A Decision-Oriented Approach to System Design and Development

Morton L. Metersky

Abstract—**Design and development of major systems has become an institutionalized, regulated process. Systems, in one form or another, are built to provide input to a decision maker. The decision maker, a major system component, is often neglected in current system design practice. As a result, the systems do not meet pivotal requirements of typical users, and full performance potential is rarely attained when they are deployed into the operational environment. This paper describes a two-phase, decision-oriented process that overcomes many of the system usability problems and unfavorable cost/effectiveness consequences resulting from current design and development processes. The system design and development approach presented combines the use of a unique version of prototyping, a decision-oriented prototype, detailed requirements analysis, strong user involvement and extensive user operational testing throughout design and development. The first phase ends with a user-tested and user-endorsed prototype. This prototype becomes the source of a preliminary system specification that launches the second phase, which includes formal specification development, hardware and software implementation, and predeployment formal usability testing. The intent of this approach is to provide a seamless match between user-centered design and overall system design.**

I. Introduction

MANY SYSTEMS can be defined as being information systems since they, in one form or another, process and provide output (data or information) for a human to use in making a decision. An information system is designed to meet requirements imposed by the organizational and operational environments in which it would be employed. These environments, especially those of a military nature, normally present an overwhelming, diverse and extensive data input that the information system must accommodate. Since a human is an integral part of an information system, what advantage can be made of this plethora of input if it is not constrained or the human computer interface (HCI) not properly designed to conform to human cognitive limitations? One obvious answer is an inability to successfully cope with the input, resulting in unacceptable human and therefore system performance [1]. To alleviate this outcome, humans must depend upon computers for more and different functions than they have in the past. Andriole [2], states that users "would demand analytical solutions to complex problems; they would expect computers to execute data-intensive functions on their own. They would assume that analytical algorithms would help to plan, assess, decide, monitor and allocate, not merely calculate, sort and display." These increased functional requirements place a heavy emphasis on designing a system that reduces both physical and cognitive workloads but still allows the user to maintain a satisficing level of performance.

Designing a system that users will utilize requires that it be attuned to their wants and needs. Conventional development approaches do not communicate to the users, except through documentation, exactly how the system will meet their requirements. An alternative to the conventional approach is to use a prototype as a tool to help the user and developer establish requirements together. The decision-oriented approach to design and development described herein includes the user as a participant throughout the design and development of the system. Moreover, it emphasizes, during the requirements definition phase, the decisions that the user makes. Moreover, this paper presents a two phased approach that integrates the human as a decision maker with software and hardware components to produce a decision-oriented requirements justified system design. It expands on the ideas presented in a paper by Metersky, Ryder and Leonardo [3], by explicitly addressing contextual decision making. The approach postulated and issues raised with regard to current practice are based on experience gained in applying the decision-oriented system design approach to several design efforts; one of which is described below.

The remainder of this paper is organized as follows. An example of a fielded prototype in which the decision-oriented approach was employed is presented first, followed by a discussion of the role the users and their operating environment should play in system design. The second section discusses current prototyping approaches and their limitations in producing a system that incorporates the human as an integral component. The next section describes the decision-oriented system design approach proposed as an alternate to current approaches. The paper concludes by addressing why the author considers the proposed approach to be a better way to design systems.

A. A Fielded Prototype

In February, 1989, a multi-disciplinary system design and development team, comprised of both military and civilian personnel (including the author), visited a fleet unit to obtain requirements for the design and development of a decision support system (DSS, and in the context of this paper considered mission critical software). Team personnel had previously developed instruments for use in gathering data by both questionnaire and interview. Since they had the support of

Manuscript received June 8, 1992; revised January 8, 1993.
The author is with the Naval Air Warfare Center, Aircraft Division, Code 1033, Box 5152, Warminster, PA 18974.
IEEE Log Number 9209639.

the officer-in-charge (OIC), the military personnel interviewed were very cooperative and provided substantial input. There are designers who feel that users do not know what they need. That goes against the author's many years of experience in obtaining input from the user community, specifying what their requirements are, for both current and future systems.

The development team stayed for a period of time sufficient to both collect and analyze the data and allow the OIC and his staff to review the findings. This initial set of DSS requirements was later translated into functional requirements and published for review by several related fleet units in order to obtain inputs from a broad spectrum of operational environments. The revised functional requirements became the foundation upon which the DSS system was designed. However, it was realized that the functional requirements obtained from the user community were only a starting point. What had been provided was based on perceived needs as viewed from the perspective of their interaction with the operational environment. Additional requirements that defined how the user best interacts with the DSS within an organizational context was lacking. This data was obtained by conducting an organizational analysis that concentrated on studying how and with whom the DSS users interact and communicate, both inter and intraorganizationally.

In a decision-oriented system design, it is necessary to delve deeply into the relationship among the system architecture, HCI, information format and type of presentation. A relatively low cost and quick reaction approach was used to determine these requirements, i.e., a strawman DSS storyboard [4]. The storyboard was developed on a PC compatible with the computer system available at the operational sites. The first build, which had 40,000 lines of code, was completed in six months. The DSS storyboard was demonstrated at the sites and inputs obtained guided the development team in the decision-oriented aspects of the system design.

After completing the first build of the DSS, it was delivered to one of the operational units that would act as a Beta test site. A three man team installed the DSS and trained the operational personnel in its use. A User's Document was written, but because of the simplicity of the DSS design, it was not extensively referred to. The OIC started to use the system before receiving any training. During the on-site period and the ensuing weeks, modifications and additions to the existing functional capability were suggested by both operational personnel and the installation team members. There were a total of 48 changes recommended. During the following of months, these changes were incorporated into the DSS and issued as Build 1.1. Another installation visit was made to the Beta test site. The new Build, as did its predecessor, performed well and was favorably received.

B. Discussion

It should be apparent that for a system to reach its full performance potential, particular consideration must be given to the demands and expectations of its user community. Even though a system is technologically state-of-the-art, it must be responsive to the user's needs and easy to operate or it may not be utilized. To avoid this type of occurrence, it is important that the developer understand both the user communities' requirements as well as the system's context, i.e., the physical, organizational, operational and decision making environments. The best way to assure complete understanding of the system's context is to include the user as an integral part of the the system development team. In a paper on user acceptance by Rouse and Morris [5], they state that, "Obviously, a key element of this approach, *on how to improve user acceptance*, (author italics) is a high level of user involvement in the front-end analysis, automation decisions, and implementation process." This means that the user should be involved throughout the entire design and development effort, not just as a source of initial requirements. In addition, the user can also provide the necessary input to help define and provide insight related to the subtle characteristics of the context in which the system will be utilized and decision making performed.

There is an extensive body of literature on system design published by DoD agencies, non profit government funded organizations, commercial companies, and in technical journals. In spite of this, and the accepted and widely practiced systematic approach to system design, there are still, unfortunately, many examples of inferior products. They often have huge cost and schedule overruns, unacceptable performance and inadequate operability. Many of these products fall into the realm of information systems.

One category of information system that has been receiving great attention in recent years is decision support systems, both commercial and military. Military information systems differ from commercial ones in several respects because of the cost of the decision. In a military setting, decisions made using a DSS can have life and death consequences. Satisficing decisions may be acceptable, but erroneous or subsatisficing ones are not. Design factors that differentiate a commercial from a military environment have a large impact on system design. Factors, though not unique to the military environment, that differentiate the two environments by intensity and rigidity and that affect design include:

- Level of stress
- Decision time pressure
- Decision context
- Fatigue and boredom
- Work environment
- Organizational environment
- Nonwork environment, and
- Cost of the decision.

Designing a DSS for use in a military environment requires that special consideration be given to the decision context. The context in which the operator's decision making is taking place, which in a military environment is very critical to obtaining acceptable performance, does not receive the attention that its contribution to system effectiveness dictates. Screen design to optimize the communication interface between the machine and the man is not the only contextual element of concern. This subject has been extensively discussed in the literature [6]–[8]. It is the architecture of the decision making process that appears to be inadequately integrated into current

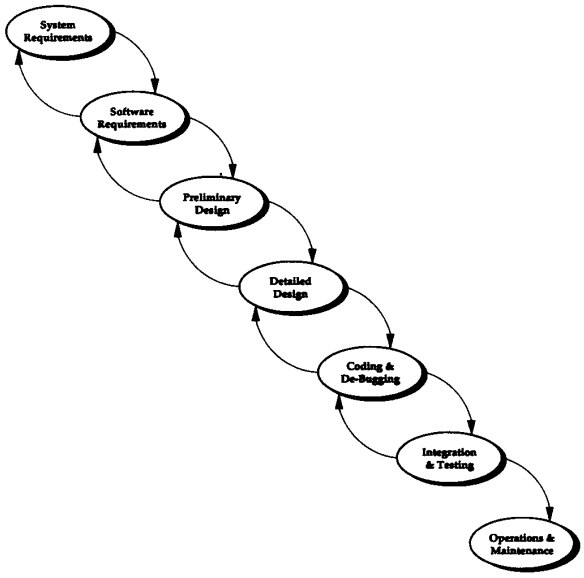

Fig. 1. The "Waterfall" Life Cycle.

system design practice. An architecture that accounts for both the decision task structure and the operator in the environment in which the decision occurs has the potential for enhancing the operator's decision making performance. Michael A. Fineberg [9] states, " What is still lacking is a deeper appreciation of the interaction between the internal capacity of the human operator and the external demands placed on him by the task at hand and the stakes of the game." Both the task and the "stakes of the game" have contextual implications that must be considered during the design process.

II. System Design Approaches

Life cycle models of software system design were developed to support the development of large, complex software programs. Current system design approaches are dominated by variants of the Boehm "waterfall model" [10], Fig. 1, originally introduced by Royce [11]. It is characterized by a logical progression of phases with validation and feedback inherent in each phase. This model is the one most widely used and forms the basis for DoD-STD-2167A and other standards. Another model, an outgrowth of the waterfall model, also developed by Boehm *et al.* [12], is the "spiral model," shown in Fig. 2. However, these approaches fall short in providing the user with a comprehensive system design. Each of these system design models lack at least one of the following considerations:

- Establishing requirements based on the decision making needs of the user. The difference between user and client (the organization paying for the effort) must be understood
- Studying the organization, the user's working environment, to understand its impact on requirements, especially as it may impact decision making
- Initiating an iterative prototyping approach to ensure all requirements have been included and the design, especially the human computer interface (HCI), is acceptable to the user
- Including the user in all aspects of the developmentprocess, including evaluation of each prototype increment and laboratory HCI experiments;
- Writing minimal documentation until after the tested prototype has been completed; Developing software and hardware requirements specifications based on the user tested system prototype;
- Implementing the system design employing as much prototype software as possible
- Testing the system against both the software requirements specification and the system prototype.

The full needs of the user are often not fully understood, even though a requirements analysis is performed, until after the system has been fielded and software change requests are initiated. Part of the problem is 1) the need for continuing the requirements analysis throughout the design phase is not considered, 2) current system design life cycle models frequently do not include the decision maker (the user) as an element of the system, nor 3) do they give serious thought to the impact of the decision process on user requirements. The models specify system requirements studies but often the results do not include full consideration of the end user's needs. Some of the models, by their design, create the tendency to freeze requirements too early in the design process.

Many components make up a DSS, but what is most important to the user is *how* the information is presented, rather than algorithms, software architecture, programming language, etc. These are all transparent. The user's understanding of how functional the system will be is reflected in the HCI. The designers often develop interface displays without consulting either the user or human factors specialists. This occurs because the system design team often views itself as a fount of knowledge and ignores the knowledge and experience of the user community. There is often an evident mismatch between the users' and designers' perception of the operating problem, or a failure on the part of the designer to recognize certain constraints faced by users in the operating environment that affect the system design's operational utility. It is not appropriate to wait until the testing phase of system development. Laboratory testing, whose prime objective is software verification, does not provide a complete understanding of all factors that influence valid HCI design. To adequately determine if the interface will be easily understood and properly formatted, it is necessary that the system be exercised by the users in their operational environment. However, HCI design is not the only concern. There are other interfaces that must be taken into account. How to best transform algorithmic output to information that is effectively applied by the decision maker requires more than just good HCI. The organizational environment, a major element of the decision process, should also be considered an integral part of the requirements process. Interfaces, both personal and organizational, communication procedures, organizational structure, level of stress and physical surroundings all contribute to effective decision making and therefore impact system design.

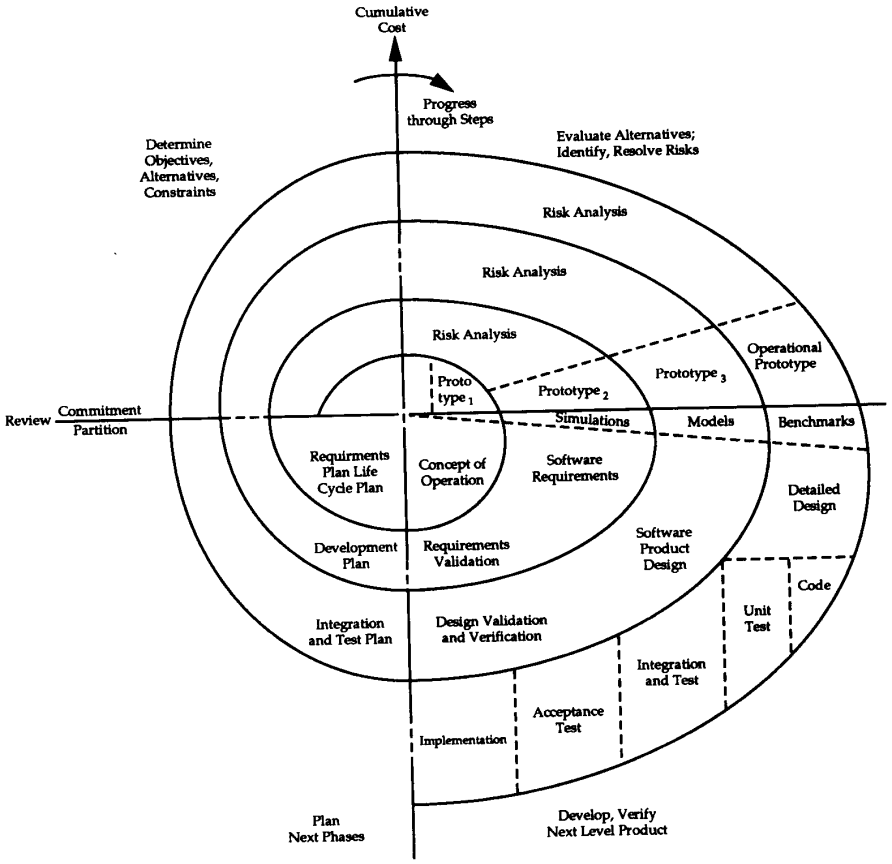

Fig. 2. Spiral model of the software process.

Incomplete requirements can have a profound impact on system cost as well as effectiveness. In determining where the major source of a system's problems originate, Boar [13] states that 60-80% come from inaccurate specification of requirements. This can be virtually eliminated if the user is the source of the requirements. Gomaa *et al.* [14] found that

> "The changes made to the Requirements Specification, as a result of user feedback obtained through using the prototype, were of relatively low cost and would have been considerably more expensive to make had they been left to later in the software life cycle."

Another problem that often results in higher costs and confounds system design, especially in the requirements development phase, is the lack of understanding by the clients that they are not the users. This is especially true in DoD. The clients are often military officers who have had previous operational experience and feel highly qualified to impose their feelings vis-a-vis system design issues. What should be realized is that the system is being designed to service the user *community*, not any one individual. The system design must satisfy the user not the client or the design team.

As described by Davis *et al.* [15], evolutionary prototyping provides incremental builds for the primary purpose of ensuring that a complete set of requirements is established. Using this approach even when a detailed requirements analysis has been performed provides other benefits. The user has the opportunity to evaluate each build in the operational environment from several perspectives: 1) exercising the software to detect errors, 2) commenting on the HCI, 3) evaluating how the build "fits" within the organizational environment, and 4), one often not realized, helping in the transition process. In addition, at the completion of the prototyping phase, a system exists that can be used operationally, and forms the basis for formal system specification.

The knowledge that the users possess of the operational environment, tasks to be accomplished and decisions for which they are responsible can be applied to areas of the design process other than requirements specification. These areas are: 1) system architecture, 2) build evaluation in terms of requirements, 3) usability, 4) code error determination, 5) HCI evaluation and 6) HCI experiment participation.

One of the reasons that development costs spiral is the necessity for modifications to the system discovered late in the development cycle. A major part of these costs result from having to change all the documentation. If, during the prototyping, only minimal documentation is performed, e.g., user's manuals, software architecture design and algorithms, large

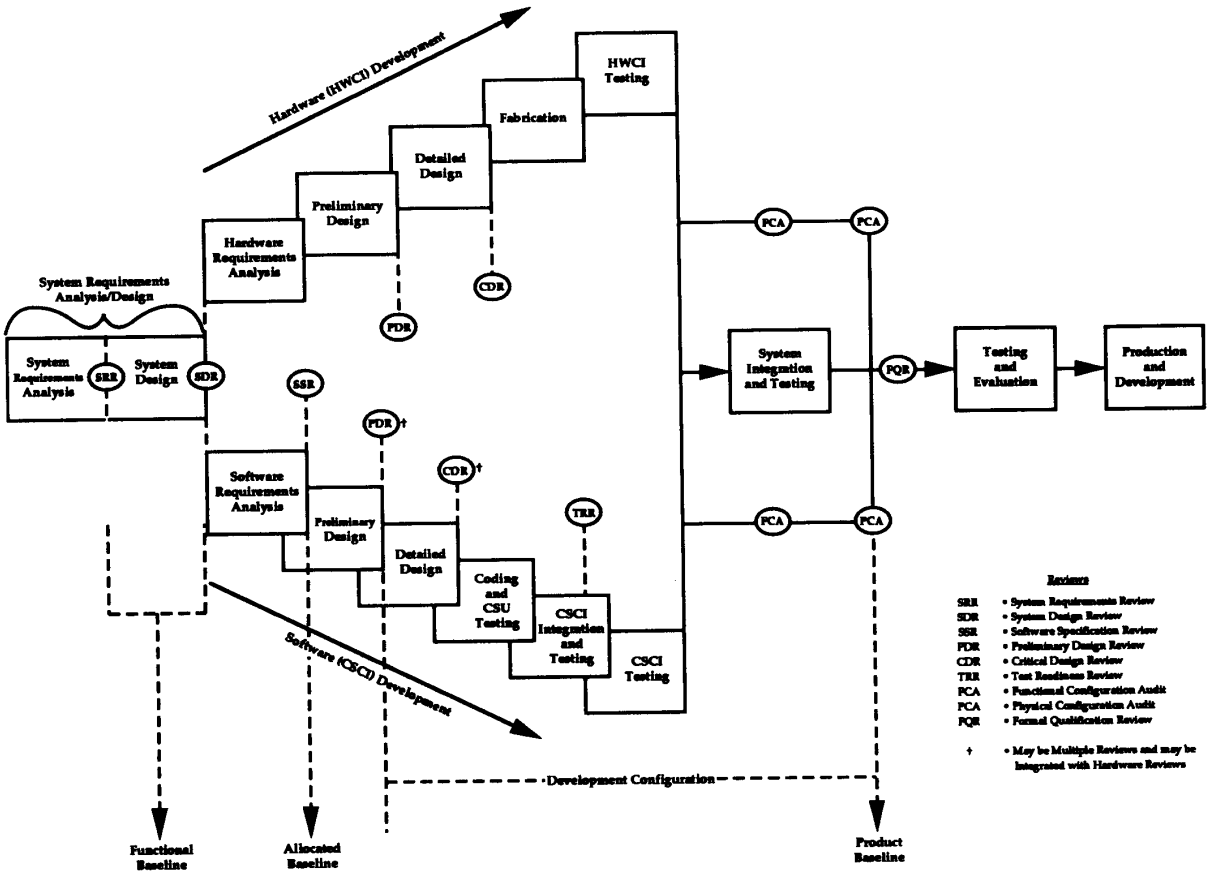

Fig. 3. The defense department's standard life cycle, DOD-STD-2167A.

savings will accrue. Only after the prototyping is complete should formal documentation be performed.

Conventional life cycle models portray hardware and software developments performed in parallel, as shown in Fig. 3 of the Defense Department's Standard Life Cycle. Software requirements should drive hardware requirements. One reason for this is that unique characteristics of the software may not be determined until late in the development. For example, timing studies may indicate that in order to be operationally useful in a real time environment requires the use of parallel processing capability. This may not have been considered. If the software requirements are not used to specify hardware requirements, the result could be incompatibility. If the hardware as well as the software specifications were to be based on the user tested prototype, this would not occur.

When the prototype system is implemented (i.e., it undergoes formal documentation and coding), the code will be optimized to minimize run time and storage requirements. It is important to consider the prototype system code. Both schedule time and cost savings may result by using large portions of the prototype system code in implementing the final system design.

Normally, the software system is tested against the software requirements specification (SRS) to ensure compliance. These tests only determine that the system is functionally complete as defined in the specification. It does not test whether the system provides the users with the complete capability they need, i.e., whether the SRS represents the full spectrum of user requirements. By testing against the user tested prototype system, this objective can be accomplished. Using the prototype as a basis for testing the SRS provides another way of reducing potential cost increases that can result by having to make changes late in the development cycle.

III. Decision Oriented System Design

The system design and development approach proposed by the author is illustrated in Fig. 4. It is divided into two major phases, each with a different objective. The Prototyping Development Phase's objective is to define, design and build a preliminary system; the Implementation Phase's objective is to provide a formal set of software and detailed documentation in preparation for operational test and evaluation, a precursor to formal system acceptance. It is the addition of a multi-attribute prototyping phase and the impact that it has on the implementation phase that differentiates this approach from those that are employed currently.

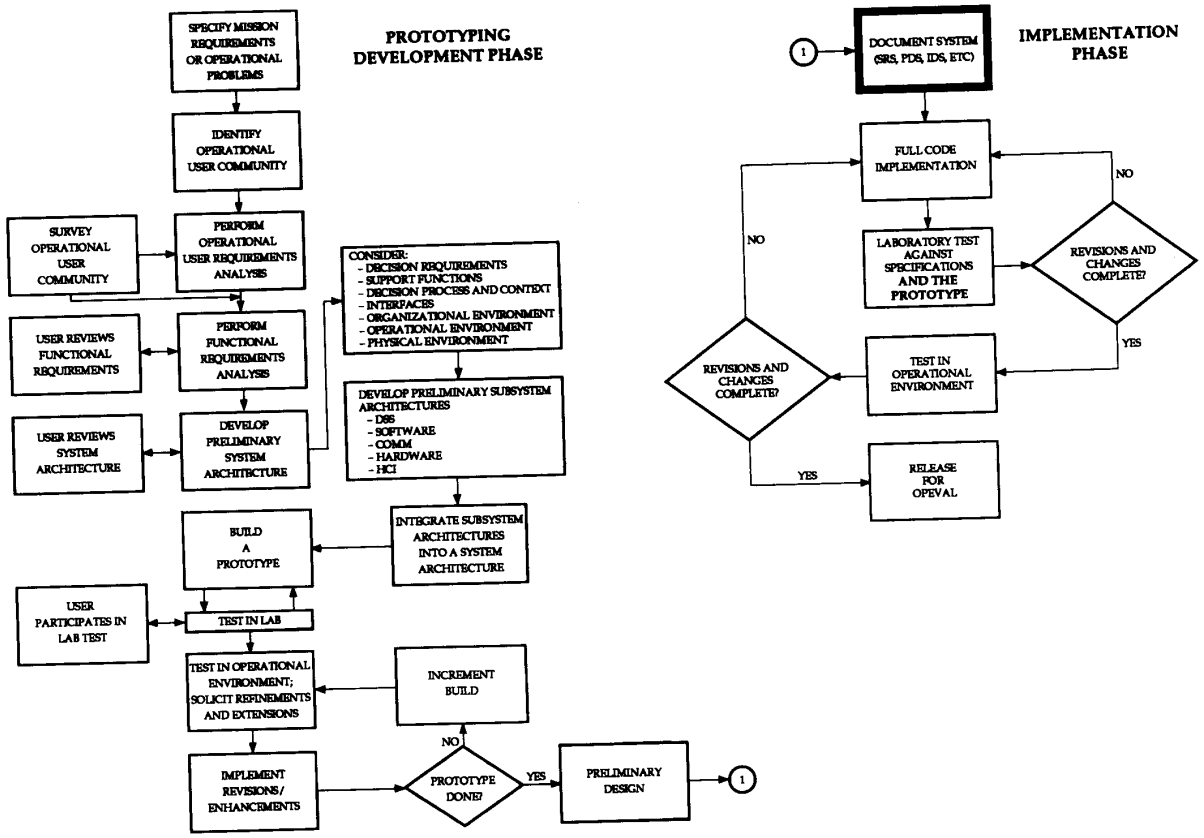

Fig. 4. A system design approach.

A. Prototyping Phase

The prototyping phase is distinguished by several characteristics:

- Heavy emphasis on defining what the system should be capable of doing, i.e., requirements analysis
- Involving the user community in the system design and development effort
- Determining the user's decision making environment and integrating this information into the system design
- Issuing incremental builds to the user community for evaluation during development
- Completing the prototyping development phase with a functionally complete and operationally tested preliminary system.

Although at the completion of the prototyping phase the system is ready to use, it is not ready to be officially accepted. The code may be highly annotated, and therefore not necessarily efficient, and the documentation not written in compliance with accepted standards. Formal preparation for deployment is accomplished during the implementation phase. This includes code optimization, complete system documentation and formal evaluation. The implementation phase is conducted in a manner similar to the process described in Military Standard 2167A [16]. Fig. 5 shows a modified 2167A process that more logically integrates with the prototyping phase. A major change has the preliminary design impacting both the software and hardware designs. Usual 2167A developments have the software and hardware designed in parallel. Moreover, in many system developments the hardware is specified prior to the initiation of software development. Both these approaches that impose constraints on proper system design that are inherently fraught with the potential of cost and time overruns. These overruns occur because the realization of a possible mismatch between hardware and software may not be discovered until late in the development cycle. Andriole [2] comments on 2167A:

> "Return for a moment to the 2167A life cycle. Note that *hardware* requirements analysis, design, and development proceed *simultaneously* with *software* requirements analysis,design and development. *This small assumption about optimal systems design and development represents the institutionalization of a major constraint we face all the time: the prespecification of hardware.* In practice we are usually told what the hardware configuration will look like before we even talk to the first user. We are seldom able to negotiate this constraint; it is almost always immovable. It is also ludicrous."

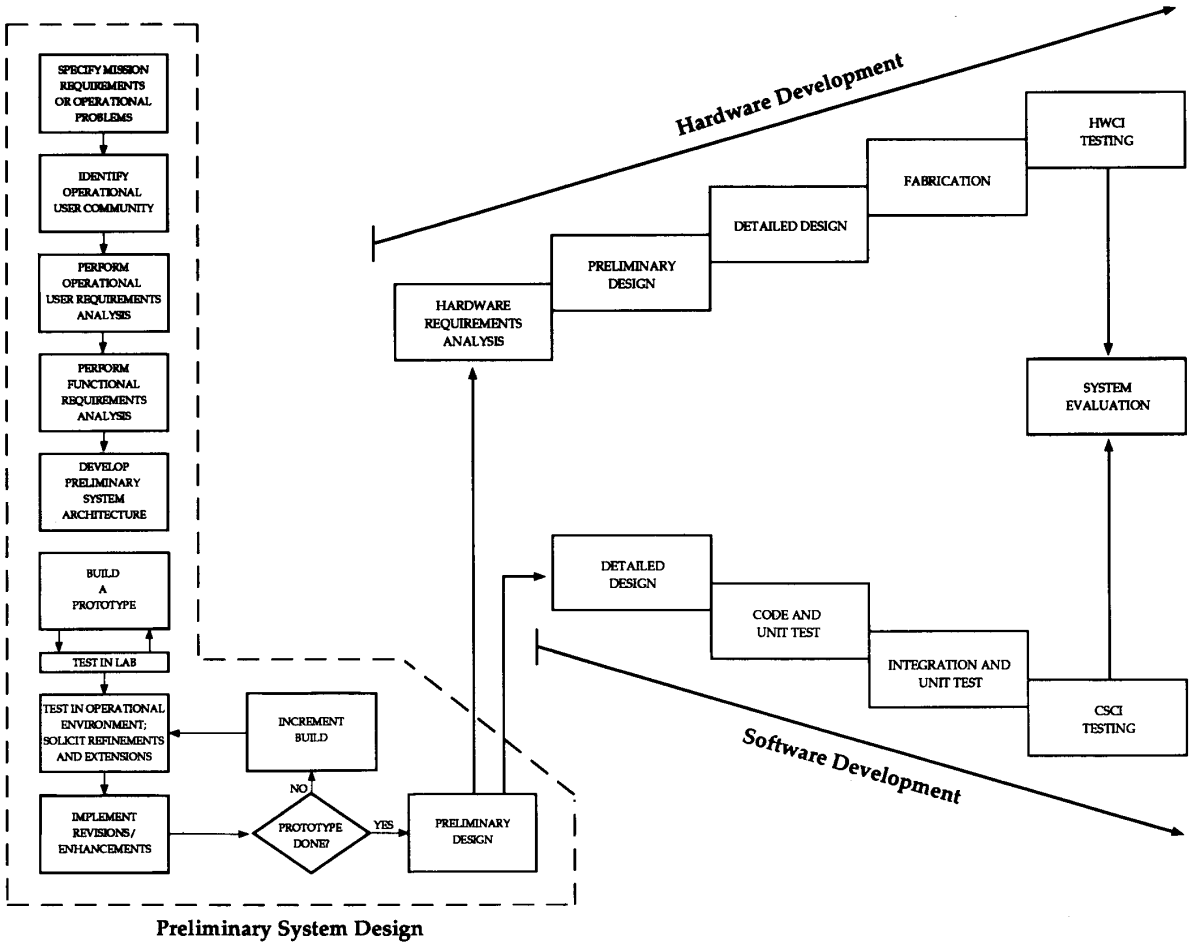

Fig. 5. Modified 2167A development.

The proposed approach produces a completed system, although only a prototype, on which to base a hardware architecture, thereby greatly reducing development uncertainty.

The implementation phase is completed relatively quickly and cheaply because the system exists and can be referred to for both documentation and coding. In addition, testing is performed against an actual system, not just a written specification. This, as well as other aspects of the proposed design and development approach, can overcome costly and time consuming changes that have become an inherent part of most developments. One major reason for the cost savings accrues as a result of the detailed requirements analysis, whose output forms the basis for the system design. Fig. 6 [17] shows the relative cost to correct an error as a function of where in the design and development cycle it is discovered. It very clearly indicates the cost benefits of a thorough requirements analysis.

The prototyping development phase, shown in Fig. 4, has three major parts, i.e., requirements analysis, system architecture design and cyclic development and evaluation.

B. Requirements Analysis

The first, and most important part of the prototyping phase, is the requirements analysis. Requirements analysis usually starts with a detailed statement of the requirements the system must satisfy. However, all the information needed to define a decision-oriented system design is not always available in official requirement's documents. Frequently, the users input to system requirements is not solicited. Andriole [2] states, "Systems analysts see users as the *indirect source* of software requirements. Nearly all texts on requirements start with software requirements; user requirements are treated as "givens." ... It is much easier for systems analysts to assume user requirements from a limited data base (and then begin to specify a computer program), than to work closely with users over some period of time."

Rather than this inadequate approach, a multidisciplinary systems analysis team, including at least one behavioral scientist, should obtain input on requirements from the user community by conducting comprehensive surveys and interviews. Emphasis is, of course, placed on obtaining inputs from

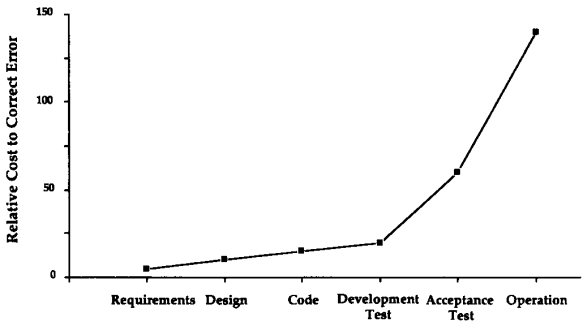

Fig. 6. Cost of a software error.

the user community directly involved in the system being developed. However, it is important that related system(s) and organization(s) be considered since the system being developed either services or interfaces with them. The data collected is analyzed from several perspectives and a high level functional requirements document is prepared and returned to the user for comment. After the requirements are finalized, a high level functional architecture is developed. This document forms the basis for the first build of the prototype.

Decision support systems, like other types of information systems, operate within the structure of an organization. An organization can be generically defined as a systematic arrangement of physical entities, both human and hardware, whose aim is to accomplish some objective or related set of objectives. It is important to realize that organizations exist within the boundary of a larger more encompassing environment. To fully understand an organization, both its environment and boundary must be completely defined and their effect on organizational effectiveness understood. Organizational effectiveness in many systems becomes the major factor impacting system performance, and consequently, operational performance.

To determine the influence of organizational factors on operational performance, analyses not normally a part of the system design process are required. These analyses use techniques from the psychological sciences and concentrate on factors that relate to the decision process. Fig. 7 indicates the task-source relationship for analyses that are conducted for a decision-oriented system design. This figure is intended to indicate the impact that consideration of the organization should have on system design, especially when viewed from a decision perspective. Integral to these tasks are identification and/or evaluation, at a minimum, of the following.

- Organizational decision processes and their impact on operational performance
- Decision-making responsibility assignments to components of the system
- Relationships among organizations and groups within an organization
- Informal and formal interpersonal and inter/intragroup communication links
- Need and/or availability of decision support
- Compatibility of the organization's structure with the environment in which it must operate
- Organizational and procedural problems that negatively impact the decision process
- Obstacles that impede timely, accurate and complete information/data exchange between and among people, groups and organizations.

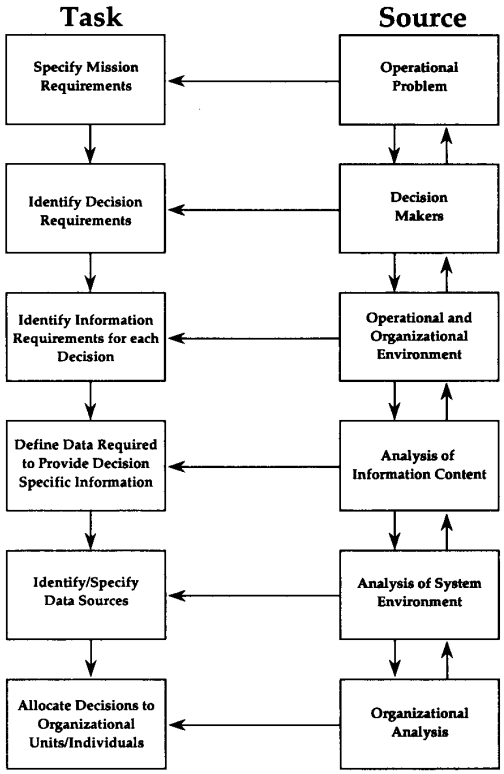

Fig. 7. Decision task analysis.

The organizational analysis that addresses these and other considerations is decision oriented, i.e., emphasis is placed on those factors that influence the design of a decision support system.

By far, the most important aspect of any system development is specifying a complete set of functional requirements. Requirements are the foundation upon which a system is built. The proposed approach includes an extensive requirements analysis in which the user community plays a central and active role. Functional requirements are usually considered to be sufficient in terms of defining a system. However, they are only one aspect of system requirements. It is also important to the user how functions are implemented, integrated and interfaced. This aspect of system requirements is addressed in two ways. First, data obtained from the user during the requirements phase is analyzed in order to determine how it may impact design of a decision-oriented software architecture. Second, a strategy of incremental builds are employed that play an important role in the development effort. They are issued to the user for evaluation and the input received is utilized to modify the system design on a build by build basis.

A strawman, or baseline, "system" depicting a set of displays and an architectural framework is established from an initial set of requirements conceived by the development team. An important requisite of this approach is, of course, that personnel on the team be familiar with the user's operational environment. Through use of a portable computer, it is possible to generate, at the user's installation, variations to the baseline interactively with the user. These displays and architecture are implemented in a "storyboard." A "storyboard" is a computerized representation of the display sequence that can be shown to operate within a specific architectural context. It facilitates efficient and effective exchange between the system designer and the user. Andriole [4] suggests that, "great cost-effectiveness can be gained from developing working models of interactive systems before any programming commitments are made." The strawman storyboard serves several purposes. In addition to aiding in requirements validation and system sizing, it: 1) acts as a catalyst in soliciting requirements, 2) allows the design team to elicit responses relative to display formats and information content, 3) helps to determine the functional flow of the system and 4) simulates the decision process' contextual architecture. Additional data concerning 1)–4) is obtained from the data collection techniques previously mentioned.

C. System Architecture

The system architecture design is predicated on the need to present information the operator requires to make decisions and provide the necessary decision context environment. The elements involved in performing a decision-oriented design are shown in Fig. 8. As in any design effort, it begins by defining what has to be accomplished, in this case the decision set. The decision set is a collection of functions, previously defined, that comprise a major decision task, e.g., threat assessment, cued search, uncued search, etc. The command and organizational structures, the operational environment, and the immediate decision environment are also an integral part of the context that determines how well a decision is made. This context is also defined so that issues related to its impact on system design can be considered. There are several attributes a decision support system demonstrates that relate to the context in which it is used. These are extracted from Adelman [18].

- Compatible with the user in terms of technical background and training
- Consistent with the operational environment
- Aids not only in decision making but also in efficient task accomplishment, and
- Compatible with organizational procedures.

These attributes, among others, are important when considering the impact of the decision context on DSS design.

To reduce the cognitive load of the user, the sequence of functions that make up a decision set must be defined to make reaching a decision almost transparent to the user. This process is a major component of the decision context. Decision-oriented information presentation, properly formatted, reduces the cognitive processing load of the operator by providing only information pertaining to the decision being addressed, thus allowing him/her to concentrate on the problem under consideration. Providing the operator with a well designed decision environment also improves operator efficiency, as well as performance, by assisting with the determination of how to best accomplish the decision making task. Different subsets of decision functions are associated with decision tasks that the operator is required to perform, e.g., situation assessment, planning, etc. The operator does not have to think about which functions are associated with which task. The system is designed to provide the operator with the decision process flow for each task. Predefining the decision process flow helps to overcome problems caused by the decision context environment.

The system architecture includes, in addition to the DSS, all operational and application software not directly related to the DSS, communication and digital interfaces with other systems that supply data and/or information, and all system hardware. The definition and design of these subsystems is based on the need to support the decision making function of the system. For example, data requirements derive from the requirement to produce information for decision making. The question that drives data requirements is, " what data is needed to produce the aforementioned information?" Organizational interfaces are established to either allow data and/or information to flow into (and out of) the system or provide a means to implement a decision. How best to establish these interfaces impact communication network and hardware requirements and computer net specification.

When completed, the system architecture design is reviewed with the user community to verify that it not only meets their requirements but is also compatible with the physical, organizational, operational and decision environments in which it must function. This includes ensuring that user inputs are obtained that relate to the decision environment, since it is not a standard consideration.

D. Human Processor Considerations

Decision support is provided to individuals primarily to reduce the amount of input that they are required to process. The product of a decision support system development is an information system that essentially absorbs large amounts of data and outputs limited decision specific information. In essence, decision support systems perform specialized data fusion. In a military environment that is characterized by high stress and short time pressures, the individual, if faced with excessive input, finds a means to reduce cognitive load. This is usually accomplished by utilizing cognitive biases. Sage [19] lists 27 different forms of biases. In general, when confronted with excess data, humans resort to heuristics that simplify the problem and reduce the amount of input that must be processed, with the result sometimes being an erroneous and ineffective decision [20] . To reduce the effect of cognitive biases, consideration is given to the information that is presented to the decision makers and how it is presented and how they are allowed to interact with the system. Thoughtful attention is given to reducing cognitive load and minimizing the impact of cognitive biases that are operant even when

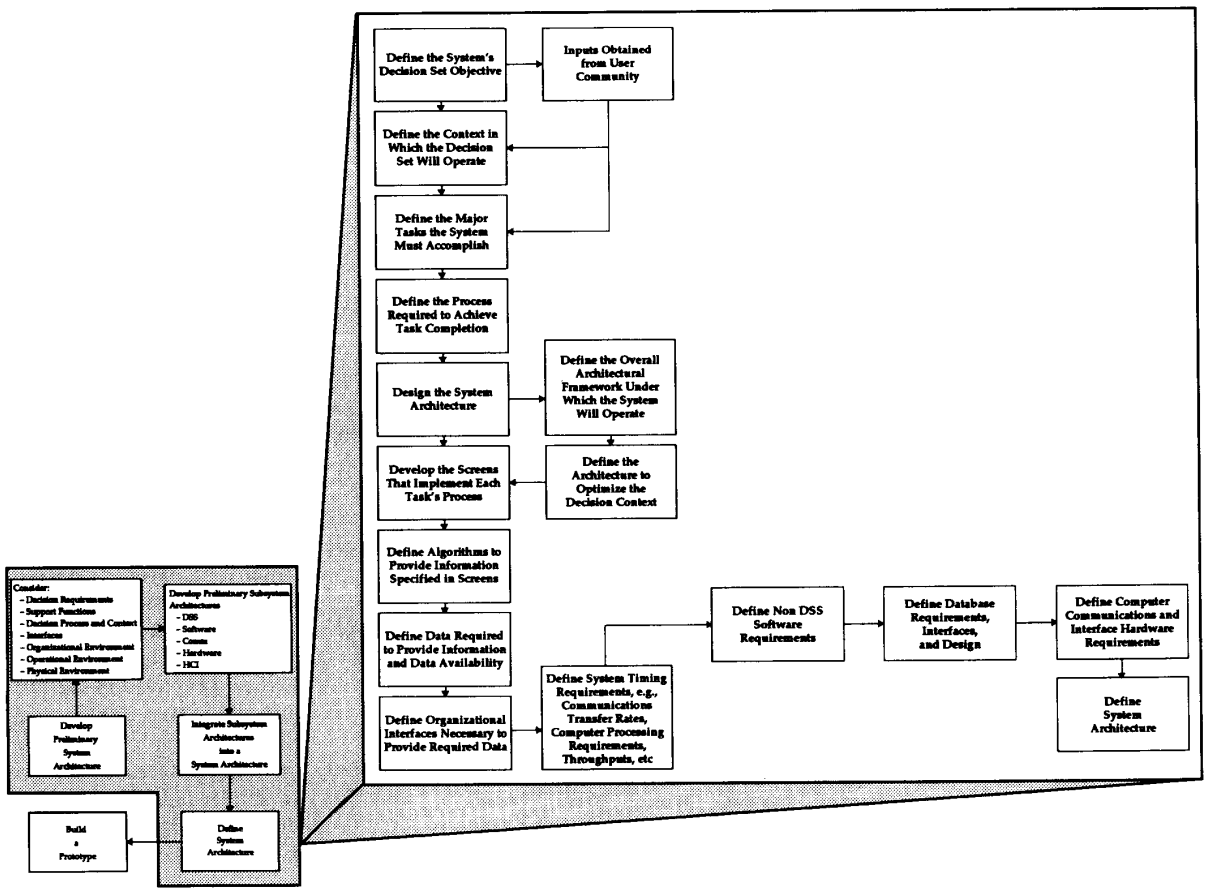

Fig. 8. Decision emphasis design.

cognitive overload does not occur. Fig. 9 shows a partial list of the factors that effect cognitive biases. Many are a function of the individual decision makers and the environments in which they operate. Alleviating cognitive bias effects that result from these factors is another reason that the decision context is an important concern in establishing the system architecture.

It is important that the designer investigate the implications of the system's functions with regard to decision making. The question is asked, "What decisions must the operator make to accomplish these functional tasks, how are they implemented to ensure compatibility with the limitations of the human processor and what is the impact on system architecture and design." Since a major influence on human decision making is cognitive biases, consideration of debiasing techniques [20] during system design is absolutely necessary in order to attain good decision performance.

Cognitive biases are only part of the undesirable cognitive/psychological factors that a human has that undermines system effectiveness. Two cognitive factors worth noting is that humans have limited working memory [21] and are not proficient at doing complex numerical calculations unaided. Furthermore, in evaluating alternatives, they have difficulty integrating input from multiple sources [22] and usually do not contemplate more than three hypotheses simultaneously [23], [24]. Of equal importance are psychological factors, e.g., once individuals have formulated hypotheses, i.e., made up their minds, disconfirming input is both ignored and not sought. Additionally, humans are strongly influenced by authority figures, and may ignore disconfirming input because it contradicts the input provided by the authority figure. This is especially true in a military environment. The human, the system's element least considered in system design, presents the most challenging design problem and the single greatest determinant of system architecture.

E. Cyclic Development and Evaluation

A prototype system is developed based on the high level system architecture using an incremental build approach. Each build is designed to include a set of functions that add, at a minimum, new functional capability to ensure that the user wants to employ it operationally. It is very important that the users feel that what is being delivered for test and evaluation is either easier to operate and/or makes them more effective than the present system. The users evaluate each build in the operational environment and provides the development team with input as to its usability, operability, performance and effectiveness in performing its functions.

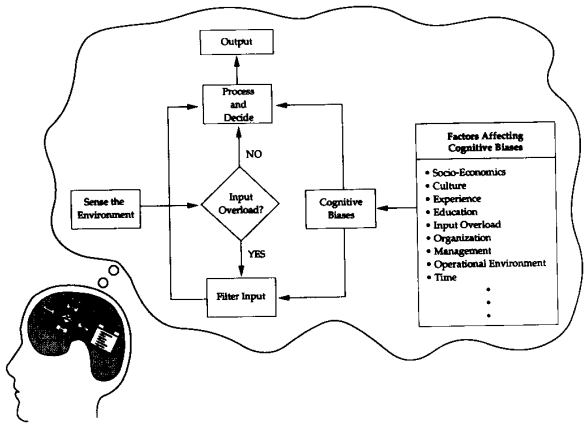

Fig. 9. Human data fusion.

In this proposed approach to system design and development, informal documentation is accomplished after each build has been operationally tested by the user. By waiting until the user is satisfied with the design, there is a high likelihood that the need for system software and documentation changes is greatly reduced. In addition, the continuous build by build testing by the operational user increases code stability, operability and the likelihood of complete functionality.

The major benefits that derive from employing the modified prototyping approach, shown in Fig. 4, is a system that 1) has been tested in the operational environment, 2) fully meets the user's requirements, 3) has a stable set of code, 4) provides a system that is useable operationally prior to software finalization and 5) provides a preliminary design specification that forms the basis for formal documentation (e.g., SRS, SIRD, PDS, IDS, etc.) and code optimization. In this approach, the client has the option of informally introducing the prototype to the user community concurrently with the initiation of formal system engineering efforts.

F. Implementation Phase

The implementation phase starts with documenting the preliminary system design that is the product of the prototype phase. At a minimum, software requirements specification (SRS) and interface design specification (IDS) are written and provided to the contractor for development of the software. Since the software architecture is hierarchically modular, a significant number of the prototype software modules can be used when the system is implemented. During the prototyping phase, in developing the initial build, optimum software techniques would be employed. However, when changes or additions are made to the build based on testing and evaluation, it may not be possible, because of time constraints, to maintain the same level of programming efficiency. In addition, the requirement that software must be easily maintainable may not be a prime consideration during the prototyping phase. However, it is estimated that at least 50% of the code developed during the prototyping phase is useable in the implementation phase. The system software documentation is based on the prototype specification, since the system architecture and design is determined in the course of prototype development.

Since the software system architecture is based on a modular design, laboratory testing is accomplished in a similar manner. Each module is individually tested from four perspectives. One, is it designed to accept the inputs it needs to accomplish its function, two, does the function perform the task it is designed to do and meet a specified requirement, three, does it provide the required outputs and four does it produce realistically meaningful output. Each function is tested hierarchically, i.e., individual modules, or groups of modules making up a subfunction (or lower level function), all the way up to the function and groups of functions. The last group of functions tested is, of course, the entire system.

The final system design, developed based on the SRS, is first tested against both the SRS and the prototype system in a laboratory environment. In current practice, the system is only tested against the SRS making it difficult to determine whether 1) it is operationally useful, 2) it fully meets user requirements and 3) the algorithms produce answers that are operationally realistic or meaningful. After passing laboratory testing, the system is sent to operational sites for Beta testing. After successfully completing Beta testing, the DSS is ready for formal testing and evaluation. It is anticipated that after the exhaustive testing and evaluation performed of the DSS by operational units, it will successfully pass this final test prior to release.

The implementation phase provides the formal software engineering that is necessary to ensure optimized code and the documentation required to understand and upgrade or amend the system. Because this phase employs the final prototype design as the basis for specification and testing, there is less likelihood of redesign, added functionality being required and rewriting of documentation. This all adds up to large savings in design and development costs.

G. System Design Evaluation

An incremental build approach allows the user to test and evaluate each build and to provide real time feedback to the development team. This approach is the basis for providing 1) requirements that weren't obvious during the requirements analysis, 2) discovery of software bugs and 3) a useable prototype. Each build, with its associated user's manual, is hand delivered to the test site for installation and training. Members of the development team spend a period of time with the user community, enabling them to obtain immediate feedback on the build's performance. A form is left with user personnel to record any changes, additions or amendments they desire made to the build. This facilitates build improvements being incorporated into the ongoing build development.

In addition to the on-site testing and evaluation performed, user personnel are asked to participate in laboratory experiments. The primary emphasis of these man-in-the-loop experiments are twofold. The first is to determine the most effective information display formats and content in terms of operator understanding and acceptance. The object of this design effort is to minimize the operator's cognitive processing

load by maximizing information content and decision applicability of what is being presented. The user is primarily interested in obtaining useful information and takes for granted that what drives the screens is correct. The second objective is to assist in the design of the decision context architecture. Ensuring that the software is performing as specified, or verification, is relatively straightforward when compared to evaluating if a DSS improves decision effectiveness, the overall measure of interest.

To test the decision making context to determine how well it meets the demands of these attributes, requires the use of both qualitative and quantitative test procedures that involve analytical or empirical techniques. Sage [25] outlines a methodological framework and provides a set of criteria for evaluating decision support systems. These criteria fall into three categories: 1) algorithmic effectiveness of performance objective achievement, 2) behavioral or human factors and 3) efficacy evaluation. Rouse [26] describes three types of issues that should be addressed, 1) compatibility (with human cognitive capability), 2) understandability (in terms of what is being communicated) and 3) effectiveness (achieving design objectives). These evaluation approaches emphasize both the decision makers and the environments in which they perform decision functions. Although it is preferable to perform DSS evaluation from the perspective of the decision maker in a laboratory setting, it is also necessary to perform evaluations at the Beta test site. Both methods referenced above include in their considerations the impact of the environment in which the DSS is employed. This can best be accomplished in a real world environment.

Each time a build is delivered for user evaluation and testing, it includes all the changes made to previous builds based on laboratory and user testing results. Use of this cyclic test and evaluation procedure guarantees that each build gets reviewed many times. Software coding errors, functional discrepancies and unacceptable human computer interface (HCI) that previously may not have been detected have a high likelihood of eventually being discovered.

IV. Why Is It Better?

Most software systems are developed to provide information for decision making. Yet when the various software development paradigms, both prototyping and life cycle models, are examined, there is no attention given to the decision making process. The conventional model and its descendants reflect an emphasis on the software engineer and the prototyping paradigms are most effective in ensuring that requirements reflect the user's needs. There is a real necessity for a paradigm that expands the breadth of the presently employed paradigms. The importance of the decision process, the user's participation throughout the design and development, design and development time and costs cannot be over estimated. The latter two directly influence whether a system might be funded and the former determines how useful the product will be. Fig. 10 shows the three paradigms. The bottom two were obtained from Agresti [27]. In a paper by Jones [28], he states,

> "The waterfall (or conventional) model relies on the domain knowledge of individuals (doing the design) combined with their *interpretation* (author italics) of the perceived customer needs as a basis for specification of each software component. In a situation where the product is new or the customer is unsure of the requirements, or even both, the waterfall model becomes inappropriate."

The decision-oriented approach to system design and development, because of its strong reliance on user input from requirements through introduction, does not suffer from the aforementioned shortcoming of the conventional model. Obtaining user requirements, although heavily emphasized at the initiation of the design, continues throughout the design phase. There are several other characteristics of the proposed approach that differentiate it from the conventional and other life cycle models.

- Initial build software architecture and human computer interface (HCI) is based on results of the requirements analysis
- The prototype consists of multiple builds, where each provides an operationally useful set of additional functions. Lessons learned, including new requirements, from each prototype iteration tested by the user, are integrated into the next build
- Formal HCI laboratory experiments are conducted as part of the design process. Subjects are obtained from the user community
- The final prototype build provides the input for the implementation phase for writing the software requirements specification, the hardware requirements specification and for testing the system after implementation has been completed and,
- Detailed documentation is not prepared during the prototype phase.

In essence, the prototyping approach described combines the best features of the five prototype classes defined by Law [29]. These were summarized by Jones [28] as follows

- "Exploratory prototypes provide the early focus for discussion during requirements elicitation and verification phases
- Experimental prototypes provide the platform on which to evaluate the proposed software design
- Evolutionary prototypes are concerned with the gradual adaption of operational systems to cater for newly identified or changing requirements, ...
- Performance prototypes are concerned with confirmation that the system will meet its stated requirements ...
- Organizational prototypes are concerned with exercising the system in the end user environment..."

None of these approaches alone is sufficient to ensure a design that is functionally complete in terms of user requirements. Nor do they specifically consider the decision making aspects of the system. If you will, the use of the decision-oriented prototype (DOP), which combines features of all the above, ensures that the system is designed to meet the needs of the

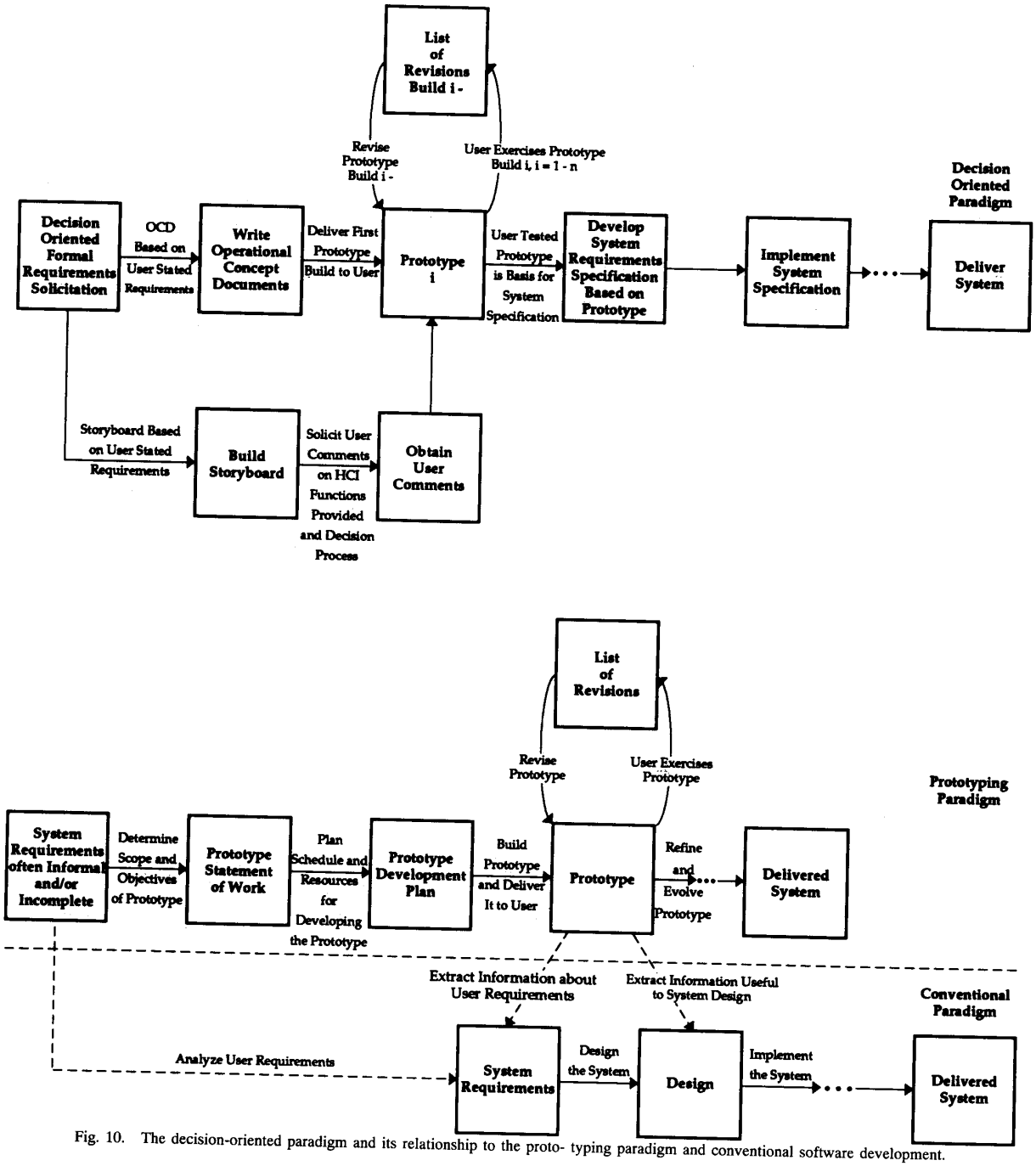

Fig. 10. The decision-oriented paradigm and its relationship to the proto- typing paradigm and conventional software development.

user. The most important result of employing the DOP and the decision-oriented design and development approach is greater assurance of user acceptance.

V. Summary

The use of current system design and development approaches often lead to cost and schedule overruns. Even worse, they have often led to systems being fielded that either didn't meet user functional requirements or were difficult to use, or both. The proposed system design and development approach combines the use of a unique version of prototyping, a decision-oriented prototype, a strong user involvement throughout design and development, a detailed requirements analysis and extensive user testing. A major factor that the proposed approach considers is the decision process and its impact on the decision maker. Any system is built to provide,

in one form or another, input to a decision maker. Yet this design element is neglected in current system design practice. Not only should humans be considered an integral part of a system, but the role they play as decision makers is basic to achieving high performance. The decision-oriented approach to system design and development proposed herein should help to provide useable and effective systems that emphasize the importance of the decision maker as a major element of the system.

ACKNOWLEDGMENT

The author would like to recognize the important contributions made by Dr. Stephen J. Andriole, Drexel University, Philadelphia, PA, Dr. Robert Dick, Systems Research and Design, Inc., Santa Barbara, CA, and the anonymous reviewers for suggestions that improved the clarity and flow of this paper.

REFERENCES

[1] R. Hogarth, *Judgement and Choice*, second ed. New York: Wiley, 1987.
[2] S. J. Andriole, *Information System Design Principles for the 90s: Getting It Right*. Fairfax, VA: AFCEA Int. Press, 1990.
[3] M. L. Metersky, J. Ryder, and M. A. Leonardo, "A change in system design emphasis: From machine to human," in *Advanced Technology for Command and Control Systems Engineering*, S. Andriole, Ed. New York: AFCEA Int. Press, 1991.
[4] S. J. Andriole, *Storyboard Prototyping for Systems Design: A New Approach to Requirements Validation and Systems Sizing*. Wellsley, MA: QED Information Sciences, 1989.
[5] W. B. Rouse and N. M. Morris, "Understanding and enhancing user acceptance of computer technology," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, pp. 965–972, 1986.
[6] W. B. Rouse, "Design of man-computer interfaces for on-line interactive systems," in *Proc. IEEE: Special Issue on Interactive Computer Systems*, vol 63, no. 6, pp. 847–857, 1975.
[7] S. L. Smith, and D. Mosier, *Design Guidelines for the User Interface to Computer-Based Information Systems*. Bedford, MA.: The Mitre Corporation 1984.
[8] B. Scneiderman, *Designing the User Interface* Reading, MA: Addison-Wesley, 1987.
[9] M. A. Fineberg, " Human performance in military command and control," in *Advanced Technology for Command and Control Systems Engineering*, S. J. Andriole, Ed. Fairfax, VA: AFCEA International Press, 1991.
[10] B. W. Boehm, "Software Engineering," *IEEE Trans. Comput.*, vol. C-25, pp. 1226–1241, Dec. 1976.
[11] W. W. Royce, "Managing the development of large systems: Concepts and techniques," in *Proc. WESCON*, Aug. 1970.
[12] B. W. Boehm and P. N. Papaccio, "Understanding and controlling software costs," in *IEEE Trans. Software Eng.*, vol. 14, no. 10, Oct. 1988.
[13] B. H. Boar, *Application Prototyping: A Requirements Strategy for the 80's*. New York: Wiley-Interscience, 1984.
[14] H. Gomaa and D. Scott, "Prototyping as a tool in the specification of user requirements," in *Proc. 5th IEEE Int. Conf. Software Eng.* Mar. 1981, pp. 333–342.
[15] A. M. Davis, E. H. Bersoff, and E. R. Comer, "A strategy for comparing alternate software development life cycle models," in *IEEE Trans. Software Eng.*, vol. 14, Oct. 1988.
[16] U. S. Dept. Defense, *Military Standard: Defense System Software Development*, DOD-STD-2167A, Feb. 29, 1988.
[17] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.
[18] L. Adelman, "Evaluating decision support systems: toward integrating evaluation methods into the development process," in *Advanced Technology for Command and Control Systems Engineering*, S. J. Andriole, Ed. Fairfax, VA: AFCEA International Press, 1991.
[19] A. P. Sage, "Behavior and organizations in the design of information systems and processes for planning and decision support," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, pp. 640–678, 1981.
[20] D. Kahneman, P. Slovic, and A. Tversky, Eds., *Judgement Under Uncertainty: Heuristics and Biases*. New York: Cambridge Univ., 1981.
[21] G. A. Miller, "The magic number seven, plus or minus two: Some limits on our capacity for processing information," *Psych. Rev.*, vol. 63, pp. 81–87, 1956.
[22] R. M. Dawes, "The robust beauty of improper linear models in decision making," *Amer. Psychol.*, vol. 34, pp. 571–582, 1979.
[23] L. B. Lusted, "Clinical decision making," in *Decision Making and Medical Care*, D. Dombal and J. Grevy, Eds. Amsterdam: North Holland, 1976.
[24] J. Rasmussen, "Models of mental strategies in process control," in Human Detection and Diagnosis of System Failures, J. Rasmussen and W. Rouse, Eds. New York: Plenum Press, 1981.
[25] A. P. Sage, "A methodological framework for systemic design and evaluation of computer aids for planning and decision support," *Comput. Elec. Eng.*, vol. 8, no. 2, pp. 77–101, 1984.
[26] W. B. Rouse, Design and evaluation of computer-based decision support systems, *Microcomputer Decision Support Systems*, S. J. Andriole, Ed. Wellesley, MA: QED Information Sciences, 1984.
[27] W. W. Agresti, "What are the new paradigms," in *New Paradigms for Software Development*, W. W. Agresti, Ed. Los Alamitos, CA: IEEE Computer Society, Jan. 1986.
[28] S. P. Jones, "Rapid prototyping: for want of better words," Electron. Res. Lab., Information Technol. Div., Tech. Rep. ERL-0526-TR, Commonwealth of Australia, Sept. 1990.
[29] D. Law, "Prototyping: A state of the art report," Tech. Rep., NCC, 1985.

Morton L. Metersky (SM'86) was born July 30, 1934, in Brooklyn, N. Y. He received the B.S. degree in aeronautical engineering from the Georgia Institute of Technology, Atlanta, GA, the M.S. degree in applied statistics with a minor in psychology from Villanova University, Villanova, PA, and the M. S. degree in management science from Lehigh University, Bethlehem PA. He completed all work for a Ph.D. in business and economics majoring in behavioral science, except for dissertation, from Lehigh University, Bethlehem, PA.

He has been at the Naval Air Warfare Center, Aircraft Division, Warminster, PA since 1960, working in the field of Antisubmarine Warfare (ASW). Since 1967, his efforts have been primarily focused on improving fleet performance through use of decision augmentation. He was responsible for introducing the first decision aid into operational use in 1969, and he conceived and directed the initial operational implementation of computer-assisted ASW search programs. Most recently, he directed the design of a decision support system for the Navy's Maritime Patrol Aircraft. In addition to his ASW work, he has been an active participant in the strategic defense initiative (SDI) program and consults in the area of command and control and decision-oriented system design. He is an active member of the Military Operations Research Society (MORS) and has been a board member twice, serving first as secretary-treasurer and then as vice president, meeting operations. He was the program chair for the 49th MORS Symposium, whose theme was the Art and Science of Military Decision Making. He has been an invited lecturer at the Naval Postgraduate School and the Naval Academy.

Dr. Metersky is an alumni member of AIAA's National Technical Panel on C3I. His publications include papers in applied psychology, ASW, decision augmentation and command and control. While at Lehigh he was elected to Beta Gamma Sigma, national business honor society