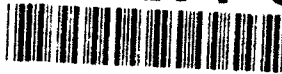


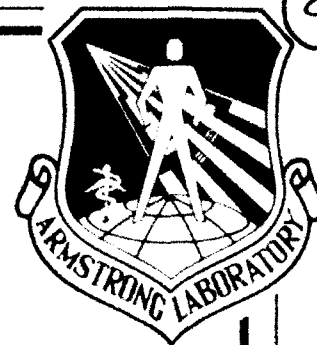
ARMSTRONG
LABORATORY

AL-TP-1992-0050

AD-A261 594



DTIC
ELECTE
MAR 2 1993
S C D



2

IDEF6: A DESIGN RATIONALE CAPTURE METHOD CONCEPT PAPER

Richard J. Mayer
Patricia A. Griffith
Christopher P. Menzel

93-04241



28 pr
200

KNOWLEDGE BASED SYSTEMS LABORATORY
DEPARTMENT OF INDUSTRIAL ENGINEERING
TEXAS A&M UNIVERSITY
COLLEGE STATION, TX 77843

Michael K. Painter, Capt, USAF

HUMAN RESOURCES DIRECTORATE
LOGISTICS RESEARCH DIVISION

NOVEMBER 1992

INTERIM TECHNICAL PAPER FOR PERIOD JANUARY 1990 - MARCH 1991

Approved for public release; distribution is unlimited.

AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6573

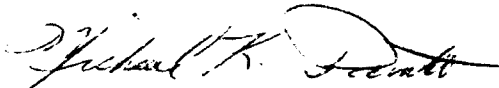
98

NOTICES

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation, or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this paper, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.



MICHAEL K. PAINTER, Capt, USAF
Program Manager



BERTRAM W. CREAM, Chief
Logistics Research Division

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
<small>Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.</small>				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE November 1992		3. REPORT TYPE AND DATES COVERED Interim - January 1990 to March 1991
4. TITLE AND SUBTITLE IDEF6: A Design Rationale Capture Method Concept Paper			5. FUNDING NUMBERS C - FQ7624-90-00010 PE - 63106F PR - 2940 TA - 01 WU - 15	
6. AUTHOR(S) Richard J. Mayer Patricia A. Griffith Christopher P. Menzel Michael K. Painter, Capt, USAF				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Knowledge Based Systems Laboratory Department of Industrial Engineering Texas A&M University College Station, TX 77843			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Armstrong Laboratory Human Resources Directorate Logistics Research Division Wright-Patterson AFB, OH 45433-6573			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AL-TP-1992-0050	
11. SUPPLEMENTARY NOTES Armstrong Laboratory Technical Monitor: Michael K. Painter, AL/HRGA, (513) 255-7775				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) When explicitly captured, design rationale typically exists in the form of unstructured textual comments. In addition to making it difficult, if not impossible to find relevant information on demand, lack of a structured method for organizing and providing completeness criteria for design rationale capture makes it unlikely that important information will be documented. Unlike design methods which serve to document WHAT a design is (Design Specification), the IDEF6 Design Rationale Capture Method is targeted at capturing WHY a design is the way it is -- or WHY it is not manifested in some other form -- together with HOW the final design configuration was reached. IDEF6 was intended to be a method with the representational capability to capture <i>information system</i> design rationale and associate that rationale with the design models and documentation for the end system. Thus, IDEF6 attempts to capture the logic underlying the decisions contributing to, or resulting in, the final design. The explicit capture of design rationale serves to help avoid repeating past mistakes, provides a direct means for determining the impact of proposed design changes, forces the explicit statement of goals and assumptions, and aids in the communication of final system specifications.				
14. SUBJECT TERMS design design history design rationale		IDEF information engineering information systems integration		15. NUMBER OF PAGES
		knowledge acquisition method methodology systems engineering		16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR	

Table of Contents

Preface	iv
Summary	v
1.0 IDEF6 Method Development Approach.....	1
2.0 Motivations	2
3.0 Nature of Design Rationale	4
3.1 Nature of Design.....	4
3.2 Characteristics of the Design Process	9
4.0 Design Rationale Phenomena.....	12
4.1 Preliminary Ontology of Design Rationale.....	13
4.2 Issues with Design Rationale Capture	18
5.0 Preliminary IDEF6 Concepts	18
6.0 The IDEF6 Automation Support Environment	20
7.0 Unallocated Thoughts, Comments, Ideas, Issues, etc.....	21
8.0 References.....	23

Accession For	
NTIS	<input checked="checked" type="checkbox"/>
CRA&I	<input type="checkbox"/>
DTIC	<input type="checkbox"/>
TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

Preface

This paper describes the research accomplished at the Knowledge Based Systems Laboratory of the Department of Industrial Engineering at Texas A&M University. Funding for the Laboratory's research in Integrated Information System Development Methods and Tools has been provided by the Logistics Research Division of the Armstrong Laboratory (AL/HRG), Wright-Patterson Air Force Base, Ohio 45433, under the technical direction of USAF Captain Michael K. Painter, under subcontract through the NASA Research Institute for Computing and Information Systems (RICIS) Program at the University of Houston.

Summary

This paper presents the results of research towards a design rationale capture method referred to as IDEF6. The purpose of IDEF6 is to facilitate the acquisition, representation, and manipulation of the design rationale utilized in the development of enterprise-level information systems. The term "rationale" is interpreted as the "reason, justification, underlying motivation, or excuse" that moved the designer to select or adopt a particular strategy or system feature. More simply, "rationale" is interpreted as the nature of the answer given to the question, "Why is this design the way it is?"

IDEF6 will be a method that possesses the conceptual resources and linguistic capabilities needed (i) to represent the nature and structure of the information that constitutes design rationale within a given system, and (ii) to associate that rationale with design specifications, models, and documentation for the system. The scope of IDEF6 applicability covers all phases of the information system development process, from initial conceptualization through both preliminary and detailed design activities. To the extent that detailed design decisions for software systems are relegated to the coding phase, the IDEF6 technique should be usable during the software construction process as well.

The scope of IDEF6 is being conceived under the following assumptions.

- 1) IDEF6 is targeted towards facilitating the capture of design rationale for enterprise-level information systems from system-level design to detailed design of the implementation of data structures, algorithms, user interface, and processes.
- 2) People rarely write down design assumptions or rationale. To the extent possible, IDEF6 must be able to be incorporated in a transparent manner into a wide variety of design methods (both formal and informal). It is unreasonable to expect designers to sit down at some point in time and "model" design rationale. Rationale must be captured at the source--at the point in time at which decisions are made.

1.0 IDEF6 Method Development Approach

The nature of design both as an individual cognitive process and as an organizational endeavor is an area of intense research. Understanding this phenomena is important to researchers in artificial intelligence, developers of next generation computer-aided design (CAD) and computer-aided engineering (CAE) environments, and initiatives in concurrent engineering. Design rationale (the focus of this study) is intimately tied to a conceptualization of the design activity. These considerations led to the establishment of a recommended three-front approach to the evolution of IDEF6 as outlined in the following paragraphs.

The first activity in the development of IDEF6 should be focused on the definition of the characteristics and the nature of design rationale by analytic means. This will involve characterization of the types of information used to rationalize a design. This characterization process requires the study of both completed and ongoing software development projects to classify information types and understand the time frame and design process context in which they are relevant. It also involves discussion with experienced information system designers. The method we are using is thus essentially to build an ontology of design rationale. As IDEF5 (Ontology Description Capture Method) evolves, this process will be augmented by the identification of the types of structures used to rationalize a design and the building of an IDEF5 model of that information.

The second activity should be focused on the definition of the characteristics and the nature of design rationale by direct experimentation. This activity would also experiment with language structures designed around the concepts identified in the first task. Specifically, we recommend experimentation with an IDEF6 v0.1 annotation format using IDEF4 (Object Oriented Design Method) as a test case. IDEF4 models could be easily extended to include IDEF6 annotations describing assumptions, constraints, or usage categories underlying IDEF4 entities. In addition to the annotations, we foresee the need to define an Information System Constraint Language (ISyCL) slice to accurately capture the necessary information.

Once the annotations and language facilities are available, we would like to pursue two avenues for direct experimentation. The first would focus on the discovery of design rationale from a study of individuals trying to understand existing code. This would primarily involve the observation of (1) questions asked and (2) answers given or derived. The second avenue

for direct experimentation with the IDEF6 v0.1 annotations and the ISyCL slice would involve structured interviews with the Integrated Development Support Environment (IDSE) software designers to determine what rationale can be captured during the design definition process. This approach could be augmented by building an IDEF3 description of the design process as experienced during the IDSE experiment. One characteristic of design rationale we might check is the opinion that "design rationale only becomes important when there are options, that is, when a design decision is not completely determined by the constraints of the situation." Thus, decision points must be identified, the situations and constraints associated with those decision points defined, and (if options exist) the rationale for not only the option chosen but also the reasons for discarding those not chosen must be recorded.

The third activity should focus on the conceptualization of the necessary automated support for IDEF6. It is expected that IDEF6 will be truly useful only within the context of a sophisticated, automated, design support system because it must serve as the bridge which connects the various perspectives on an evolving design that are generated over the design life cycle. Thus, it must be supported in an environment that allows direct "windowing" into various artifacts of the design process (e.g. requirements specs, IDEF0, 1, 3, 4, and 5 models, PDL descriptions, and code). While the strict form of IDEF6 may be based on a classification of design rationale information, it must nevertheless be designed with usage in an automated environment in mind. Hypertext capabilities and alternative versioning are among the capabilities that must be explored. The development of requirements for such an environment will be part of this IDEF6 automation support conceptualization task.

The remainder of this report details the results of our analysis in the first and third areas described above. This first phase focused primarily on identification of technical issues and proof of viability of the strategy outlined above.

2.0 Motivations

To support the evolution of integrated information systems with lifetimes which may extend over many career periods requires the explicit capture and storage of design rationale in a computer-manipulatable form. However, capture of design rationale is also important during the development phase of large-scale systems. In these situations, the logic (chain of arguments or reasons) behind the design is invaluable to the downstream developers, testers, and integrators.

The idea of environments for the purpose of software design specification and analysis is not new. In some sense, all the common utilities that comprise programming environments--compilers, debugging tools, version management tools, and so on--support the design and development of software. Such tools, of course, are inherently limited by their applicability only at the stage of immediate code production. Computer-aided software engineering (case)¹ environments attempt to bring automated support to the design stage. In specific situations, they have demonstrated an ability to accomplish the purpose for which they were intended. Even so, existing case tools are inherently limited in at least two important respects. First, case tools are intended to document various aspects of *what* a design is, but they were never intended to document *why* a design exists, certainly not in any methodical way that cuts across design representations. Second, even when design rationale comments exist, they are just that--unstructured textual comments. There have been no tools (or methods) designed to gather the design rationale that can only be gleaned from recordings of the design decisions that are made over time and in the course of working through various aspects of a design. The only coherent record of the design rationale for a software system exists solely in the head of a lead designer for the system. For large systems, even the lead designer will never acquire all of it. In such cases, the rationale is distributed across many people, and at any one time, parts of it will have been forgotten or be unavailable. Sympathize with the implementors; pity the maintainers.

The consequences of the loss of design rationale are all too familiar to software engineers: past mistakes are repeated, and decisions made contrary to the original design assumptions solve one problem but create three more. In software design, the local changes are fairly easy (as long as they are really local); it is the global changes (restructuring) that are difficult. One motivation of design rationale capture is to force software designers to carefully plan the global structure.

One benefit of maintaining design rationale is that it forces designers to state goals as well as assumptions. Goals, like assumptions, are frequently unstated. Forcing their statement for the purpose of rationale capture will inevitably lead to a more focused, disciplined approach to design. In addition, the area of information system development has had a long-standing need for "languages of thought" to leverage the capabilities of the designer or to communicate specifications to other members of the design team. As discussed later in this document, much of design thinking appears

¹ Lower-case letters are used to distinguish between Computer-Aided Software Engineering and Computer-Aided Systems Engineering (CASE).

to be abductive in nature, with experience-directed insights being fashioned and rationalized in the context of the current task. This rationalization may be the only basis for understanding why a system is the way it is. Without the capture of this chain of reasoning or arguments, communication of the design becomes difficult and error-prone.

Another motivation for rationale capture is that the definition of subsystems and subsystem boundaries is an experimentation process with each designer discovering the boundaries he/she finally imposes. Knowledge of the paths of inquiry that failed, the path to the final success, and the final result is key if the organization is to avoid costly errors.

3.0 Nature of Design Rationale

In considering the nature of design rationale (why and how), we must contrast it with the related notions of: 1) design specification (what), and 2) design history (steps taken). Design specifications describe what should be realized in the final physical artifact. Design rationale describes why the design specification is the way it is and how the specified artifact is intended to work. This includes such information as: principles and philosophy of operation, models of correct behavior, and models of how the artifact is intended to behave as it fails. The design process history records the steps that were taken, the plans and expectations that led up to these steps, and the results of each step.

3.1 Nature of Design

What is design? Why can some people do it well and others not? Why does the design process for systems require so much time and remain so error-prone? What mental processes are involved in understanding another person's design? What information constitutes the essence of a design description? Why, in some situations, does a short passage and a sketch communicate far more than volumes of structured documentation? While complete answers to all of these questions are not necessary for our task, a consistent characterization of the set of "reasonable" answers is needed to effectively define the Design Rationale Capture (DRC) method requirements. In this section, we present our characterization of the nature of design.

The nature of design is characterized by the following points of view.

- 1) Characteristics of the design situation - both the circumstances in which design occurs and the elements constituting an instance of a design process.
- 2) Characteristics of the cognitive skills or "generic activities" of the human designer - including a characterization of the distinguishing reasoning methods.
- 3) Characterization of predominant design strategies under various design situations - arrangements or orderings of the design activities to deal with different design situations.
- 4) Types of knowledge possessed by designers - characteristics of the content and structure of knowledge used to recognize the design situation, perform the design activities, and interact with the related product development processes within a particular organization.

One means of characterizing design is by the type of situation in which it occurs. The following are characteristics of the design situation [Goel 89, Friel 88, Ramey 83] that we have chosen to describe the environment of use for IDEF6.

- 1) Many degrees of freedom - the problem constraints do not determine the solution.
- 2) A large solution space - there may be an infinite number of very good solutions.
- 3) Delayed/limited feedback - the time between commitment or specification to realization of the artifact is long.
- 4) High cost of action - benefits are realized for correct decisions; penalties are paid for incorrect decisions.
- 5) Lack of clear right/wrong decision (evaluation) criteria.
- 6) Input consists of goals and intentions that are generally unclear and subjective.
- 7) Complexity of the problem addressed varies.

- 8) Many, poorly understood parameters or subproblem interactions.
- 9) A large search space - parameters may take on a large number of possible values that could be reasonable in some context.
- 10) Existence of referents (reusable system elements) - many of the problems faced by the designer have already been solved before, at least to some extent.
- 11) Need for producing a specification of an artifact.
- 12) Need for languages of thought to leverage the capabilities of the designer or to communicate the specifications.
- 13) Inherently iterative process.
- 14) Involves learning a mapping of the problem space onto the design space - the iterations are not random trial and error but rather later iterations employ knowledge gained from previous iterations on the same problem.

Another means of characterizing design is by the generic activities that take place during a design process. The set of generic design functions identified in [Goel 89 and Mayer 89] include the following:

- 1) Extensive problem structuring - partitioning, decomposition, establishment of minimal conditions and constraints.
- 2) Fitting of prestructured approaches - use of knowledge of previous problem structures or solution approaches to force structure into the problem and to provide a means of understanding how the current situation maps into previous situations.
- 3) Performance modeling - use of mathematical idealizations designed to reliably predict the performance (steady state or dynamic) of a proposed solution or of the problem situation and its environment to assist in the understanding of each.
- 4) Prototyping - building artifacts that mimic the problem situation or a proposed solution in certain ways in order to evoke

information/decisions from the domain experts or to demonstrate feasibility of a concept.

- 5) Application of personalized stopping rules - completion criteria for level-to-level decision-making, component identification, and determination of the completeness of a design effort.
- 6) Use of a limited commitment strategy - the use of multiple contexts for decisions allowing them to be reversed by elimination of the context.
- 7) Minimal decision-making strategy - a preferred design decision-making strategy that leaves open the maximum number of alternatives as the result of each decision.
- 8) Initiation and propagation of commitments and constraints discovered throughout the design process.
- 9) Solution decomposition into "leaky" modules - toleration of the delay in specification of the components that form the interfaces between proposed modules of a solution.
- 10) Use of hierarchies of idealizations - in general, sets of descriptive models that fit together into a hierarchy such that models lower in the hierarchy can represent all the relations and objects of their parents as well as some additional relations. This allows the quick determination of how more detailed decisions might compromise the commitments made on more global or higher priority issues.
- 11) Artificial symbol systems - special purpose (often graphical) languages for representing critical information that must be identified or managed during the design process.
- 12) Process-driven reasoning - envisionment of courses of events, causality consequences, and enablement relations to determine if the design will "do the right thing."

Design strategies can be considered "meta-plans" for dealing with the complexities of frequently occurring design situations. They can also be viewed as methodizations or organizations of the primitive design activities identified above. Three types of design strategies are considered within the IDEF6.

- 1) External constraint-driven design - design carried out under situations where the goals, intentions, and requirements are not well-characterized, much less defined. These situations often result when the designer is brought into the product development process too early.
- 2) Characteristic-driven design - design in a closely controlled situation in which strict accountability and proof of adequacy are rigidly enforced. These design situations often involve potentially life-threatening situations.
- 3) Carry-over-driven design (sometimes referred to as "routine" design).

Analysis of the life-cycle and generic activities views presented above and our experience indicate that the following knowledge classes are evident in the practice of system design.

- 1) Knowledge of basic principles.
- 2) Knowledge of the general design process in a domain.
- 3) Knowledge of available components.
- 4) Knowledge of previous solution approaches.
- 5) Knowledge of available engineering performance models and workable modeling approaches.
- 6) Knowledge of test capabilities and results (e.g., what sorts of data can be affordably, reliably, or physically acquired).
- 7) Knowledge of the human network (i.e., where the knowledge and information exist in an organization or professional association).
- 8) Knowledge of the requirements, design goals, design decision/evaluation process, and design environment of the current problem.
- 9) Knowledge of political or governmental constraints.

In summary, design as a cognitive endeavor has many characteristics in common with other activities, such as planning and diagnosis. It can be

distinguished by the context in which it is performed, the generic activities involved, the strategies employed, and the types of knowledge applied. A major distinguishing characteristic is the focus of the design process on the creation (refinement, analysis, etc.) of a *specification* of the end product.

3.2 Characteristics of the Design Process

Understanding the characteristics of design "modes"--delineation of major classifications of design activities during the product life cycle--is an important step in understanding the nature of design rational. The design process is a part of a product life cycle. It is a predominant activity early in the life cycle as well as in the sustaining engineering activities after the product has been produced. The notion of a design life cycle is a convenient device often used to help produce an understanding of the basic design processes, particularly for administrative purposes. The design process from such a view is assumed to begin at some point, continue through maturity, and eventually stop. This view of design as a series of incremental and sequentially interdependent steps is an attempt to order the steps of the process in such a fashion that each step can be considered an independent state except for its occurrence relative to the other states that surround it. With its chronologically ordered events, the life-cycle model of design, particularly one embedded within a product life cycle, is an effective tool for understanding the administratively oriented aspects of design evolution. With it, we can understand the role IDEF6 might provide for the project administrator to capture resource-based rationale for design strategy decisions.

The life-cycle design model is only one view needed to understand the IDEF6 requirements because IDEF6 is targeted at the capture of the individual designer's rationalization process which takes on an ad-hoc application of numerous skills (described later). This ad-hoc approach is not chaotic, however, it is certainly not linear as life-cycle models would imply. What the life-cycle partitions do provide is a characterization of the types of specifications that emerge from the design process and a framework for describing the more basic design activities (as certain of those activities predominate certain portions of the life cycle). Figure 1 illustrates the predominate modes of design commonly found in information system development [Ramey 83].

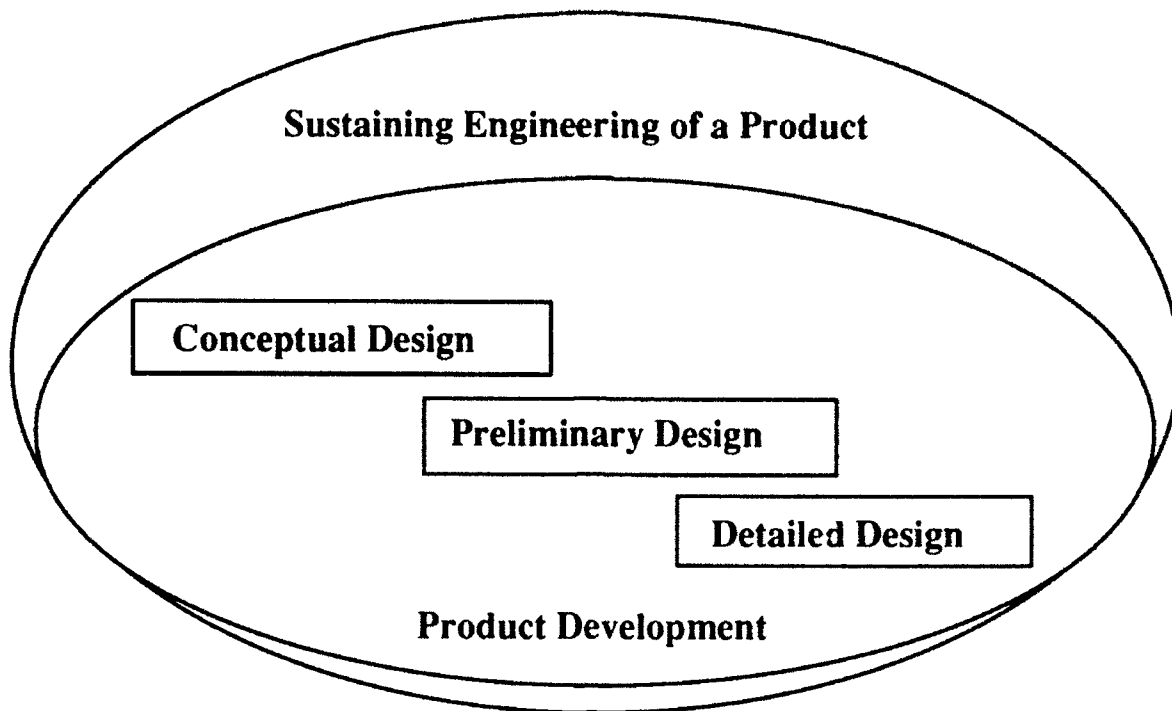


Figure 1. Modes of Design

Depending upon the level of application within the solution artifact, the purpose of conceptual design is primarily the structuring of the problem. This includes both the discovery or analysis of requirements and the identification of the boundaries of the solution space [Friel 88]. The purpose of preliminary design is to separate the promising from the unlikely solutions. Ultimately, the objective of the designer is to select the optimal solution approaches to the problems posed. In the detailed-design mode, the designer undertakes three major tasks [Ramey 83]:

- 1) identify and define interfaces between system elements,
- 2) separate subsystems (elements requiring further structural definition) from components (elements requiring no further structural definition), and
- 3) detail the characteristics that will govern the realization of components.

This is called a detailed-design mode because it deals with the detailing of component definitions. These activities have nothing to do (necessarily) with a particular placement in time--that is, preliminary design is not a "sketchy detailed design" and vice versa. The detailed-design mode is

focused sharply on identification of the detailed characteristics of components to the end that those components may be reliably realized by a supplier.

If we take a strictly technical view of the design process and abstract away the structural influences of the life-cycle model to identify the cognitive primitives (generic activities) of design, this results in what Ramey refers to as "patterns" of behavior. All such patterns are present to some extent in each life-cycle step. However, certain patterns predominate in each step. For example, the following are predominant generic activities in preliminary design.

- 1) Generation of plausible design alternatives (primarily based on historical precedence [Ramey 83, Friel 88]).
- 2) Identification and exploration of the boundaries of the design space (principally the identification of constraints).
- 3) Evaluation of the global performance of the alternatives to select the most appropriate ones for detailed analysis and refinement. If a candidate is found to violate a set of the constraints, the following actions can be taken:
 - a) alter the candidate solution,
 - b) try a different candidate solution, or
 - c) change the original design specifications.

In summary, we characterize the design process as one which involves the continued re-invocation of a set of primitive design modes (patterns of design behavior). In a project context, life-cycle phases may be superimposed on the process; however, at best they characterize the predominant mode during a time interval. In fact, all modes will be exhibited to some extent during the course of any particular life-cycle phase. This conclusion has several implications for the task of capturing design rationale. First, it implies that design rationale will, from the beginning of the design process, involve reference to: 1) structural aspects of the problem, 2) boundaries of the solution space, 3) elements of the solution space, 4) interfaces and subsystems, and 5) characteristics of component realizations. Accommodation of these references will be challenging because the formal life-cycle documentation that would normally house such information will not have been created.

4.0 Design Rationale Phenomena

A general characterization of design rationale is: "The beliefs and facts as well as their organization that humans use to make (or justify) design commitments and propagate those commitments."

In our investigation into the nature of design rationale, we have characterized both "types" of design rationale and "mechanisms" for representation of these types. The types of design rationale identified include arguments based upon the following.

- 1) The philosophy of a design, including:
 - a) process descriptions of intended system operation, and
 - b) design themes expressed in terms of particular object types standing in some specific relations or exhibiting some specific behavior.
- 2) Design limitations expressed as:
 - a) range restrictions on system parameters, and
 - b) environmental factors.
- 3) Factors considered in tradeoff decisions, including:
 - a) level-of-requirements matching,
 - b) project budget or timing constraints,
 - c) general method of doing business in the organization,
 - d) technology available to realize (implement) the design, and
 - e) technology available to test the resulting product.
- 4) Design goals expressed in terms of:
 - a) use or lack of use of particular components,
 - b) achievement of particular structural arrangements,
 - c) priorities on problems requirements,
 - d) product life-cycle characteristics (e.g., disposable versus maintainable, robustness, flexibility), and
 - e) design rules followed in problem or solution space partitioning, test/model data interpretation, or system structuring.
- 5) Precedence or historical proof of viability.
- 6) Legislative, social, professional society, business, or personal evaluation factors or constraints.

Possibly due to the commonness of the carry-over strategy or the complexity of design rationale expression, the most common rationale given for a design is that it worked in a previous situation. Without making judgment on this phenomena, a design-knowledge management capability, as a minimum, must be able to record historical precedence as well as statements of beliefs and rationalizations for why a current design situation is identical to the one the previous design serviced. This phenomena may be less common in software design rationale. The malleability of the medium gives rise to the belief that we can be creative each time. But then we expect the product to perform as reliably as if it were a modification of last year's model; this is unrealistic. Note in software, as contrasted with hardware, creating a new model based on last year's doesn't mean the same thing. In software, the reused parts are literally copied whole, not rebuilt from the same plans, so you lose the opportunity to fix small design flaws, and the interaction of the new parts with the old is likely to be much less well-understood.

Other important rationales for a design are simply, "it feels better" or "it seems more balanced, symmetric." Clearly, there is an important aesthetic side to software design.

Finally, software design rationale includes expectations about how the design will evolve through the development process itself (e.g., how the program structure will probably change). Such expectations do not appear to be as well-defined as similar expectations seen in mechanical hardware design.

The important general conclusion about the nature of design rationale is that it takes the form of a trace of a reasoning process. This trace starts with the element of the design being justified and provides a set of supporting arguments that ultimately "ground" (terminate) to proven elements of the problem or design space. This chain of arguments may also terminate in previously rationalized design elements.

4.1 Preliminary Ontology of Design Rationale

Webster's dictionary [Webster 88] defines ontology as "the branch of metaphysics dealing with the nature of being, reality, or ultimate substance." Practically, an ontology can be viewed as a description of the kinds of things, both physical and conceptual, that make up a given domain and the relationships among them as represented by the terminology in that domain. For example, an ontology for semiconductor manufacturing might describe wafers and reagents and the relationships between them. In

the domain of automotive fasteners, an ontology might address machine-threaded bolts, protective sealants, and machine-threaded bolts with sealant. An ontology can also be viewed as the skeleton or framework for a knowledge-based system. It lays out the concepts, terminology, and structure which will be used to organize specific knowledge in the knowledge base.

The development of design rationale ontology starts with the identification of a set of commonly used terms or phrases that express elements of rationale. An example of such a term is "satisfies"; an example of such a phrase is "is satisfied by." These are used in statements having the following structures.

- 1) Design feature A *satisfies* the requirement B.
- 2) Requirement B *is satisfied by* design feature A.

The next step is to catalogue the intuitive meanings of those terms. Because the nature of design is poorly structured, this is a formidable task. For example, if A satisfies only B, then if B goes away, logically so should A. However, in practice, A may be retained because of feared detrimental side effects or because it is benign. That is, there is a momentum to design; design initiation may be pushed by requirements, but designs tend to take on a life of their own and may resist continued responsiveness to requirements. This may or may not be detrimental in specific cases; there can be benign unresponsiveness to changes in requirements. Such complexity in the process is naturally reflected in the semantics of the terminology used in that context.

When dealing with a term that specifies a relation, delineation of the types of arguments that can be used with that term is another important part of the ontology. In the "satisfies" relation, the house of quality notion would prescribe that only *user* requirements (alias constraints) make their way into requirements documents. But design is also constrained in many other ways--hardware/software constraints, imagination limitations of designer, etc. The characterization of all types of constraints that really affect a design and hence can participate in a "satisfies" relationship is likely to be a formidable task. Similarly, it is nontrivial to determine what constitutes a "design feature" in a program design, a database design, or an information system design. Possible collections of such "features" might include the following.

- 1) Any of the design concepts (e.g., class, feature, slot, function) representable in IDEF4.

- 2) Programming language constructs. (Should these ever be part of a design rationale or should only logical design terms be used? When programmers record design rationale (or change rationale) they will most likely want to use programming language terms. If the design notation used closely tracks language constructs, it might be possible to do an auto-translate for the record. This gets into the whole problem of maintaining threads of rationale from requirements through maintenance.)
- 3) Schema definitions for a database system.
- 4) Procedure definition (either automated or manual).
- 5) File descriptions, window layouts, menu items, etc.

The meaning of the term "satisfies" is also complicated by the number of individual concepts participating in a use of the term. For example:

- 1) design features A & B & ... & G satisfy requirement Z,
- 2) design feature B satisfies requirement Y, and
- 3) design features A & H satisfy requirement X.

This type of complexity affects the difficulty of updating rationale as well as using rationale information to assist in extricating unneeded features if one of the requirements goes away.

As described in the previous sections, designs evolve. One challenge in the development of an ontology for design rationale is to determine the manner in which the interpretation of statements can change as the objects referred to in them mutate over time. For example, in the following typical sequence note that of the "satisfies" relation may be sensitive to substitution of equals (or near equals?) for objects that stand in that relationship over time.

- 1) Design feature A satisfies requirement Z.
- 2) Requirement Z evolves into requirement Y.
- 3) Design feature A is irrelevant to requirement Y.
- 4) Design feature B replaces design feature A.

Another facet of the semantics of design rationale is dealing with the modality and belief aspects of the objects (both constraints and design features) with which such rationale statements must deal. The difference between stated requirements, unstated but actual (e.g., hardware) requirements, and perceived requirements must be delineated. Perceived requirements may come and go quickly in the grey area between requirements analysis and design. For example, we thought we needed a

status-line data structure in a particular knowledge-based designer window because we perceived a requirement for maintaining a certain collection of information. Further problem analysis revealed that the required decisions could be based on a simpler set of information that required no additional data structure, so the status line was removed.

Characterizing the concepts and processes associated with system organization/partitioning in the rationale formulation process will play an important role in the ontology of design rationale.

Software systems are often represented using different, orthogonal representations as part of the design process (e.g., structure charts, Program Description Language (PDL) descriptions). However, within a representation, different partitionings are not normally given, unless as design alternatives. A design rationale, however, may reference an unlimited number of views that subdivide a design in different ways. "If you look at it from this point of view" would be a common comment. The point of view taken will in no way depend on some "natural" way of subdividing the system; rather it will depend on at least:

- 1) the purpose of the rationale giver (RG),
- 2) the background understanding of the RG, and
- 3) the RG's perception of the background understanding of the rationale receiver (RR). (Note a problem here: in general, the RG will not know (or even be able to take a good guess at) the background understanding of the RR).

If there is opportunity for feedback from the RR (in which case he/she may express lack of understanding), one strategy the RG may use is to change the partitioning--"Let's look at it from a slightly different point of view." Thus there is no invariant frame of reference.

One might suppose, initially, that partitioning may be different if the purpose of the design rationale is documentation versus explanation. However, documentation may be viewed as a form of generally directed explanation. The typical length of the documentation is a direct reflection of the documentalist's inability to make any assumptions about the background understanding of the audience. In an automated system in which user profiles are available, more assumptions might be made which could increase the succinctness of effectively communicated design rationale statements.

Considering the implications of dynamically shifting system-partitioning gives rise to several salient issues. Suppose we have partitionings, P1 and P2, of a system, S, and rationales, R1 and R2. Such partitionings are collections of design features.

- 1) If R1 explains P1 and R2 explains P2, where P1 and P2 are overlapping partitionings of S, then is there any sense in which R1 and R2 combine? What are the consistency requirements between R1 and R2, if any? Or is the requirement just that R1 and R2 not be inconsistent? How do, or may, requirements for consistency vary over time?
- 2) If R1 explains P1 and R2 explains P2, where P1 and P2 are non-overlapping partitionings of S, then is there any sense in which R1 and R2 combine? Are there any consistency requirements? (Note that there are likely interactions that would imply consistency requirements but which are not obvious from the given partitionings. Another partitioning may be required to adequately show the interactions.) How do, or may, requirements for consistency vary over time?
- 3) If P3 is a new partitioning at time T2 that involves parts of A and B, and if rationale R3 explains P3, then what are the consistency requirements between P1 or P2 and P3? R3 may or may not reference R1 and/or R2.
- 4) Assume constraints (user requirements and other constraints) are denoted Cx and design features are denoted Fx. In the case of R1 = (F1 satisfies C1 assuming P1 at T1) and R2 = (F2 satisfies C1 assuming P2 at T2), since C1 didn't change, one can conclude that either F2 or F1 satisfies C1 and that the choice of P2 may have affected the choice of F2 over F1.

Other terms/phrases that must be considered in an ontology of design rationale include:

- 1) system,
- 2) subsystem,
- 3) component,
- 4) requires/is required by,
- 5) constrains/is constrained by,
- 6) bounds/is bounded by,
- 7) supports/is supported by,
- 8) creates/is created by, and
- 9) translates/is translated by.

4.2 Issues with Design Rationale Capture

One of the reasons for the loss of rationale is partly rooted in the long lag time between specification of the software artifact and its completion.

There are basic problems with the **Kind of Thing** that Design Rationale is. A notable difficulty with the expression of design rationale is that it (as a concept) exhibits a resistance to being uniformly understood. It shares this characteristic with all other forms of "explanation" that Artificial Intelligence (AI) researchers have tried to deal with over the last 20 years.

One of the problems with the capture of design rationale is that it requires the statement of characteristics beyond the minimum specifications required to produce the product. Since the major goal of design has traditionally been the construction of specifications for artifacts so complete that any realization of them will satisfy the requirements (and thereby solve the problems), the underlying logic of the decisions that contributed to, led up to, or resulted in such a design description is not normally recorded. After all, their inclusion into the traditional document structures used to record the design artifacts may cause confusion or at best complicate the acquisition/interpretation of the critical information being communicated by these artifacts. In addition, as noted by Friel [Friel 88] and Goel [Goel 89], a designer may make hundreds of focused component decisions or, through interpretation of test results, may implicitly make thousands of configuration decisions in a very short time. Lack of efficient methods for the capture and representation of these decision alternatives and considerations is a primary impediment to the capture of design rationale.

5.0 Preliminary IDEF6 Concepts

Moving beyond text capture and association for the support of design rationale capture becomes quite complex. We approached controlling this complexity by examining the typical questions a user might ask about a design knowledge base. The information required to answer these questions then becomes requirements for the Design Rationale Capture (DRC) "method" to capture and manage. We partition these into several types of questions about: (1) specified artifact composition and structure, (2) object purpose or function relative to the intended behavior of the artifacts, (3) causality/enableness characteristics of the established relations between individual objects or subsystems, (4) supportability of particular beliefs used as rationale for design decisions, (5) the design process (how it was

planned and carried out), and (6) device behavior or failure (the proverbial "What if ..?" questions).

In general, our approach uses several strategies based on a situation theory structured description of the evolving design and its situation (objects in relations in situations which are in planned involvement relations with other situations). Type 1 questions are answered through information access of the materials design bill. The key to Type 2 questions is the recognition of the difference between purpose and function. At an initial level, function can be explained by knowing the dependence or independence of an object in the device relative to the environment of operation of the device [Forbus 84]. At a more detailed level, knowledge of inter- and intra-state phenomena is required. Questions about purpose require knowledge of the events in the planned operation of the device and then an association of an object's existence with its role in those events. Supportability questions can be answered by collecting and displaying the set of support from problem and first principle premises (or assumptions) to the fact in question [Hobbs 86]. In this area, a major role is played by engineering-discipline-specific "thematic abstraction units" (commonly agreed upon assumptions) to establish grounding conditions for many assumptions [Dyer 83]. Questions about the design process itself can be addressed with the use of qualitative simulation and planning techniques, as in Allen [85] and Wilensky [83]. Finally, the answer to "What if" questions appears to be relegated to simulation (either qualitative or quantitative) [Kuipers 84, Laughton 85].

One strategy we might pursue based on the situation model is to cast rationale as the formation of involvement relations between a designer in a particular decision-making situation and one of a number of different "constraining" situation types. The types of constraint situations identified so far include:

- 1) Conventional Constraints - e.g., historical precedence, societal (both marketing and professional);
- 2) Nomic Constraints - e.g., Laws of nature;
- 3) Necessary Constraints - e.g., model-based (analytic or ontologic);
- 4) Requirements Based Constraints - e.g., customer or contractual requirements;
- 5) Goal Based Constraints - e.g., customer, project team or personal; and
- 6) Resource Constraints - e.g., time, skills, manpower, money.

6.0 The IDEF6 Automation Support Environment

The IDEF6 task addresses the problem of capturing design rationale. As such, it must not be associated primarily with code documentation but rather must be a method usable during requirements analysis and design. The goal, of course, is to produce an automated capability for the recording and subsequent browsing of design rationale. To be effective, such a capability would necessarily have to be embedded within a comprehensive analysis/design environment since design rationale typically evolves piecemeal. It is possible that IDEF6 will evolve into a series of adjuncts to other modeling tools rather than being a stand-alone method. Thus, IDEF4 models, for instance, might be extended (or annotated) with IDEF6 elements.

- 1) Tool should have on-line capability to reference constraints (i.e., software requirements + resource constraints + business or legal constraints).
- 2) IDEF5 models of concepts for various design artifact types will be useful (e.g., relational databases, data flow diagrams, IDEF4 models, C system programs).
- 3) Design issues regarding automated support:
 - a) Unobtrusiveness
 - b) Generality across design tools
 - c) Ontology usage and development
 - d) Need something like a system document examiner
 - e) Environmental dependency - Must IDEF6 itself be configurable as an IDSE would be?
 - f) Use with versioning system?
 - g) Capability to use information in it to help generate documentation. (Note that this is different from documentation in that consistency with the implementation is not maintained.)
 - h) Should the IDEF6 "tentacles" extend into tools that are primarily used for requirements development? (There is sometimes a fine line between requirements analysis and design.)
- 4) Tool should support:
 - a) design rationale capture
 - b) design rationale browsing

- c) design rationale usage (clipping) in report/documentation production
- 5) Need expectations (or knowledge base) of conventional organization patterns. A knowledge base would probably need to be built for each tool/formal representation (e.g., IDEF1) .
- 6) Sources of rationale input:
 - a) context
 - b) parsable language
 - c) text or graphical annotations

7.0 Unallocated Thoughts, Comments, Ideas, Issues, etc.

- 1) What is the level of abstraction in software design (analogous to points-and-lines vs. features in geometric modeling)? Ideally, software design occurs at the logical structure level rather than at the language level. But we suspect that software design frequently occurs at the language level.
- 2) Should IDEF6 reference strictly software program design? Database design?
- 3) How should IDEF6 relate to the Life Cycle Artifact database proposed for the IDSE?
- 4) What is the relationship of IDEF6 to a version control system?
- 5) Further research is needed to determine the degree of similarity between design approaches used by different designers for the same problem. This research would probably be related to the "routineness" level. Particular attention should be paid to how the designer's level of concern with code generalization affects the design decisions.
- 6) IDEF6 must be able to handle the "why not" questions as well as the "why" and "how."
- 7) The software development process should be expanded to include a semiformalized way of recording overall design rationale.

8.0 References

- Allen J. F., "Towards a General Theory of Action and Time," *Artificial Intelligence* (23), 1985, pp 123-154.
- Dyer, M., "In-depth Understanding, A Computer Model of Integrated Processing for Narrative Comprehension," MIT Press, Cambridge, MA, 1983.
- Forbus, K., "Qualitative Process Theory," *Artificial Intelligence* (24), pp 85-168, 1984.
- Friel, P. Griffith, "Modeling Design Reasoning in Automotive Engineering," PhD Dissertation, 1988, Texas A&M University.
- Goel, V. Pirolli, P., "Motivating the Notion of Generic Design with Information Processing Theory: The Design Problem Space," *AI Magazine*, Vol 10, No. 1, Spring 1989.
- Hobbs, J., "On the Coherence and Structure of Discourse," in "The Structure of Discourse, L. Polanyi Ed. Ablex Publishing Corporation, Norwood, NJ, 1986.
- Kuipers, B., "Commonsense Reasoning About Causality: Deriving Behavior from Structure," *Artificial Intelligence* 24, pp. 169 - 123, 1984.
- Laughton, J., "Qualitative Reasoning in Mechanical Design," Technical Report #XXXX, Department of Computer Science, University of Texas, Austin, TX, 1985.
- Mayer, R. J., A. A. Keen, and C. J. Su "Design Knowledge Management System," SBIR Phase I Final Report, 1989.
- Ramey, T. L., "Guidebook to Systems Development," Internal Research Report, Hughes Aircraft Co., El Segundo, CA, 1983.
- Webster, "Webster's Ninth New Collegiate Dictionary," Merriam-Webster Inc, Springfield, MA, 1988.
- Wilensky, R., "Planning and Understanding, A Computational Approach to Human Reasoning," Addison-Wesley Publishing Company, Reading, MA, 1983.