DIAGRAM	BOX DRE	
Calls	21	SHOW SHARED OR VARIABLE DECOMPOSITION	DRE WITH (MODEL NAME)		SA CALL ON SUPPORT	BOX STUB DRE	

Table 1. *(continued)*

		PURPOSE	CONCEPT	MECHANISM	NOTATION
Feedback	22	SHOW COOPERATION	INTERCHANGE OF SHARED RESPONSIBILITY	SA 2-WAY ARROWS	
Pipeline	23	SUPPRESS INTERCHANGE DETAILS	ALLOW 2-WAY WITHIN 1-WAY PIPELINES	2-WAY TO 1-WAY BUTTING ARROWS	
Tunnel	24	SUPPRESS "PASS-THROUGH" CLUTTER	ALLOW ARROWS TO GO OUTSIDE DIAGRAMS	SA "TUNNELING" (WITH REFERENCES)	 PARENT OFFSPRING
To/From All	25	SUPPRESS NEEDED-ARROW CLUTTER	ALLOW TAGGED JUMPS WITHIN DIAGRAM	TO ALL or FROM ALL	 TO ALL (A) (A) (A)
Note	26	SHOW NEEDED ANNOTATION	ALLOW WORDS IN DIAGRAM	SA NOTE	NOTE:
Footnote	27	OVERCOME CRAMPED SPACE	ALLOW REMOTE LOCATION OF WORDS IN DIAGRAM	SA FOOTNOTE	 (n=integer) n words n
Meta-Note	28	SHOW COMMENTS ABOUT DIAGRAM	ALLOW WORDS ON (NOT IN) DIAGRAM	SA META-NOTE	 n (n=integer)
Squiggle	29	ENSURE PROPER ASSOCIATION OF WORDS	TIE WORDS TO INTENDED SUBJECT	SA "SQUIGGLE"	 label n (TOUCH REFERENT)
Sequence	30	UNIQUE SHEET REFERENCE	CHRONOLOGICAL CREATION	SA C-NUMBER	AUTHOR INITS INTEGER
Node	31	UNIQUE BOX REFERENCE	PATH DOWN TREE FROM BOX NUMBERS	SA NODE NUMBER (BOX NUMBERS)	A, D, OR M ç PARENT # ç BOX #
Model	32	SAME FOR MULTI-MODELS	PRECEDE BY MODEL NAME	SA MODEL NAME	MODEL NAME/NODE#
Interface	33	UNIQUE INTERFACE REFERENCE	ICOM WITH BOX NUMBER	SA BOX ICOM	BOX# ç ICOM CODE
To-From	34	UNIQUE ARROW REFERENCE	FROM - TO	PAIR OF BOX ICOMs	BOX $ICOM_1$ BOX $ICOM_2$
Reference	35	SHOW CONTEXT REFERENCE	SPECIFY A REFERENCE POINT	SA REF.EXP. "DOT"	 A122.4I1 "WHICH SEE"
Dominance	36	ASSIST CORRECT INTERPRETATION	SHOW DOMINANCE GEOMETRICALLY (ASSIST PARSE)	STAIRCASE LAYOUT	 DOMINANCE
Description	37	ASSIST UNDERSTANDING	PROSE SUMMARY OF MESSAGE	SA TEXT	NODE# ç T ç INTEGER
Highlights	38	HIGHLIGHT FEATURES	SPECIAL EFFECTS FOR EXPOSITION ONLY	SA FEOs	NODE# ç F ç INTEGER
Glossary	39	DEFINE TERMS	GLOSSARY WITH WORDS & PICTURES	SA GLOSSARY	MODEL NAME ç G ç INTEGER
Index	40	ORGANIZE PAGES	PROVIDE TABLE OF CONTENTS	SA NODE INDEX	NODE# ORDER