THE CONSTRAINTS AND ASSUMPTIONS INTERPRETATION OF SYSTEMS DESIGN: A DESCRIPTIVE PROCESS MODEL

William C. Sasso
and
Monte McVay

October 1986

Center for Research on Information Systems
Information Systems Department
Graduate School of Business Administration
New York University

__Working Paper Series__

CRIS #137

GBA #86-102

DRAFT VERSION
Please do not cite or quote.

ABSTRACT

The largescale ineffectiveness of current systems development methodologies may be attributed to the inaccuracy or inadeaucy of their underlying assumptions concerning the systems development process. In this paper, we propose a descriptive, alternative model of the Information Systems (IS) design process. This model emphasizes the importance of constraints in defining the feasible design space, and of assumptions as a vehicle for discovering constraints. Moreover, rather than assuming that design activities occur in a logical and prescribed sequence, as the current dominant model, the Systems Development Life Cycle (SDLC) does, the Constraints/Assumptions (C/A) Model focuses on the interdependent nature of design activities.

The importance of developing and validating alternative models of the system design process is evident from three sources. First, there is the paucity of empirical research on systems design, which we attribute to a scarcity of theory to guide such research. Second, educators evince serious doubt as to our ability to educate students in this process. Third, the widespread inability of professional systems designers to develop systems on schedule, within budget, and providing the full set of specified functions is disconcerting, if not appalling.

Previous research suggests that superior designs are produced when both clients and designers regard the IS design process as a learning experience, and work to educate each other. The Constraints/Assumptions Model further elaborates this mutual learning thesis, by differentiating what clients learn from designers and what designers learn from clients. The C/A Model asserts that, at any stage in the design process, two dialogues occur simultaneously. The client/designer dialogue elaborates the design space, i.e., a set of constraints on the design process specifying required performance and function, the organizational and political climate, the resources available for developing and operating the system, etc. The designer/team dialogue, on the other hand, focuses on the generation of a working solution to the design problem, its validation with respect to technical feasibility and its congruence with the acceptable design space, and its elaboration into an implementable design. Both the design space and the working design are inputs to each dialogue, and their interdependence results from each dialogue's ability to modify only its own product.

The current state of the systems development process in most organizations raises serious doubt as to whether we can justifiably speak of an "art" of systems design, let alone such a "science." Consistently characterized by serious development backlog, poor performance in terms of the continued completion of projects over-budget, beyond schedule, and with fewer functions than originally intended, and resulting designs which often resemble Rube Goldberg creations, the practice of systems design remains in a clearly unacceptable state. Educators evince serious doubts as to our ability to educate students in this process [23]. The common failure of systems to meet user expectations is a possible explanation for the recent downturn in investments in computer hardware and software. For instance, Nutt [19] notes that

> Disillusionment [of executives with MIS] has led to MIS redesign, replacement, and, in extreme instances, the disbanding of entire departments. Much of the difficulty with the MIS has been laid at the designer's doorstep, suggesting that MIS design practices merit careful study to determine their value. [p. 139]

In the small research literature on systems design, there exists only a minimal amount of rudimentary theory of the process, which may account, in turn, for the paucity of empirical research which has been reported on the topic. In short, our current knowledge concerning the nature and practice of systems design may be accurately compared to the medieval alchemist's understanding of chemistry.

In the absence of knowledge, primitive societies frequently use ritual as an appropriate response to adversity. Ritual may be conceptualized as a set of symbolic practices performed in a prescribed manner to achieve desired results. This description might be applied to many of the responses to the shortcomings of our systems design processes, such as the Systems Development Life Cycle (SDLC) concept, and many of the structured development methodologies which have been promoted as "facilitating" it. Another common response of primitive society is to "externalize" the responsibility for dealing with the crisis, by invoking the appropriate deity. In like fashion, we see systems developers beseeching the omnipotent "Technology," in the form of automated systems development tools, or turning to such unlikely powers as the users themselves, via "End-User Computing," to deliver us from our unhappy fates. Unfortunately, neither ritual nor invocation of deities seems to have reduced the application backlog or improved our ability to design and implement systems to a significant degree.

This paper will structure and, where appropriate, contrast, our current understanding of the systems design process and related knowledge about problem solving. We will delineate serious shortcomings in the existing models of the process, and present an alternative model, which emphasizes the central importance of constraints in defining the "design space" and of assumptions as a tool for discovering constraints. Moreover, this model underlines the interdependence of different design processes, rather than assuming that they occur in discrete steps in a a clearly delineated sequence. We will then relate the Constraints/Assumptions (C/A) Model to the research literature on systems design, and discuss its implications for research on, education regarding, and the practice of systems design.

In the following section, then, we will review the empirical literature on systems design and that on creative problem solving (mainly from Cognitive Psychology). We will also present briefly some of the current "wisdom" on systems design, as propounded by experienced designers and developers of systems design methodologies. The third section will present the Constraints/Assumptions Model, both defining its elements and their interrelationships and exemplifying its application to understand a typical systems design scenario. The fourth section will discuss the support provided to the C/A Model by existing research findings, and the fifth will outline its major implications for research, education, and practice.

What Do We Know About Systems Design

Our current understanding of the systems design process comes from three main sources:

1. Empirical research on Information Systems (IS) design;
2. Thoughts and perspectives on IS design, from both academic and practitioner sources; and
3. Empirical research on the general topic of creative problem solving.

This section will review relevant work from each of these sources in turn, classifying work from the first two categories along three dimensions: scope, problem definition, and development phase. Scope refers to the size and complexity of the design activity investigated; i.e., does the design activity have ramifications for the entire system (e.g., defining system boundaries) or is it focused on a particular component, such as the design of a program, given a high-level system design as a constraining framework?

The degree of problem definition ranges over a spectrum from well-defined to ill-defined, with those end-points being more easily described than the spectrum's intermediate positions. We can perhaps best characterize well-defined problems by stating that the existence and general form of the solution is known, that the constraints imposed by the situation are explicit, and that a known set of operators exists which can be applied to the current state to achieve a set of successive transitions to a demonstrably correct goal state. At the other end of the spectrum, none of this information may be available. We are ignorant not only of the goal-state, but also quite possibly of the current state. The fact that a global set of operators exists is of little use, because we are generally ignorant of the relevant set of situational constraints which governs the applicability of those operators. IS design occurs across the entire spectrum, with program and subroutine design being characteristically well-defined, and system delineation at the outset of a project typically very ill-defined. We feel it important to differentiate IS design research according to the degree of definition present in the design activity it studies.

Our third classification, development phase, relates the focus of each study to the SDLC phase in which it most frequently occurs. We characterize the SDLC phases as: Requirements Definition, Design, Development, and Implementation.

Empirical Research on IS Design

In general, there is a relatively surprising scarcity of this type of research, given the growing importance of Information Systems in our society and the unacceptable state of our systems development capabilities.

In one of the earliest empirical studies of IS design, Boland [2] investigated the relationship between different styles of client/designer interaction and their impact on the resulting design. In the first interaction protocol, characterized as "traditional," designers familiarized themselves with the problem independently from the clients, and developed a structured interview which formed the basis for their interaction with clients. In the alternate interaction protocol, clients and designers were encouraged to "educate each other" and work together in deciding where the system development effort should focus. Boland concluded that the designs produced did differ according to the interaction protocol used — that the traditional protocol produced computer-centered designs, in which the degree of control allocated to clients was reduced, while the alternate protocol generated more client-centered designs, enhancing the responsibility of the clients.

Several years later, Jeffries et al [13] compared experts and novices as they designed a computer program, given a problem definition and set of objectives. By collecting think-aloud protocols, they hoped to develop a general theory of computer software design. They considered decomposition central to successful design, and their analysis supports this, with experts showing a more sophisticated ability to decompose a problem than novices. They hypothesize that experts have a "design schema," i.e., a knowledge of how the design process works, that drives the process of decomposition.

Carroll et al [4] investigated how the presentation of the problem to designers affects the solutions they create. Subjects received a booklet describing the same problem in one of four different ways. These descriptions differed in the degree of hierarchy they made explicit and in the sequence they presented system requirements. Designs were evaluated with respect to the number of requirements they satisfied, the time required to complete them, the degree to which they matched the problem, and the degree to which they were revised over the course of the experiment. Their results suggest that methods which make problem structure explicit and encourage the designer to consider problem structure during the design process will produce designs that better reflect the problem structure and have a greater degree of stability over the design process.

Also in 1980, Malhotra et al [15] published a report describing three distinct experiments on the design process. In perhaps the most important, they collected and analyzed client/designer dialogues. They also studied an architectural design problem and an IS component design problem. In the client/designer dialogue experiment, Malhotra and his colleagues found that new requirements were often made explicit during the evaluation of a potential solution, and that the structure of dialogues seemed to iterate between designers' requesting information from users and designers' presenting possible solutions for evaluation by users. In the restaurant design experiment, the researchers noted a correlation between (a) the originality or practicality of the design and (b) the subject's expressed goal of creating an original or a practical design. Based primarily on the client/designer dialogue experiment, Malhotra et al propose a three-phase model of the design process, consisting of goal elaboration, design generation, and design evaluation.

In a key expert/novice[1] study, Vitalari and Dickson [26] used think-aloud protocols to study problem solving behavior in a requirements determination context. They observed differences in the expert/novice in so far as:

1. Experts appeared to search more for particular "clues" (trigger behavior);
2. Experts rejected working hypotheses more frequently;
3. Experts verbalized and modified their general problem-solving strategies more frequently;
4. Experts more frequently set specific, intermediate goals (i.e., they decompose the design problem); and
5. Experts understand the importance of problem facilitation.

Both groups divided their time consistently, with about 25% being spent on the acquisition of information and about 75% on structuring the problem. This study's results seem to underline the important roles played by analogical reasoning and prior experience in design. Like the study conducted by Carroll et al, Vitalari and Dickson's results suggest the importance of approaching and structuring the problem in an appropriate manner.

In 1985, Vitalari [25] reported an analysis of a second set of data from the previous experiment, in which he coded the protocols to determine the degrees to which expert/novice designers referred to different knowledge content areas. While all subjects were concerned with the system's functional requirements, error control mechanisms, documentation, and organization structure, experts focused more on report definition, physical implementation, design team interaction establishing user involvement, and political issues. From this, Vitalari proposes a design knowledge base model in which core domain knowledge is surrounded by knowledge of the organization, function, application, techniques and methods, and expert/novice differentiating knowledge.

Sasso [22] compared the effects of two strategies for bounding an analysis situation, using the transcript

[1]Technically, Vitalari and Dickson compared "high-rated and low-rated analysts," rather than "experts" and "novices."

experiment technique. While noting differences in the information acquisition activities of the two groups, he did not observe the predicted differences between the sets of descriptions and recommendations produced by the different groups. Sasso did observe an interesting phenomenon, that the more completely the subjects documented the existing system, the lower was the value of their recommendations. He terms this the descriptive/prescriptive tradeoff, and suggests that it deserves further study.

The relatively small number of empirical studies on IS design per se appear clustered in two corners of our "Research Space." According to Table 1,[2] about half focus on ill-defined, requirements definition problems at the system level, while the remaining half deal with well-defined design problems, split fairly evenly between the subsystem and program levels.

Thoughts and Perspectives on IS Design

Contrary to the scarcity of empirical research on IS design, there exists a large and ever-growing body of literature which presents the ideas of academics and practitioners on how systems design should be done. The practitioner publications, especially, can be meaningfully differentiated as formal versus informal, with former more frequently "selling" techniques[3] associated with their authors' perspective on systems development, while the latter tend to share experiences which lead to the development (and in some cases, the personal validation) of the author's viewpoint. While the informal are often more complete in their assessment of the problems associated with system design, neither the formal nor the informal --- nor the academic, for that matter --- can provide any rigorous data substantiating the efficacy their perspective and/or techniques.

Weber [28] and Turner [23, 24] are good representatives of the academic "insights and perspectives" literature. Weber reviews the empirical research on structured problem solving, and relates it to IS design processes, emphasizing the importance of functional decomposition. Her paper is essentially a call for

[2]Tables and Figures will be found following the Bibliography, at the very end of the paper.

[3]And in some cases, automated development tools, instructional seminars, and consulting services.

research into the nature of design, based on the theories we can apply from problem-solving research. Turner, similarly, stresses the need for better models of IS design, and suggests the use of engineering design principles as a possible "goal state template."[4] He argues that design is a creative process, constrained on the one hand by a general set of design principles, and on the other by the specific objectives and constraints of the actual design situation --- a "creative sandwich" model of design. Turner is heavily influenced by Malhotra et al [15], and suggests that the SDLC fails to consider the presence of much iterative activity in design and the importance of experience in the process.

Gould and Lewis [10] present three key principles of design derived from studies done at IBM's Thomas Watson Research Center:

1. Establish an early focus on users and tasks;
2. Conduct empirical studies of user interaction with the proposed system; and
3. Design in an iterative fashion.

Gould and Lewis conducted an informal survey of designers, and noted that only 16% of the respondents mentioned all three of these basic principles, even under a very liberal coding scheme. Furthermore, a more detailed analysis of the context in which the principles were mentioned suggests that they often were presented as afterthoughts, or desirable options, should the development schedule permit.

Among the formal practitioner works, Yourdon and Constantine [29] have received a great deal of attention and form the basis of a number of curricula in IS Development. They propose coupling and cohesion[5] as the two key characteristics of a design and articulate a fairly wide-ranging set of techniques whose application (they argue) leads to the creation of systems with low coupling and high cohesion. The importance of there two characteristics was discovered in a private study done by Constantine in Hughes Aircraft and IBM, in which he found them highly correlated to the cost of system maintenance. Unfortunately, this study was never publicly reported. The Yourdon and Constantine analysis seems heavily biased toward the processing activity in the system, rather than its data structures.

[4] "Goal state template" is our phrase, not Turner's.

[5] Coupling reflects the degree of interdependence between program modules, and cohesion the degree of interdependence within a single module.

Warnier [27], on the other hand, takes just the opposite perspective, arguing that data is (or should be) more central than process. His argument is, in essence, that for a given organization, there is an appropriate data structure most capable of supporting its decision making processes, and another data structure which represents the data that is currently available to it from the environment. Processes are used to transform the latter into the former --- they are merely a means, while the data organization itself is the end.

Martin [16] and Case [5] argue for the automation of steps in the SDLC wherever possible, suggesting that this will lead to improved system development productivity and enhanced quality of the resulting systems. Martin goes further, suggesting that we can retire the traditional SDLC, replacing it with enterprise modeling, somewhat akin to Warnier's data-oriented approach.

A final class of formal practitioner works is the systems development manual, for example [20]. These manuals, developed in-house by major systems development and systems consulting organizations, attempt to present detailed sets of rules, instructions, and/or guidelines whose application will increase both the efficiency of the systems development process and the quality of the system produced. Many of these development manuals provide a fairly detailed set of procedures, with their accompanying forms and worksheets.

The informal practitioner thoughts and perspectives works tend to discuss much more the problem, and even the context of the problem, rather than the all-purpose solution. In Brooks' classic work, The Mythical Man-Month [3], he emphasizes the importance of learning and change as the design is created, and stresses the need for communication within the development team. Brooks argues strongly for the presence of a chief architect, to make the decisions which will ensure that the evolving design retains as much conceptual integrity as possible. He asserts that certain aspects of the design and development process cannot be effectively performed in parallel, and argues that it is counterproductive to assign additional staff to a project which is behind schedule.

Fox [9] concurs with Brooks regarding the importance of a single "chief" designer and the highly iterative nature of the systems development process. Fox argues that design remains more an art than a science, and goes Brooks one better by suggesting that the development process really never ends --- that we should regard "maintenance" as "ongoing development" of the evolving system. He sketches out a three-tiered model of the design process, with the macro process defining modules of capability, the second-level process reducing these modules into programs and subroutines, and the lowest level process creating or specifying the programs/subroutines themselves. Fox notes the difficulties and complexities associated with the development of an abstract product like software, and argues that the choice of design techniques and tools is contingent upon the specific design context.

Mills [17] compares the design process to a funnel, in which the users' requests enter at the top, and are successively transformed into requirements, an architecture, a system design, and a detailed design before emerging from the funnel as actual code. Mills believes that the design team should be organized with an overall system architect supervising the work of a set of system designers. He sees the architect's major design responsibilities as:

1. Defining the system's high-level components and formulating their interfaces;
2. Ensuring consistency of design, primarily by reviewing the component designs prepared by systems designers; and
3. Designing for evolution, both limiting the scope of the first release, and by providing the necessary "hooks" to which additional features may be attached in the future.

Discussing design work done in a fairly different systems context, that of computer games, Crawford [7] makes some interestingly similar points. He offers seven design precepts, several of which underline the importance of considering the hardware on which the system is to be implemented early in the design. Several others stress the importance of maintaining the unity of design effort and keeping the design "clean" i.e., developing the game around a small but powerful and general set of processing rules which embody the game's theme and ensure its "artistic unity."

Because of the extreme size of the practitioner insights literature, only a representative set of works has been discussed in this section. Even from this small sample, however, we note a consensus among academics and researchers that far too little has been done to understand the basic process of systems design, and a puzzling absence of publicly-reported scientific evidence supporting the claims made by proponents of formal methods of systems development. Perhaps most striking, however, is the convergence of the opinions presented by Brooks, Fox, and Mills, in terms of the importance of a chief designer or systems architect in ensuring the overall integrity of a design, and regarding the centrality of learning and iteration to the successful design process.

Empirical Research on Problem Solving

Several of the studies discussed earlier, most notably Malhotra et al , Vitalari, and Weber have referred to the significant body of research on general problem-solving capabilities[6] and suggested its potential value as a model for research into the nature of systems design activity. Most of this research has investigated human behavior in solving fairly well-defined problems, such as winning chess games, solving physics problems, and proving theorems. Because we feel that the general system design process is better characterized as ill-defined, we will review here a smaller body of less well-known research on creative problem solving, i.e., problem solving in an ill-defined problem context.

Reitman [21] initiated this line of research in a study in which he collected think-aloud protocols from a composer as he created a fugue for piano. Reitman characterizes attributes of the problem as constraints upon the problem solution, and suggests that, in an ill-defined problem context, these attributes form very general constraints initially. As the composition process continues, supplementary constraints are created, further limiting the acceptable solution space, but the composer can consciously choose to violate these, should he uncover a possible design component that violates existing constraints but has positive implications for the problem solution overall. The initial constraints, which define the problem itself, cannot be violated without changing the fundamental problem. One cannot, that is to say, create a fugue for piano to be played on violin, or create a fugue for piano which does not correspond to the general form of a fugue. Within these basic constraints, however, the design activity may frequently be discontinuous, in the sense that the current design-state and the plan for its completion may often fail to imply the next activity that will be undertaken.

[6]For a summary of this body of work, see Chi et al [6].

Greeno [11] reports a study which investigated the occurrence of indefinite goals as intermediate states in the solution of well-defined problems. Using protocol analysis, he observed high-school students as they proved a geometry theorem concerning the congruence of triangles. He noted that students did not proceed directly to proving the theorem, but rather studied the problem rather generally at first. Because the initial problem focus was one of pattern recognition, Greeno suggests that pattern recognition can frequently occur as a first step comprising an ill-defined goal in many well-defined problem contexts. Similar pattern recognition activities are likely to be involved in the studies of IS design at the program level, such as those reported by Jeffries et al and Carroll et al.

The relevance of work such as Reitman's to understanding the systems design process should be obvious. Perhaps equally obvious is the extreme degree of difficulty involved in successfully isolating the creative act so that it can be meaningfully studied, which may account for the small number of studies we are able to review in this section. Nonetheless, the importance of their insights into the problem-solving process is very great --- when combined with those of the informal practitioners, they form the basis for the Constraints/Assumptions Model of Systems Design, to whose presentation we now turn.

The Constraints/Assumptions Model

Consider the following client-designer dialogue:

C: I want the personal computer to be able to communicate with the mainframe so I can call up information on customer balances.

D: Why do you need a personal computer for that?

C: Well, we've got some spreadsheet reports we use.

D: But we have IFPS on the mainframe.

C: I don't know IFPS.

D: But we could train you ...

C: No. No. Listen. I really just want a PC.

D: So it's a requirement.

C: Yes.

D: So we can configure a PC connection to the mainframe, but you won't get instantaneous access ...

C: What do you mean ...

D: I mean we don't have a way for you to access the data base on-line through the PC.

C: Why can't I just do a transmission, that would take a few minutes but, ... well ... that wouldn't do ...

D: Does the information have to be current?

C: Well ...

D: What if we transmitted down a file of customer balances to you every morning, then you could access the balances all day but they'd be effective as of that morning.

C: Well. How would I access them once they were transmitted?

D: Simple. You'll have a menu that will come up, automatically, when you turn the machine on and it will ask you if you want to enter Lotus, or transmit customer balances, or inquire against customer balances...

In this hypothetical, but very typical[7], scenario, we observe a design process rather different from that suggested by the SDLC Model. The designer's statement that "we can perform functions A and B with software C, so long as we have hardware components D and E and the users can handle F..." is a clear intermingling of the "logical design" and "physical design" stages adamantly differentiated in the Yourdon Structured Development methodology [29, 8]. The discussion of costs and training implications violates the tenets of the SDLC, where only long after preliminary design has been determined user training considered. These incongruities suggest that either our scenario is atypical, or that the SDLC and Yourdon Models of the systems design process bear limited relationship to reality.

The Constraints/Assumptions Model, on the other hand, provides a simple and elegant explanation of this dialogue. The designer has three primary objectives. First, he needs to delineate the feasible design space, by identifying the many constraints (e.g., "I really just want a PC." scenario) which apply to the design. Logically subordinate to this, but generally done simultaneously, he needs to partition this design

[7]The reader may wish to compare it with the dialogues presented by Malhotra et al [15], for example.

space into a set of design problems of manageable scope, so far as possible both mutually exclusive and exhaustive of the design space. Third, the designer needs to test possible solutions against the current set of explicitly defined design space constraints as well as any implicit ones which have not yet surfaced in discussion. Here, for example, the required degree of currency of customer balance information is tested.

We turn now to a formal presentation of the Constraints/Assumptions Model of Systems Design. We begin by specifying the elements of the model, and follow this with a delineation of the more important relationships between these elements.

Elements of the Model. Among the central elements of the C/A Model, we include constraints and assumptions, solutions, implications, designers, clients, the design space, the design concept, and the design itself. The Model's key input factors include the client, the designer, and the independent constraints, those regarded by the designer and client as beyond their individual or mutual control. The client and the designer both bring additional constraints to the process, but these are dependent constraints, at least to some degree manipulable by the client and/or designer. Designers and, to a lesser extent, clients, also bring to the process solutions, familiar design components[8] from which a design may be created. Experience, and to a lesser degree, professional education, are the primary sources from which solutions arise. The designer thus comes to the design process equipped with a set of solutions, any of which may be highly applicable or totally inapplicable (or somewhere intermediate in its applicability) to the design problem at hand. The broader the designer's experience, the more solutions he carries in his "bag of tricks", and the larger the number of design situations he recognizes as familiar in the sense that he has applied a known solution in an analogous situation to achieve acceptable results in the past. An experienced designer, moreover, has not only more solutions available than does the novice one, but can also more efficiently determine the applicability of each in a particular design context. Furthermore, he is more advanced in his ability to delineate/demarcate the design space.

The design space, as its name implies, is the area within which solutions are considered feasible --- not necessarily good, but acceptable. The design space can be conceptualized, as shown in Figure 1, as

[8]These might include, for example, computational algorithms, data structures and access methods, communications protocols, hardware configurations, etc.. depending to a large degree on the particular design problem.

consisting of the region inside of a number of "frontiers." These frontiers are composed of sets of related constraints. The performance/function frontier, for example, might consist of constraints that specify particular processing and reporting capabilities, their associated mean response times, acceptable levels of system reliability, and peak transaction loads the system is expected to handle. A partial listing of other frontiers would include the organizational/political, the development resource, the operating resource, the available technology, and the physical/cultural/geographical ones. The design space, then, is a very complex concept, composed of intermediate units called "frontiers." Frontiers are themselves complex, often being composed of many potentially important and relevant constraints, and further complicated by the fact that different constraints may be in either explicit or implicit conflict with one another. This can be true for constraints which are components of different frontiers (e.g., low development budgets may make it impossible for the system to achieve the desired "non-stop computing" specified in the performance standards) as well as constraints which from parts of a single frontier (where desired error-checking capabilities might conflict with required response time standards).

The designer and client thus face several critical and challenging activities in the design process:

1. Delineation: the delineation of the feasible design space, often through the explicit statement of constraints;

2. Identification/Resolution: the identification and resolution of logically conflicting constraints;

3. Decomposition: the decomposition of the design space into a set of fairly independent design problems, of both manageable size/complexity and (ideally) familiar to the designer;

4. Generation/Evaluation: the generation and evaluation of solutions, which can help make explicit additional constraints, through the presentation of explicit assumptions, which specify the capabilities of a solution and its implications and their evaluation relative to known and unknown constraints;

5. Integration: the integration of different solutions into a design, in which the implications of the various solutions are tested against each other, their conflicts are resolved, and their aggregate implications are tested against the design space; and

6. Formulation/Distillation: presentation of the overall design, in a form intelligible to the users and systems development team, and useful to the latter in terms of guiding decisions in the detailed design, coding, testing, and deployment stages.

Several important observations should be made regarding these activities. First, they do not occur in a sequential fashion, but are all active simultaneously and interdependently. Resolution can only occur after a certain amount of delineation has occurred, but delineation cannot be complete until resolution has been done. Similar mutual precedence anomalies exist between decomposition and generation/evaluation and, indeed, between almost any pair of these six processes. Even formulation/distillation has the implicit capability to generate questions concerning decisions and tradeoffs made earlier, in light of its more complete understanding of the overall design. Brooks' admonition that we should "plan to throw one away" is a more extreme statement of this assertion [3].

The second key observation is that the execution of these processes produces a design, rather than a completed system. What is a design? We see a design as a model, or a visualization, of the arrangement of certain elements (i.e., solutions) in such a way as to provide means to accomplish specific desired purposes subject to the constraints of the design space. A design not only lists the various pieces of the system; it shows us how the pieces fit together, and that their assemblage does not violate the design constraints. We follow Turner [24] in his description of the component elements of a design, including the design concept, the boundary, the division of labor between man and machine, the design architecture (describing the system's configuration of hardware, software, and data), and operating and control procedures.

Third, the assumption-testing process, in which explicit assumptions are presented to users in order to identify and test constraints is ubiquitous throughout these activities. The general form of an assumption is "In order to achieve *a*, we can apply solution *b*, which means that *c, d, e,...,*." Here *a* is a set of one or more performance/functional constraints, *b* is a set of one or more solutions, and *c,d,e, ...,* are implications. Implications are specifications of general constraints in light of specific solution. The designer's familiarity with the particular solution enables him to enumerate its requirements as implications, and these, in turn, can be used to gauge the robustness of their corresponding constraints.

Consider the following assumption as an example. "If you want the capability to do serious simulation modelling on a personal computer, and you're not planning to go on vacation while the model runs, you can use PCSIM, which requires a math processor, a hard disk, and at least 512k of memory in your PC." Here, the goal-set consists of capability (simulation) and performance (response-time) constraints, the solution is a software package (PCSIM), and the implications concern a number of hardware components (math processor, hard disk, 512k memory). If the user accepts this assumption, these implications become elevated to the status of constraints themselves, as does the solution. For example, PCSIM may not exist in an Apple compatible form, which implies that the hardware frontier has become more restricted. Similarly, the optional hardware implications of PCSIM have implications for our budget. If we have a strong preference for the MacIntosh, or if purchase of these additional hardware components cause us to violate an inflexible budget constraint, we return to the original assumption, and ask whether another solution might generate a more acceptable set of implications.

At this point the reader may ask "What part does creativity play in the design process?" From our perspective, creativity does not enter the design process neatly, fitting in at a particular point in a certain process. Rather, we see creativity as permeating each of the six interdependent activities noted above, to a greater or lesser degree. The extent to which this occurs is largely determined by the interaction of the designer's experiences with the design space --- if his experience includes an overall solution obviously adaptable to the current design space, very little creativity will be needed. If, on the other hand, the design situation is entirely foreign to the designer, practically every aspect of the design process will require creative thinking. Most designer/design space combinations will fall somewhere between these two extreme cases, i.e., the design situation will include components both familiar and foreign to the designer. Minimally, creative thinking will be required to decompose the foreign elements, generate and evaluate potential solutions to them, integrate their solutions with the overall solution, and formulate and distill the resulting design.

Creativity, of course, can enter the design process via an altogether different channel, being not so much mothered by necessity as by serendipity. This occurs when the designer perceives a familiar design space in a strikingly new way, in terms of one or more of the six design activities. Perhaps he perceives a major advantage to shifting the design space's boundary, or develops an alternative means of resolving a conflict between important constraints. Perhaps he comes to see significant advantage stemming from a decomposition he had considered bizarre at first glance. Though they occur infrequently, spontaneous creative interventions such as these can occur in any of the design activities, and their impact will ripple through the others.

Relationships Between the Elements of the Model

Figure 2 depicts the most important relationships between elements of the Constraints/Assumptions Model of Systems Design. The user and designer together define the dependent constraints and the design concept. They also work together to identify relevant independent constraints, i.e., those beyond their ability to change. The union of the dependent and independent constraints defines the design space. The design space and design concept lead to the design, in terms of the overall solution generated by the designer and accepted by the designer and user.

As should be evident from the Figure, the client and designer collaborate to create at least four intermediate products before the system can be built. These include the design concept, the design space, the "working design," and implementable design itself. While it might seem logical that these can be produced in a purely sequential order, with delineation of the design concept and space preceding that of the working design, and that in turn preceding the design itself, we will argue that such a purely sequential approach is both inaccurate and inadequate. Malhotra's client/designer dialogues, for example, provide strong evidence that the presentation of solution components is an excellent device for uncovering unstated design constraints. Thus, although we will distinguish the processes which lead to these intermediate products, we assert that these processes are parallel, interdependent, and mutually iterative, rather than occuring in a well-ordered sequence.

Macro design deals with the discovery of the design space itself. Since part of the design space is defined in an arbitrary fashion by the client and designer (i.e., the dependent constraints), the space itself can be changed during the design process, should a superior overall solution present itself. The goal of macro design is to define a logically consistent design space, in which user-suggested solutions are reduced to their underlying requirements, the completeness of the available set of constraints is assured, and any logical contradictions between them are resolved. It is perhaps in macro design that meaningful client participation is most important, because most constraints are (or can be) stated in language intelligible to clients, and because it is the short-term path of least resistance for the designer to define his own set of assumptions about constraints, convenient to work with if not necessarily accurate in describing the design context.

A second aspect of the macro design process deals with the joint effort of the client and the designer to articulate the design concept, similar to the "kernal" discussed by Fox and the "system concept" described by Turner. Turner notes that

> In OS/360 (IBM) the design concept was *complete*: one common operating system would support the company's complete line of computers and that system would have a complete set of features. While JCL permits almost infinite adjustment and configuration of the operating system, it is complicated, time consuming to learn, and difficult to use. Another design concept ([e.g.] user friendly) would have produced a different solution...

Other examples of possible design concepts would include:

- "state of the art," implying an ambitious attempt to reach very high performance/functional goals while allowing great flexibility with respect to the technological, development resource, and development schedule frontiers;
- "ease of use," implying a commitment toward the operating resource frontier;
- "least disturbance to existing systems," suggesting the primacy of existing technology constraints in the technology frontier;
- "system reliability/availability," implying a focus on performance, possibly at the cost of relaxing development and operating resource constraints; and
- "quick and dirty," leading to a concern for development budget and resource frontiers, with some attention paid to functional constraints, and a general disregard of other frontiers.

Both these macro design activities, delineation of the design space and identification of the design concept,, are ill-defined processes. They involve an unknown number of direct and indirect tradeoffs, many of which are non-quantifiable and subjective. Moreover, they often involve negotiation between different clients and units of the organization, and are sometimes essentially political processes. A visual presentation of the effect of different design concepts on the overall process is shown in Figure 3, "The Design Concept as the Axis of Design."

At the intermediate design level, we are concerned with generating potential solution components to the design problem, and testing their implications against the current boundaries of the design space, against the design concept, and against the design constraints implied by other solution components which are currently part of the working design. If the component seems to conflict with one or more of these previously defined elements, it may be discarded. Alternatively, if the component appears to lead to a superior overall design, and the previous elements with which it conflicts are under the control of the client/designer, it may be the previous elements which yield. Thus, the intermediate design process focuses on decomposition of the problem into familiar subproblems, the generation and evaluation of solutions to the subproblems, and the integration of the solution components into a design satisfying the general problem.

At the micro design level, we begin to approach the state of well-defined problem solving. Here we are concerned with "fleshing out" the working design, by selecting the data structures and access methods, processing algorithms, and control structures with which to implement our working design. But micro design activity can and sometimes must begin as soon as an initial decomposition has occurred at the intermediate level. Only micro design can validate the feasibility of a working decomposition. Previous experience, of course, can validate designs which have been used before, but the critical elements of a novel decomposition require a micro design validation.[9] Micro design activities can also influence the design space, minimizing the effects of conflicts between elements of the design space through elegant programming, or exacerbating unseen ones through inelegant work.

As is shown in Figure 4, Design Effort Allocations Over the Development Process, we believe that macro and intermediate design activities continue throughout the entire development process, and that micro design begins very shortly after the project begins. Naturally, the relative proportions of effort devoted to each change over the course of the project.

To use an analogy, we might compare these three design processes to the construction of a dam. [10] First, in the process corresponding to macro design, we select the river site and construct a series of

[9]For example, a recent NY Times article discusses how programming difficulties are forcing a reshaping of the Strategic Defense Initiative ["Star Wars"] program. The article notes that computer experts have agreed that " ... any large-scale cluster of weapons and sensors will have to be organized and shaped in accordance with the limitations of computer software" [1]. This has resulted in re-design of the overall approach, from an efficient, highly centralized control system to a loosely coupled set of autonomous battle stations.

[10]The major flaw with this analogy is that the dam-building activities are clearly more sequential.

temporary cofferdams which define (and drain) the area in which our permanent dam will be built. This dry area corresponds to the problem space. Then, after examining the riverbed and adjusting our plans, we construct a series of forms which define precisely the shape our dam will have, corresponding to the intermediate level. Once this is done, we can pour the concrete, fleshing out our dam the way our micro design work fleshes out our working design, transforming it into the implementable design.

What kind of design process can integrate macro, intermediate, and micro design activity simultaneously as shown in Figure 4? At any chronological point in the design process, we argue that two design-oriented dialogues exist in parallel. In one, the client and the chief designer attempt to define and ensure the consistency of the design space and the design concept. Thus, the larger part of macro design activity occurs in these dialogues, which can be quite challenging due to the complicated nature of the design space and the frequency of conflicting positions being taken by different clients. As Malhotra's study shows, intermediate design activities also occur in these dialogues, with the presentation of a potential solution often facilitating the discovery of new elements of the design space or design concept.

The second design-oriented dialogue is that between the chief designer and the members of his/her design team. Here, taking the client-accepted design space and design concept as given, the emphasis is on decomposing the problem and generating potential solution components, the central activities of intermediate design. Moreover, even early in the process, critical micro design activities occur, in order to validate the feasibility of untried components of the working design. The client/designer dialogue is primarily concerned with making explicit the acceptable design space and preferred design concept; the designer/team dialogue focuses on the generation of a working design and the validation of its novel components. This symbiotic relationship is shown in Figure 5, A Cross-Cut Through the Design Process.

As the comparison of Figures 4 and 5 suggests, over the course of the overall systems design process, the relative emphasis accorded these two dialogues shifts from an initial predominance of client/designer dialogue, with its essentially macro design focus, to a heavy preponderance of designer/team dialogue with a concomitant increase in micro design, towards the close of the design phase.

Taken literally, the design process shown in Figure 5 suggests an infinite loop. Because the working design is continually being elaborated, however, until it reaches a degree of detail sufficient for it to be considered an implementable design, what we have is less an infinite loop than a spiral. On the other hand, given a working design at a particular level of this spiral, several iterations at that level will probably be required to ensure its consistency with the client-accepted design space and design concept before it can move down the spiral to the next more detailed stage. Thus, after achieving congruence with the design space and concept via several iterations at a given level of detail, the working design makes a "quantum leap" downward, to the next more detailed stage of design, where again several iterations of the dual dialogue will transpire. Thus downwards progress toward an implementable design is achieved through iteration at a given stage, and results in movement to a new stage not drastically different from the previous one. Through very gradual changes in the design activity, not stark ones such as Logical Design to Physical Design or High-Level Design to Detailed Design, is the implementable design generated.

Finally, we note that Figure 5 suggests at least two critical research issues. One concerns the issue of appointing a single chief designer (versus several "responsible" designers) in organizing the design team. The necessity of maintaining internally consistent expressions of the client-accepted design space and concept and the designer/team-accepted working design appears to favor the "chief designer" organization. Using several responsible analysts introduces a macro design element into the designer/team dialogue, increasing its already sufficient complexity. On the other hand, employing business analysts as intermediaries between the client and the designer introduces both a third dialogue (analyst/designer) and an additional intermediate product, the analyst/designer agreement, whose consistency with client-accepted design space/concept and the working design must be validated.

The second major research issue concerns the nature and form of communication between the two ongoing dialogues. Most systems development methodologies and approaches do not differentiate the two distinct communications our model depicts. Presentation techniques such as Data Flow Diagrams, HIPO Charts, and Warnier-Orr Diagrams are proposed as basis of communication between the clients and design teams, with no differentiation of what is being communicated from whom to whom. If our model of the design process is accurate, the very different nature of these two types of communication may be best conveyed using different representational techniques. This issue, like that of organization of the design team, cries out for empirical investigation.

Discussion and Conclusions

Having presented the C/A Model of IS design, we will now proceed to elaborate on its implications for practice, research, and education in IS design. We will then close the paper with a summary.

Implications for Practice

The C/A Model of IS design has four major implications for the practice of information systems development. First, it argues that the fundamental interdependence of logically distinct stages in the life cycle necessarily precludes their efficient and effective performance in a strict sequential pattern. In other words, if the model is correct, complete and accurate requirements definition can be far more easily achieved if it is performed in tandem with construction of a working design. The design and its components play a valuable role in helping the client articulate implicit constraints. Thus, while many presentations of the SDLC approach discuss the iterative nature of individual stages, such as Requirements Determination, the C/A Model implies that iteration should exist across stages such as Requirements Determination, high-level design, and Detailed design.

Secondly, we feel the C/A Model has implications concerning the involvement of clients (or users) in the systems development process. Long a controversial issue among systems professionals, user involvement has been strongly promoted [14, 18], but a recent review has concluded that its efficacy in increasing system quality has not been demonstrated [12]. What may be going on is that users are being involved in micro-design activites, outside their expertise and experience, where their contributions are likely to be minimal. Indeed, in a development project using the SDLC approach, once the Requirements Definition stage is completed, what else is left to involve the users in, besides technical design issues?

Third, the C/A Model provides a theoretical basis for the "chief designer" or "systems architect" design team organization which experienced practitioners such as Brooks, Fox, and Mills have long recommended. The importance and complexity of maintaining congruence between the design space and the working design argue for its being the clear-cut responsibility of a single, very capable individual, because otherwise the patterns of communication and conflict resolution are likely to become unmanageable. Moreover, some current practices, such as assigning business analysts to do the Requirements Definition, and system designers to do high-level design, and programmer analysts to develop and implement detailed program and data structure specifications, clearly increase the communication effort required to develop a system and are likely to reduce its conceptual integrity. If the C/A Model is correct, the "divide and conquer" strategy has some very clear limits with respect to IS design.

Finally, we will argue that the C/A Model provides an alternative, and superior, basis for system documentation. In current practice, what documentation exists is often a hit-or-miss affair. By creating and maintaining an index relating the constraints and assumptions which relate to each system, subsystem, data structure, and program, we should be able to provide the maintainer with a far more complete and meaningful understanding of the context within which the subsystem, data structure, or module was created and intended to operate.

Implications for Research

Each of the implications for practice discussed above is currently little more than an assertion, and their empirical validations are important research issues. Even more important, however, is the investigation of the critical components of the model itself. We express them as the following set of strategic propositions.

Successful[11] system designers will:

1. more clearly differentiate the nature of the client/designer and designer/team dialogues throughout the design process;
2. shift the relative emphasis accorded these two dialogues over the course of the design process, from an initial focus on the client/designer dialogue to an ending one on the designer/team

[11]Alternatively, "experts" or "high-rated" system designers.

dialogue;

3. allocate greater effort (early in the process) to delineation of a consistent design space and identification of the design concept, and will involve clients in this to a larger extent;

4. accord a high (perhaps the highest) priority to maintaining congruence and design activity between the intermediate design products throughout the design process;

5. will exhibit a lower (weaker/smaller) tendency to "chop" their design activities into the conventional, distinct, SDLC phases; and

6. will demonstrate a lesser concern for SDLC deliverables, and a greater one for the intermediate design products and the implementable design.

These propositions could, of course, be related to successful IS designs or successful system implementations, rather than to designers. Developing our ability to distinguish successful and unsuccessful designs, prior to their implementation, is a critical conceptual research issue in itself. Furthermore, since the success of an implemented system depends not only on the success of the design but on the success of the implementation (and the general constancy of the environment), we feel it most appropriate to relate these propositions to the designer as an individual.

To prove or disprove the Constraints/Assumption Model, we need to determine the validity of these propositions via the collection and analysis of empirical data. This can be done via both field research in organizations and in laboratory experimentation. For field research, we will need to identify both successful and unsuccessful designers and elaborate these propositions in terms of the data items that must be collected an studied. Ideally, we would triangulate this data collection process, using surveys and structured interviews with both clients and designers, and studying both the working and final design documentation. Where possible, the actual observation of client/designer and designer/team dialogues would be very valuable.

Laboratory data collection, possibly in the form of think-aloud protocols or transcript experiments, can provide more rigorously controlled investigation of hypotheses derived from these propositions. Moreover, experiments which determine the relative efficacy of different representation schemes[12] in client/designer

[12]This could involve, for instance, the comparison of Data Flow Diagrams with Systems Flowcharts as a basis for client/designer dialogue, or the relative evaluation of program flowcharts and structured english in the designer/team dialogue.

and designer/team dialogues can make a valuable contribution by substantiating or invalidating the claims made by proponents of systems development methodologies and procedures. Indeed, these investigations are important independent of the validation/invalidation of the C/A Model itself.

Implications for Education

Until the C/A Model has been empirically validated, it has few immediate implications for educating future systems designers, but its potential implications are important. Here the critical questions are "What skills do we currently develop in systems design courses?" and "What skills should we develop in these courses?"

Most systems analysis and design textbooks focus their greatest emphasis on the SDLC, techniques for micro design and representation, and preparation of deliverable documents, such as feasibility studies. Lesser, but significant, coverage is generally accorded to cost/benefit analysis, scheduling and managing the overall development process, training users, and testing and evaluating the system. Generally explicitly, the textbooks accept the SDLC picture of systems design as a fairly clearly defined sequence of a series of distinct activities: Feasibility study, Requirements definition, High-level design, Detailed design, Coding and unit testing, System testing, and Deployment.

Perhaps even more striking than the consistency of what is present in systems analysis and design texts is the consistency of that which is absent. The failure to consider communication skills, the lip-service paid to the need for iteration and meaningful client participation, and the rare discussion of such essential design concepts as the design space, the design concept, or congruence between the working design and the design space/concept are all conspicuous. Even worse, only very infrequently does a text explicitly address the ill-defined, creative process of design itself. From studying a text on systems analysis and design, one might logically conclude that the most challenging parts of the process are design of input screens and definition of report layouts.

These are important issues in micro design, but we argue that building the right system is at least as important as building the system right. Macro design deals with the former, and micro design with the latter. Since the SDLC perspective dominates the macro design section in most texts, validation of the C/A Model will imply that new texts (or drastic revisions in most existing ones) will be required.

Extrapolating from the content of textbooks to what occurs in systems analysis and design classes is risky business, of course. Nonetheless, we would argue that these issues are inadequately treated in many sytems analysis and design courses, and that the validation of the C/A Model will be an impetus for the redesign of many such courses.

Summary

Practitioners and researchers in IS design are increasingly coming to recognize the inadequacy of our predominant model of IS design, the SDLC. We have proposed an alternative model of the IS design process, the Constraints/Assumptions Model, which explicitly recognizes the interdependent and simultaneous relationship of different design activites, which the SDLC attempts to perform in a strict, sequential order. The C/A Model's intermingling of macro, intermediate, and micro design activities is accomplished via a cycle of two parallel and ongoing dialogues. In the client/designer dialogue, the constraints and assumptions which govern the system to be developed and the development process itself are articulated and conflicts between them are resolved. The model terms these macro design activities. Moreover, working designs and their components are tested against them, to ensure that the designers/components do not violate critical constraints and assumptions. As previous research has shown, this comparison also helps articulate previously unexplored design requirements. The second of the two dialogues, the designer/team dialogue, focuses on intermediate and micro design activities. This involves proposing a working decomposition of the system, validating the technical feasibility of its critical components, and elaborating it into an implementable design.

We feel the C/A Model has strong implications for practice, reserach, and education in the area of IS design. While the accuracy of the model remains to be investigated in scientific fashion, so does that of the currently accepted SDLC Model. We look forward to seeing the results of empirical studies comparing the C/A and SDLC Models, and hope that, in the meantime, our model's implications receive a fragment of the devotion currently accorded the SDLC's prescriptions.

Acknowledgments

The authors wish to express their gratitude to Ms. Carole Butler, for her prompt, patient, and very valuable assistance in the preparation of this manuscript.

References

[1] Boffey, P.M.
Software Seen as Obstacle in Developing 'Star Wars'.
New York Times , September 16, 1986.

[2] Boland, R. J., Jr.
The Process and Product of System Design.
Management Science 24(9), May, 1978.

[3] Brooks, F.P., Jr.
The Mythical Man-Month.
Addison-Wesley, Inc., Reading, Massachusetts, 1975.

[4] Carroll, J.M., L.A. Miller, J.C. Thomas, and H.P. Friedman.
Aspects of Solution Structure in Design Problem Solving.
American Journal of Psychology 93(2):269-284, June, 1980.

[5] Case, A.F., Jr.
Information Systems Development: Principles of Computer-Aided Software Engineering.
Prentice-Hall, Englewood Cliffs, NJ, 1986.

[6] Chi, M.T.H., R. Glaser, and E. Rees.
Expertise in Problem Solving.
In Sternberg, R.J. (editor), *Advances in the Psychology of Human Intelligence*, pages 7-75.
Erlbaum, Hillsdale, NJ, 1982.

[7] Crawford, C.
The Art of Computer Game Design.
Osborne/McGraw-Hill, Berkeley, CA, 1984.

[8] DeMarco, Tom.
Structured Analysis and System Specification.
Yourdon Press, New York, 1978.

[9] Fox, J.M.
Software and Its Development.
Prentice-Hall, Englewood Cliffs, NJ, 1982.

[10] Gould, J.D., and C. Lewis.
Designing for Usability: Key Principles and What Designers Think.
Communications of the ACM 18(3):300-311, March, 1985.

[11] Greeno, J.G.
Indefinite Goals in Well-Structured Problems.
Psychology Review 83(6):429-441, November, 1976.

[12] Ives, B., and M. Olson.
User Involvement and MIS Success: A Review of Research.
Management Science 30(5):586-603, May, 1984.

[13] Jeffries, R., A.T.Turner, P.G.Polson, and M.E. Atwood.
The Process Involved in Designing Software.
Technical Report SAI-80-110-DEN, Science Applications, Inc., 1980.

[14] Lucas, H.
Why Information Systems Fail.
Columbia University Press, New York, 1969.

[15] Malhotra, A., J.C. Thomas, J.M. Carroll, and L.A. Miller.
Cognitive Processes in Design.
International Journal of Man-Machine Studies 12:119-140, 1980.

[16] Martin, James.
An Information Systems Manifesto.
Prentice-Hall, Englewood Cliffs, NJ, 1984.

[17] Mills, J.A.
A Pragmatic View of the System Architect.
Communications of the ACM 28(7):708-717, July, 1985.

[18] Mumford, E.
Designing Participatively.
Technical Report, Manchester Business School, Manchester, England, 1983.

[19] Nutt, P.C.
Evaluating MIS Design Priniciples.
MIS Quarterly 10(2):138-155, June, 1986.

[20] Peat, Marwick, Mitchell.
Systems Development Manual.
, 1980.

[21] Reitman, W.R.
Creative Problem Solving: Notes from the Autobiography of a Fugue.
In Reitman, W.R. (editor), *Cognition and Thought: An Information-Processing Approach.* John Wiley and Sons, New York, 1965.

[22] Sasso, W.C.
An Empirical Comparison of Analysis Perspectives.
1986.
In preparation.

[23] Turner, Jon A.
Understanding Elements of System Design.
In R. Boland and R. Hirscheim (editor), *Critical Issues in Information Systems Research.* John Wiley and Sons, New York, 1986.
Forthcoming.

[24] Turner, Jon A.
The Process of Systems Design: Some Problems, Principles, and Perspectives.
1986.
In preparation.

[25] Vitalari, N.P.
Knowledge as a Basis for Expertise in Systems Analysis: An Empirical Study.
MIS Quarterly :221-241, September, 1985.

[26] Vitalari, G. W., and G. W. Dickson.
Problem Solving for Effective Systems Analysis: An Experimental Exploration.
Communications of the ACM 26(11):948-956, November, 1983.

[27] Warnier, J.D.
The Logical Construction of Systems.
Van Nostrand Reinhold, Inc., New York, 1979.

[28] Weber, E.S.
Cognitive Processes Involved in Solving Information System (IS) Design Problems.
In *Proceedings of the Fourth International Conference on Information Systems.* Houston, Texas, 1983.

[29] Yourdon, T., and L.L. Constantine.
Structured Design.
Yourdon Press, New York, 1978.

TABLE 1

EMPIRICAL RESEARCH ON SYSTEMS DESIGN

REQUIREMENTS DEFINITION

DEGREE OF DEFINITION

	WELL-DEFINED		ILL-DEFINED
SYSTEM			BOLAND, SASSO VITALARI & DICKSON VITALARI MALHOTRA ET AL (C/D DIALOGUE & RESTAURANT DESIGN)
SUBSYSTEM			
PROGRAM			

DESIGN

DEGREE OF DEFINITION

	WELL-DEFINED		ILL-DEFINED
SYSTEM			
SUBSYSTEM		CARROLL ET AL	
PROGRAM	JEFFRIES ET AL MALHOTRA ET AL (INQUIRY MODULE)		

Figure 1

The Design Space and
Its Component Frontiers

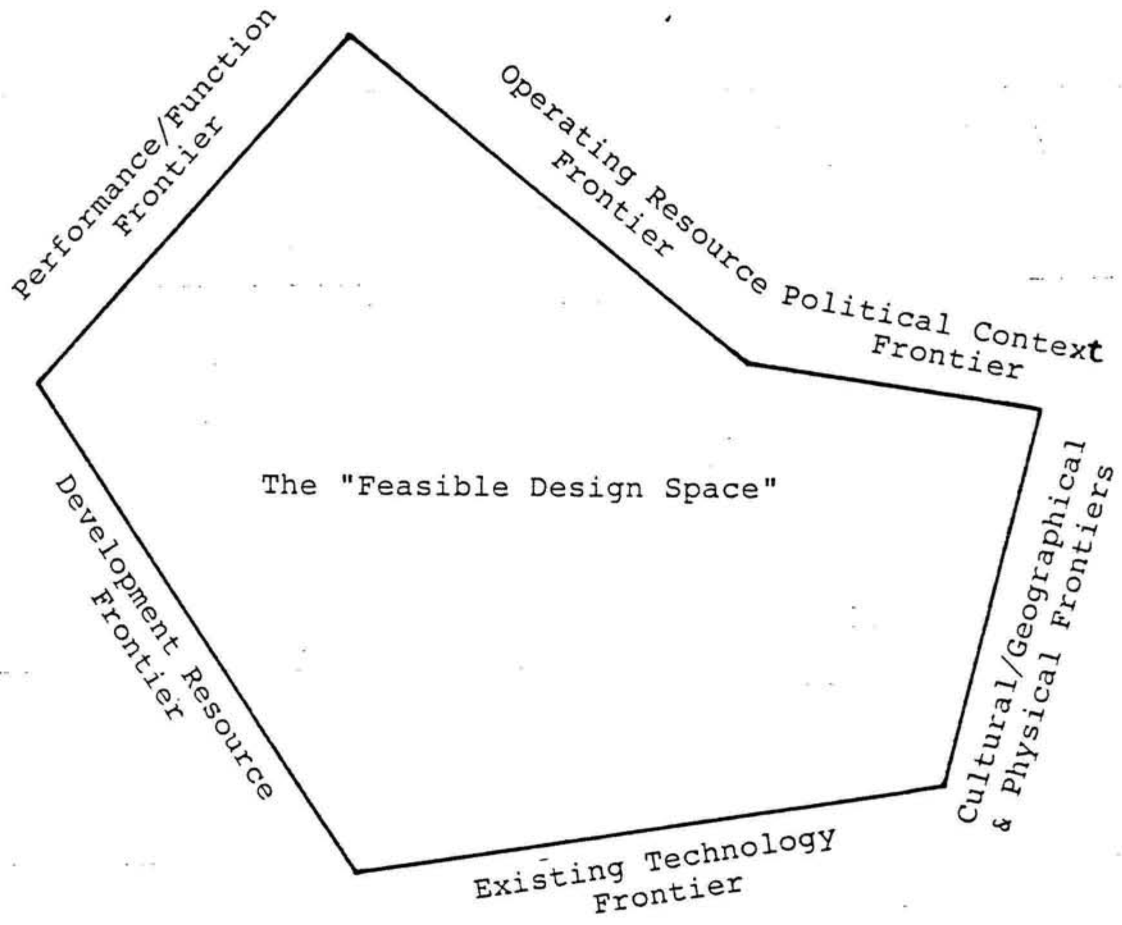


Figure 2
C/A MODEL

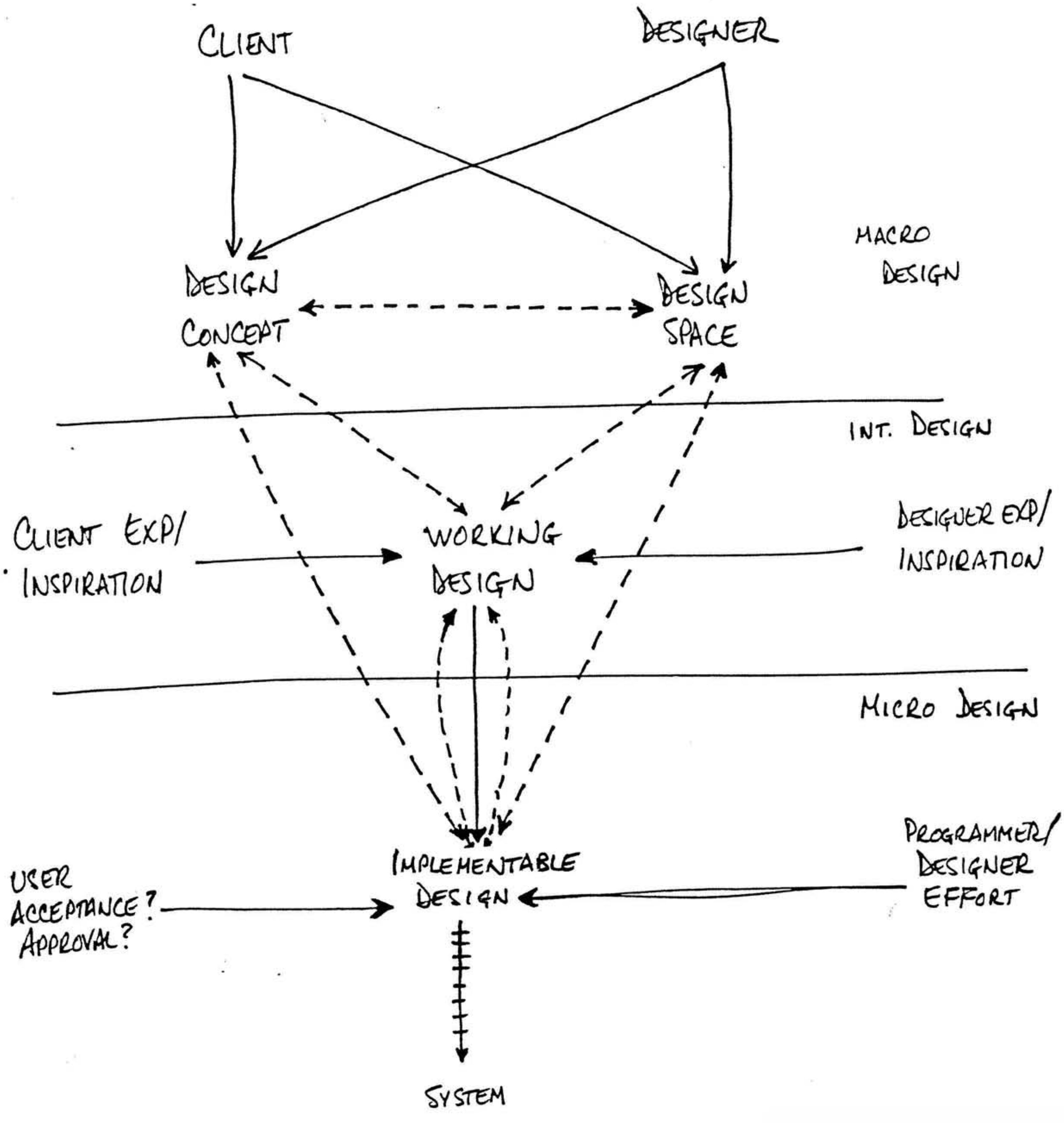
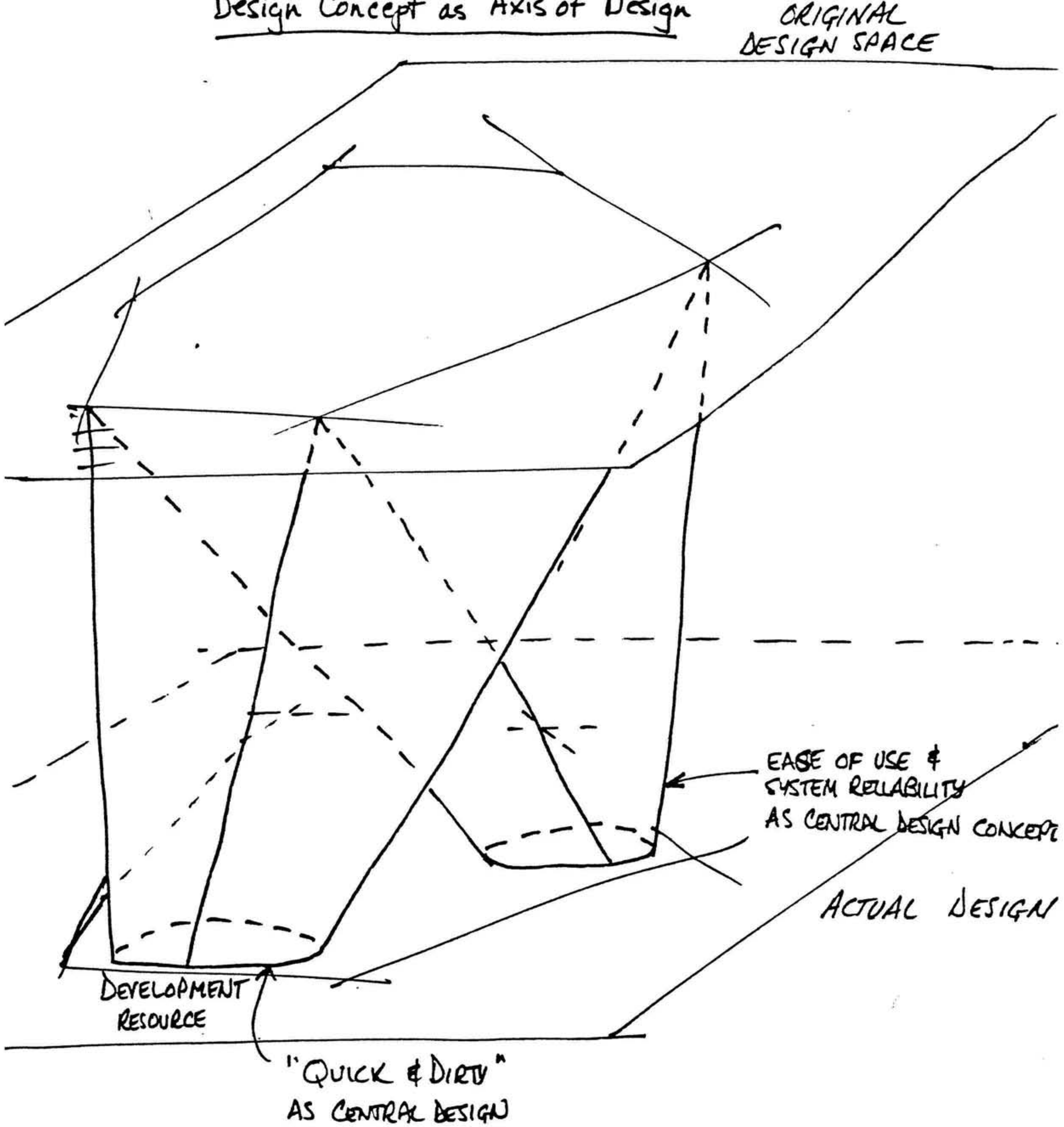


Figure 3

Design Concept as Axis of Design


ORIGINAL DESIGN SPACE
EASE OF USE & SYSTEM RELIABILITY AS CENTRAL DESIGN CONCEPT
ACTUAL DESIGN
DEVELOPMENT RESOURCE
"QUICK & DIRTY" AS CENTRAL DESIGN CONCEPTS

OTHER POSSIBILITIES: "STATE OF THE ART" "COMPLETE FUNCTIONALITY"
"HIGH TRANSACTION VOLUME"
"LEAST DISTURBANCE OF EXISTING S

Figure 4

DESIGN EFFORT ALLOCATION OVER THE DEVELOPMENT PROCESS

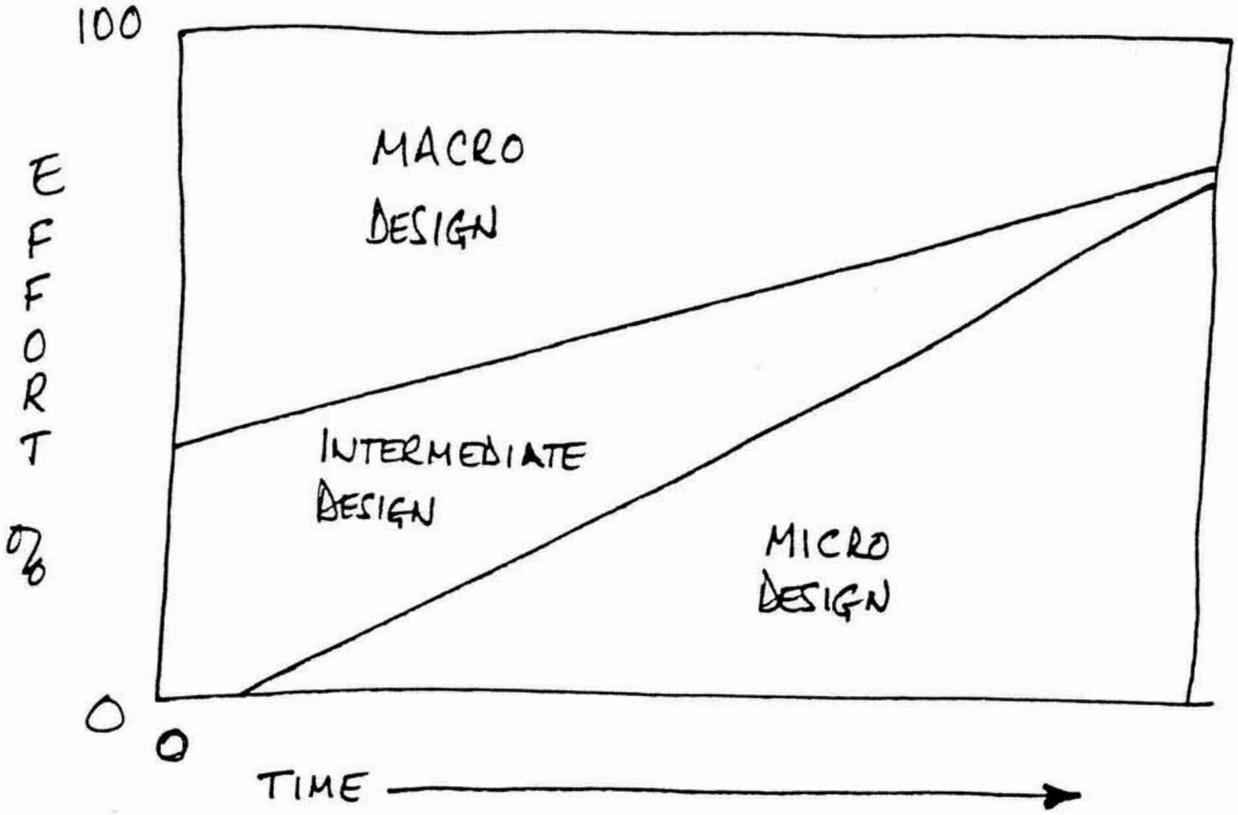
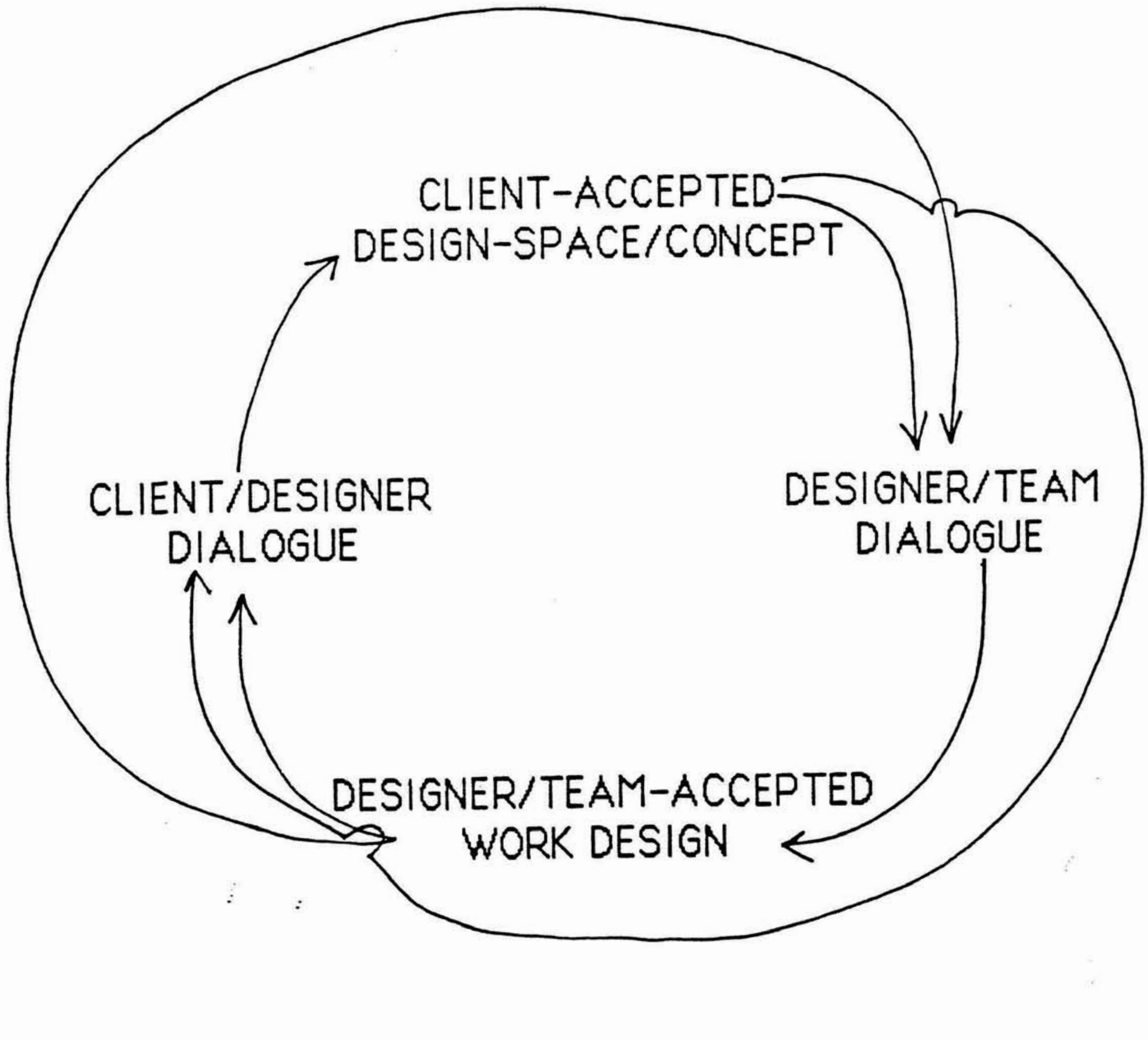


FIGURE 5

A CROSS-CUT THROUGH the DESIGN PROCESS


CLIENT-ACCEPTED
DESIGN-SPACE/CONCEPT
DESIGNER/TEAM
DIALOGUE
DESIGNER/TEAM-ACCEPTED
WORK DESIGN
CLIENT/DESIGNER
DIALOGUE