Computer information system development methodologies—a comparative analysis

by DANIEL T. LEE
Pan American University
Edinburg, Texas

ABSTRACT

A computer information system is one of the main constructs through which a business firm gains a competitive edge over its competitors. Unfortunately, after four decades of effort by computer scientists and management specialists, ideal methodologies of CIS development are still lacking. The purpose of this paper is to conduct a comparative analysis of CIS development methodologies, trace their historical evolution, and develop integrated methodologies which can be used for logical system design and physical implementation. During the past four decades, CIS development methodologies concentrated on structured methods which are efficient for small systems, but they will soon be overwhelmed by complex and large systems. Only recently, fourth generation languages and automatic design techniques have begun to emerge as driving forces in CIS development. This paper tries to integrate these new technologies into a unified whole for CIS development. It will not only be used for transactional processing, but also for office automation and decision support.
PREFACE

Computer information system development methodologies have evolved through several stages. The information system development life cycle (ISDLC) method is one of the most widely known methodologies. According to Ahituv, the traditional ISDLC has always been a troublesome, complex, costly, and time-consuming process. This inadequacy is primarily caused by the rigidity of its development process and there is no match between its logical design and physical implementation.

In the early days of system analysis and programming, according to Martin, there were few rules other than those of the programming language itself. The methods used in CIS development were often inefficient and caused many problems. The structured techniques represented a search for better methods in system analysis, design, and programming. They did improve the quality of programs and system development but were not building applications fast enough and were bogged down by maintenance problems. This led to the development of new languages, report generators, application generators, database systems, decision support tools, mini-macro computers, operating systems, data communication and networks, multifunctional work stations, automated techniques, expert systems, and integrated systems.

Under automated methodologies computers are used as design workbenches for creating, editing, expanding, and changing structured diagrams. They can also be used to automate data modeling from system specifications, extract subsets of data models for individual applications, check the design being created, and automate the generation of code. Automated designs are based on mathematical axioms so that the overall design can be mathematically verified. This is an extension of structured design; computers can rigorously check the entire design, eliminate all misuse of the constructs, and automate codes which are bug-free and can be used in applications.

Many tools are involved in the automated methodologies. These tools cannot stand alone for effective function. They have to be integrated for combined efforts. The integrated methodologies are therefore the last, but not the least, CIS development techniques with which strong CIS can be built.

This paper first discusses traditional ISDLC and follows with a discussion of various structured methodologies. Automated design methodologies, iterative design methodologies and integrated methodologies are also described.

TRADITIONAL METHODOLOGIES

The ISDLC techniques were introduced to the academic community in the 1950s and 60s. According to Awad, system development revolves around a life cycle that begins with the recognition of users' needs. Following a feasibility study, the key stages are the evaluation of the present system, information gathering, cost/benefit analysis, detailed design, and implementation of the candidate system. Kanter illustrates the application, development, and implementation cycle as being composed of three general phases: analysis, synthesis, and implementation. Analysis is defined as the analysis of company operation and the division of the total operation into logical and workable units for measurement and evaluation. Synthesis, the opposite of analysis, begins to combine and build the parts or elements into a whole. The implementation phase is the "proof of the pudding." The phases of analysis, synthesis, and implementation are never-ending cycles.

Prince divides the CIS analysis and design into five phases: (1) the planning phase, (2) the organization review and administrative study phase, (3) the conceptual system design phase, (4) the equipment selection and program design phase, and (5) the implementation phase. Unfortunately, this method almost stops at phase 3, the conceptual system design. The descriptions of the methodology are quite clear, but physical implementation is lacking.

Murray views the maturation of a system as going through the stages of analysis, design, implementation, and operation. He conceives of the cycle as beginning with a feasibility study that precedes a formal systems analysis. The systems design and implementation are followed by operation of the new (or revised) system. This cycle may repeat. Ahituv summarized that the traditional ISDLC typically contains four major phases such as definition phase, construction phase, implementation phase, and operation phase. Each phase in turn consists of several steps.

Ahituv finished this work in 1982. In 1984 even he admitted that, primarily because of the nature of the systems that must be built with it, this traditional ISDLC has always been a troublesome, complex, costly, and time-consuming process.

Actually there are hundreds of information systems development methodologies which follow this established pattern. The big flaw of this traditional method is that though the phases or steps relating to system analysis and design seem to be clear, there is no practical way to bridge the gaps among analysis, design, and implementation. Most of the traditional design methodologies concentrate on logical system design; it is very difficult, if not impossible, to follow up with physical implementation. Information technology personnel have been trying hard for three decades to find ways out of this inadequacy. They found out that the structured techniques represent an advancement in information development research; it clarifies the many uncertainties embodied in the traditional ISDLC and largely bridges the gaps among system analysis, design, and implementation. The detailed mech-
anism of the structured techniques is the topic in the following section.

STRUCTURED TECHNIQUES

Structured techniques evolved from a coding methodology (structured programming) into techniques consisting of analysis, design, test, project management, and documentation tools. According to Martin,7 structured techniques were intended to be a step toward changing software-building methods from a manual craft to an engineering discipline. In a sense, it is more an attitude than a particular methodology.

Structured Techniques

This has evolved into a set of technologies encompassing the whole software life cycle. It consists of both technical and management issues, ranging from programming to problem solving procedures such as structured programming, structured analysis, structured design, automated techniques, and computer-aided system analysis.

Structured programming

Structured programming focuses on the program itself. It involves structured coding, top-down programming, and stepwise refinement.

Structured analysis

This is the process of defining the information requirements for operation, including system constraints and performance requirements. The functions to be performed are precisely defined, but how these functions work together is not defined. The main output of the analysis is a statement of the function specification and information requirements. This statement bridges between the system analysis and design because the requirements of the system to be built, including functional specifications and constraints, are used as input to the design process.

There are two similar versions of structured analysis: Gane and Sarson,9 and De Marco10 and Yourdon.11 Both are based on structured disciplines such as top-down, bottom-up, divide-and-conquer, graphic presentation, and functional decomposition.

The structured specification is composed of data flow diagram (DFD), processing logic, data store, data dictionary, and data immediate access diagram (DIAD).

A DFD is a network representation of the process (functions or procedures) and the data used in this process. It shows what a system does, but not how it is done. It is the central modeling tool of structured analysis and is used to partition the system into a process hierarchy. DFD can be used in a top-down design and can be exploded to lower level of details.

DFD only provides an informal description of the system. The data dictionary is used to add rigor to the specification. It is a set of formal definitions of all data including data elements and data relationships.

A process specification describes what happens inside a process box in a DFD. It follows the input-process-output specification which is also used for the construction of databases. The DIAD is designed for identifying the data elements or record types (relations) needed in on-line processing or immediate access.

De Marco defines the structured analysis as a seven-step process; Gane and Sarson define a similar process, but in five steps. Both approaches are informal applications of the functional decomposition method to divide the problem into its component parts. But neither methodology offers sufficient guidelines to provide the rigor necessary for defining a precise, computable specification. Gane and Sarson's book,9 improves upon their 1977 edition and adds more rigorous specifications.

The greatest improvement in structured analysis is a change in the system specification from a large, unreadable tome to a user-friendly graphic model. A higher-level DFD can be drawn quickly to show the general picture of the system. Perhaps the most impressive improvement in Gane and Sarson's structured system analysis is that the data stores and data dictionaries are directly related to relational database systems.

However, there is no checking mechanism in DFD. It emphasizes process components; data analysis receives only secondary attention. Structured analysis techniques should only be used for small systems and simple problems with formal data modeling. For complex systems, DFD can be used to sketch a high-level view of the system. But beyond this point, more rigorous analysis and specification methods should be used to develop a precise and computable specification. The higher-order software methodology is better for this. It is the topic in the next few sections.

Structured analysis and design

Structured analysis and design emphasize a higher-level view of the system which is then applied to the lower-level process. The concept of modulization was implemented by standardizing the structure of program modules and restricting the interfaces between modules.

System design

System design is defined as finding ways to satisfy the information requirements identified in the system analysis phase. It is the process of planning how the system will be built, by determining the procedural and data components, and planning how these components will be organized to produce the information needed. Functional specifications, information requirements, and constraints defined in the analysis phase are used as the input to the design process.

Structured Design Methodologies

Functional decomposition, top-down, and bottom-up are the main techniques used in system design, but the most widely used structured design methodologies are top-down
Top-down design

The top-down design begins with the most general function and breaks it down into subfunctions which may follow with a bottom-up approach for detailed design of the subfunctions. The step-wise refinement process is the key technique. Input, function, and output should be specified for each module. Details should not be delayed until late in the design process.

Structured design methodology

The structured design methodology (SDM) defined by Steven, Myers, and Constantine, and Yourdon, is a composite of techniques for system design. Actually, it is a refinement of the top-down design method and consists of four steps.

First, a data flow diagram (DFD) composed of processes that operate on the data is drawn to represent the system. These processes and data link together as the basis for defining the programming components. The DFD is built from four basic components: the data flow, the process, the data store, and the terminator. The DFD shows how data flow through a logical system and a procedure for processing applications. For details on drawing the DFD, De Marco and Martin completely describe the techniques.

Second, a structure chart is drawn to represent the program design. It is a hierarchy of functional components. The structure chart is derived from the data flow diagram produced in the first step. There are two design strategies for guiding the transformation of a DFD into a structure chart: transform analysis and transaction analysis. Page-Jones has a detailed description of the two strategies.

Third, the design is evaluated by using transform analysis and transaction analysis and is also measured by the techniques of coupling and cohesion of modules.

Fourth, the design is prepared for implementation. This is called packaging design and is the process of dividing the logical program design into physical implementation units; these units are called load units. Each load unit is brought into memory and executed as one unit by the operating system. The purpose of packaging is to make sure that the components of the physical system can be executed in an actual computer environment. The packages should be functionally related with high cohesion but loose coupling. At the end of analysis the system is packaged into jobs and job steps. A job is a sequence of job steps. A job step is composed of a main program and its subprogram. The data flow diagram is packaged at this point by establishing three boundaries: hardware boundaries, batch/on-line/real-time boundaries, and operating-cycle boundaries. Each job step is defined in terms of a structured chart which is packaged into executable programs and load units. The smallest possible load unit is one module. Yourdon's structured design has a complete description of procedure and techniques.

The drawback of the structured design is that the rules guiding the transform analysis, transaction analysis, and factoring techniques are very vague. They offer no real improvement over the simple functional decomposition method of top-down design. The introduction of such new terminology as afferent streams, efferent streams, and central transforms confuses rather than enhances the top-down design process. The biggest problem of the structured design is that the design process will break down when used for the design of complex and large systems with many input, output, and transform processing streams. The combined strategy of using transaction analysis to divide the system into more manageable pieces and using transform analysis to design each piece can be difficult to apply in practice. No guidelines are offered for accomplishing the top level division. Besides, there is a lack in data design which constitutes a serious omission in the structure design methodology. The role of databases or data dictionaries in program design is not discussed. This limits the usefulness of the structured design methodology to designing small and simple programs with simple file systems. For these simple problems, the top-down design methodology is easier to use.

The Jackson design

The Jackson design methodology is also a refinement of the top-down design method and separates the implementation phase from the design phase. The main difference between the Jackson design and structured design is that the former is based on analysis of data flow. The former is data-oriented and the latter is process-oriented. The Jackson method advocates a static view of structure while the structured design focuses on a dynamic view of data flow. The Jackson method derives the program structure from data structure. It assumes that the problem has been fully specified and that the program will be implemented in a procedural language. Thus, system analysis and program implementation concerns lie outside the design process. The design process first defines the data structure and then orders the procedural logic or operations to fit the data structures. There are four steps in the design process. First, each input and output data stream is described as a hierarchical structure. Second, all the data structures are combined into one hierarchical program structure. Third, a list of executable operations required to produce the program output from the input is prepared. Then each operation on the list is allocated as a component in the program structure. Fourth, the ordered operations are written in the form of structure text, a formal version of pseudocode.

The major strength of the Jackson design methodology is that it emphasizes data structure design. It produces a hierarchical program structure from hierarchical data structures. The major weakness of the Jackson design is that it is very difficult to apply directly to real world problems. The design process assumes the existence of a complete and correct problem specification. This is rarely possible for most application situations. Another weak joint is that it is limited to simple programs. Third, it is batch-processing-oriented, which is not an effective design technique for on-line systems or database systems. In simple program applications it is an overkill; in complex situations it provides very few guidelines for managing the problems.

It is fair to say that the Jackson design methodology is more difficult to use than other structured design methodologies;
The Warnier-Orr design

The Warnier-Orr design methodology is a hybrid form of LCP and SPD. LCP stands for logical construction of programs while SPD represents structured program design. The Warnier-Orr design uses set theory from mathematics to describe program design. A set is an ordered collection of objects. It also adopts a top-down design method and functional composition to derive program design. There are six steps in its design procedure. First, the program output is defined as a hierarchical data structure. Second, the logical database is defined. It consists of all the data needed to produce the program output. Third, event analysis is performed to define all the events that can affect the data elements in the logical database. Fourth, the physical database is developed, which is composed of the primary data items in the logical database. Fifth, the logical process that is needed to produce the desired output from the input is designed. Sixth, the physical process is designed to complete the program design.

The Warnier-Orr design methodology is similar to the Jackson design because both are data-driven and derive the program structure from the data structure. They both work with hierarchical data structure only and stress that logical design should be separate from physical design. The main difference between the two is that the Jackson design merges old input and output data structures to form a single program structure while the Warnier-Orr design derives the program structure and the input data structures from the output data structures. Therefore, in the Warnier-Orr design, the program output completely determines the data structure which, in turn, determines the program structure. It is, therefore, an output-oriented analysis in addition to being a data-driven approach.

The Warnier-Orr design methodology limits the design to a strict hierarchical model for data and processes. Network-like data structures cannot be described. The fact is that not all databases are hierarchical. It also does not address the design of database systems or the role of data dictionaries. This is a serious omission from what is claimed to be a general-purpose design methodology. Warnier claims that the control logic is not part of the logical design, and therefore provides no guidelines for control logic design.

In general, the Warnier-Orr design methodology is suitable for small problems with simple, report-oriented systems. For these problems the methodology provides an easy-to-follow design method, but for such small and simple report designs, the methodology is too tedious to follow in detail.

AUTOMATED DESIGN METHODOLOGIES

Structured techniques for system design suffer common weaknesses in that they are only fit for small systems and not fit for complex problems. The human brain is limited and has difficulty handling complex details with precision. Structured techniques often make mistakes so higher-level automation is the predominant trend. The system analyst will create his designs at a workstation in a computer-aided fashion. The work station may use mini-microcomputers to help create and edit data models or diagrams. These diagrams may become a language themselves, and from this language executable code will be automatically generated.

USE. IT is a completely general specification language that can be applied to any type of system. It is mathematically based so that it completely checks the internal consistency of specifications and generates bug-free codes. USE. IT would not rate high on a scale of user-friendliness, but it is clearly shown that mathematically-based rigor applied to tools can be built into user-friendly constructs for system specification. USE. IT has an automatic documentation generator for generating documentation in U.S. Department of Defense format. Design and documentation of systems are closely linked.

Data modeling is vital for database design. The task is too tedious and error-prone to be done by hand. It should be designed and maintained with computers. For example, the canonical synthesis is very tedious to apply to large installations unless it is automated. Once computerized, it automatically produces fully normalized data models. A data model shows the functional dependencies and associations among data items. Data redundancy in different areas can be avoided.

DDI's data designer is one of the data modeling tools. The user's view and functional dependencies can be input to the data designer, which synthesizes them into a non-redundant data model, plots the result, and produces various reports for data administrators. If the input functional dependencies are correct, the output is in third normal form.

The future of computing lies with computer-aided design in which the analyst builds applications at a workstation screen and the machine generates executable code. Much computing will be decision support operations done by individuals at workstations. Personal, departmental, and central computing will be tightly interlinked. Most end users' computing will not involve programming, but will employ report generators, spreadsheet facilities, decision support tools, personal databases, and many other packages.

The data models are kept in a computerized form, and the users at work stations use computerized tools to build applications that use data. These tools are selected or designed to aid and automate system analysis and design, and application development.

ITERATIVE DESIGN METHODOLOGIES

Iterative design methodology (IDM) is an interim method between structured design methodologies and integrated system design techniques. As indicated earlier, the traditional information system development life cycle methodologies are troublesome, costly, complex, and time consuming. There are no rigorous rules which the system analyst can follow to develop computer information systems. There is almost no database arrangement with the traditional ISDLC. The development of structured design techniques in the 1970s rep-
resents an advancement in the search for better methods. Unfortunately, structured design methodologies are only fit for small systems and simple programs. For large systems and complex programs, the structured methodologies are completely overwhelmed. Also, there are no rigorous rules which can be used for guiding the system design process. The structured methods try to change the system design methodologies from an art to a science. This effort has only been partially successful. When faced with a very complex situation like decision making, rigid methods such as structured techniques are totally inappropriate to the ever-changing environment.

Practitioners and academicians alike are diligently searching for alternative methods to meet the needs of the executives because the tasks they face are basically unstructured. Both the traditional ISDLC and the structured design methodologies are inadequate for decision support system development. An alternative, iterative design methodology, is advocated by Sprague and Carson. This is an interim design methodology which can be used until an effective integrated design methodology can be developed for developing decision support systems.

The iterative design methodology is based upon the premise that the environment of decision making is volatile. It is difficult, if not impossible, to completely identify the information requirements before system design begins. Under such circumstances, the best way is to identify an important subproblem, develop a small but usable system first, then gradually refine it, and finally expand this usable subsystem to other areas. After all the subsystems are developed, efforts will be directed toward integrating these workable subsystems into a unified whole. So far there is still no perfect solution to system integrations. Tremendous progress has been made recently in individual areas. Eventually integrated methodologies may be developed.

The iterative design methodology also adopts the divide-and-conquer tactic. It first defines the requirements of DSS, then defines what capabilities the DSS can provide. This methodology was developed by Sprague and Carson and named the ROMC approach; it will be discussed in the next section.

There are six objectives which include three types of tasks and three types of support needed in DSS:

The three types of tasks are: (1) to support all types of structures—structured, semistructured, and unstructured, (2) to support all levels of management—strategic planning, management control, and operational control; and (3) to support the communication between all levels of decision makers—dependent, sequential, and pooled.

Three types of support are needed: (1) it needs to support all phases of decision making—intelligence, design, and choice; (2) it needs to support a variety of decision-making processes but not be dependent on any one type since each person’s cognitive organization and style are different, (3) it should be easy to use and modify in response to changes in the user, the task, or the environment.

Gory and Morton combined Anthony’s levels of management and Simon’s types of decisions into a paradigm which can be used for identifying to which categories applications belong.

In a design DSS for poorly-specified environments, Sprague and Carson suggested an approach called ROMC (representation, operation, memory aids, and control mechanisms). The ROMC is intended to identify requirements in each of three capabilities of DSS: databases, analytical models, and query interface. This approach is based on a set of four user-oriented entities: R, O, M, and C. The capabilities of DSS from a user’s point of view are that it provides representations to help users conceptualize and communicate the problem or decision situation; to operate, analyze, and manipulate those representations; to provide memory aids for the users in linking the representations and operations; and to control the entire system. Through this approach, the gap between the requirements and the capabilities of DSS can be reduced.

The iterative design of a specific DSS consists of an iterative addition or deletion of Rs, Os, Ms, and Cs. These combinations of R, O, M, and C are carried out by the three capabilities: databases, analytical models, and query dialogue. The ROMC approach is a framework for identifying the end user’s requirements and the capabilities of DSS which will support these requirements. It is a process-independent approach. One set of representations and operations may support a variety of decision-making processes. The differences among the decision-making processes are more or less in the sequencing of operations and in the decision makers’ interpretation of representation rather than in the set of representations or operations to be used in the process.

The major drawback of the iterative design methodology is that there are no rigorous rules which can be followed by the system designer. When one specific DSS is developed, it may not fit the second application and another one may have to be developed. It is, of course, costly and time consuming. With the automated design tools, the cost and time needed for system development may be reduced, but the effectiveness of this approach may still be a problem. The solution may be to have integrated rules which can be followed for system integration. The integrated design methodology is the topic of the following section.

SYSTEM INTEGRATION METHODOLOGIES

The Present State of the Art

So far four major information system design methodologies have been discussed. There is not one among them that is completely fit for designing complex systems. Perhaps there will never be one that is perfectly fit. It may also be true that for some tasks it is more appropriate than for others. For example, structured design methodologies are more appropriate than the traditional ISDLC if it is applied to structured tasks and environments. For volatile and unstructured tasks and environments, iterative design methodologies are more useful than both the traditional ISDLC and SDM. Automated design methodologies (ADM) are in the infant stage but are progressing rapidly. If ADM can be integrated with other design methodologies and tools, a new design methodology will emerge. Actually, this new methodology has popped up and is called systems integration methodology (SIM).
The ISDM integrates many design techniques: the top-down and bottom-up are the two most important techniques in the design tool kits. The top-down design is a macro system design beginning with the general function or the root of a hierarchical tree and working down to the lower level leaves of the tree. It is usually based on information planning which lays down the information development guidelines in the system analysis phase. These guidelines are used in the system design phase as input. Under such high-point guiding principles, a bottom-up micro system design can be carried out for detailed program development.

Another important characteristic of the SIM is that information system development usually involves two groups of people. One group is business oriented, and the other is technologically oriented. During the system analysis phase, the main duty of the system development team is to define the information needs. The business-oriented people are in a better position to define the information requirements of the organization. When the problem and the information requirements are defined, the technical people take over and identify the technologies which can be used to meet the requirements. This is usually carried out in the system design phase. Actually, there is no clear-cut separation of the two phases. There is some overlap between the two, and sometimes they are iterative. For more information on unified design methodology, please refer to Lee.32

Conceptual Models for Integration

In office automation there is another effort working toward integrated methodologies. Bracchi34 classifies the conceptual office models into four types: data-based models, process-based models, agent-based models, and mixed models. Most of the recent office models belong to the mixed category. This is actually an integrated method for office information development. Office information systems are one of the main components in the overall information construct.

The mixed models consist of more than one type of element as the basis for system specification, and define relationships among these elements. The semantic office system (SOS) is an example of a mixed model. SOS classifies office elements into three different submodels: the static, the dynamic, and the evolution submodels. The static submodel contains the specification of data-related elements such as documents, dossiers, and agents. The dynamic submodel contains the specification of operations and activities performed in the office. The evolution submodel specifies, through two sets of rules, both the normal evolution of office work and the possible structural modifications of office tasks. Some of the rules support office activities with information for performing normal operations or for decision making; other rules may be used for triggering the automatic execution of operation. These mixed specifications and operations actually work in an integrated fashion. This new mixed technique turns many original models into mixed models. For example, the OFFICETALK-D35 is transformed from OFFICETALK-ZERO36 (a data based model) and information control nets37 (a process-based model).

Integrated Models

Lyngarack and McLeod developed an integrated design methodology which can be applied to distributed environments. It does not employ any one of the traditional high-level database models. For example, SDD-1, distributed INGRES,39 R*,40 and the system for managing structured messages41 are more or less related to the relational data model. The simple object-oriented database model (ODM) was defined, and now this model has been extended to work in distributed environments and is called distributed object-oriented database model (DODM). It concentrates on distributed information management at object-level. Both ODM and DODM provide end users with the basic primitives for object definition, manipulation, and retrieval. The DODM supports object sharing, location transparency, and access control, and allows relationships to be established among objects in the databases. The best feature of the DODM is that the location transparency can be supported without having to have a central data structure.

The DODM can be incorporated for use with multifunctional workstations. Each workstation has a unique name. Several workstations can be grouped together at a single mode in the computer network. All the workstations have the same interface for communication, but they do not have to establish the same data model. The DODM models the distributed environment as a logical network of many workstations. Each workstation contains information such as databases. The distributed workstations can be thought of as a logical network of distributed databases.

The ODM and DODM are both modeled as a collection of objects and relationships. The relationships between objects are modeled as structural objects. Each object in the database corresponds to a relation in the set of all database objects.

Implementation of ODM can be straightforward by using existing database technology. A prototype has been built by using the INGRES relational database management system42 running UNIX operating system. The object heap is implemented as a direct access file.

DODM is a very simple database model for specification of objects and relationships in a logical network of databases. Mechanisms are provided to allow relationships to be established across database boundaries, objects are allowed to be copied and moved from database to database, and access control and object sharing among databases are accommodated.

Neither ODM nor DODM is a high-level system for inexperienced database users. Both lack semantic expressiveness, mechanisms for integrity control, and high-level operations for database integration. The contributions were made in defining a small set of fundamental concepts and constructs that can be used as building blocks for integrated systems.

Efforts are being made at the University of Southern California to design and develop a personal information system and experimental prototype called INFOBASE, which is intended to provide information management facilities to support a wide spectrum of transactional, professional, managerial, and clerical processing. Based on the above modeling concepts and facilities recently developed, integrated systems...
can doubtlessly be achieved. These integrated systems are capable of supporting multi-functional workstations, database processing, and personal computing. INFOBASE is not only an example of integrated systems but is also intended to provide a basis of information management in engineering applications including software engineering and CAD/VLSI design.\textsuperscript{46,47}

**System Integration Recapitulated**

After four decades of effort by both computer scientists and management experts, a legitimate model that can be used for system integration is still missing. Though complete integrated models are still in short supply, there are plenty of quasi models which can be combined for unifying the various components of an integrated system. The various models and components relating to system integration have been illustrated and examined, but further efforts are needed to integrate them into a unified whole.

Before integration, the question must be asked, “What is the integration for?” The answer might be, “for transaction processing, decision making, and/or office automation.” According to the definition of an integration system, it may be designed to provide any one, or all, of those functions. For transaction processing, the system design is easier; for decision making it is harder; for both transaction processing and decision making it is difficult. For transaction processing, decision making, and office automation, it is even more difficult because the current technologies are more appropriate for structured tasks, and the tasks a decision maker faces are largely unstructured.

In addition to the structure of tasks, the current trend is toward distributed processing. Theoretically, distributed processing involving concurrent control, security, and recovery is not a problem, but actually integrated systems in a distributed environment are still in an infant stage. Though system integration has a long way to go, the basic technologies required for system integration do exist. What is needed now is to try patiently using the integrated methodologies introduced in the previous sections.

So far the structured techniques, structured design methodologies, automated design methodologies, iterative design methodologies, and system integration methodologies have been discussed. Before putting them in an appropriate format, we should keep in mind the basic ingredients needed in an integrated system. In order to solve problems, task analysis is vital for identifying information requirements. These requirements are the basis for preparing a system specification which, in turn, is used for system design.

During the system design stage, the basic job is to identify the alternate ways to meet the information requirements identified in the system analysis stage. In the system design stage, data and process analysis are necessary. Data and process analysis involve data modeling, database systems, and processing logic. All the structured techniques and methodologies can be used. In addition to the above-mentioned techniques, data communication and networking, expert systems, fourth generation languages, computer-aided design techniques, and system automation methodologies should be used and integrated into the overall construct of the integrated systems.

**Integrated Systems**

After comparing 12 commercial systems in 1983, MacFarlane classified them into three kinds of systems: software-only, hardware/software, and time-shared.

In the software-only systems, the organizations buy the software and run it on their new or existing hardware. Though these are not total systems, they can perform professional and managerial tasks and can be tied into broader hardware and databases.\textsuperscript{34}

The hardware/software systems are total turnkey systems. All provide local and remote area networking capabilities serving as integrated systems.

In the time-shared systems, the organization only needs to install work stations. All the software and processing power can be obtained through these work stations. This is a cost-effective method for obtaining the services of integrated systems.

There are many more disciplines and methodologies working toward integrated systems development. It is beyond the scope of this paper to go into detailed discussion of these efforts. Interested readers may refer to Lee,\textsuperscript{33,44,45,24} Martin,\textsuperscript{22,23,3,46} Lyngback,\textsuperscript{47} and Bracchi.\textsuperscript{34}

**CONCLUSION**

This paper has intensively investigated the various CIS development techniques: the traditional ISDLC methodology is loosely defined, and there are no rigorous rules which can be followed for effective CIS development. The various structured design methodologies are only fit for small systems and simple programs. There are also no rigorous rules for the system analysts and designers to follow for checking the consistency of the system being designed. When facing complex situations, they will be quickly overwhelmed. The automated techniques and system integration methodologies are still in their infant stage. Remarkable advancement in automated techniques has been reported recently. There is no doubt that integrated systems will be within reach in the near future. Some prototypes of integrated systems have already been built. Though they are still in the primitive stage and not for user-friendly use, it can be expected that near perfect integrated systems are imminent due to the rapid growth in technology.

**REFERENCES**