DevelopMate: A new paradigm for information system enabling

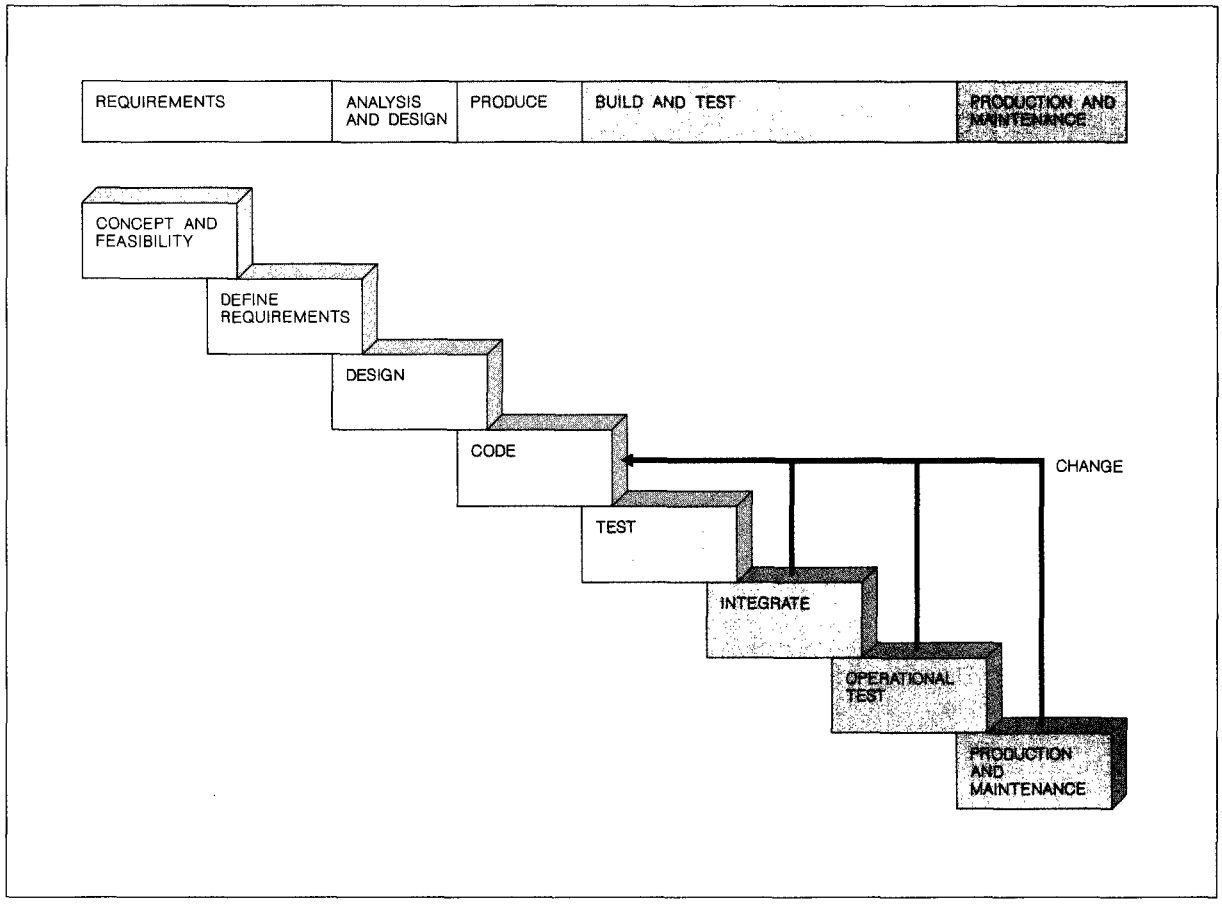
by K. P. Hein

This paper discusses a new approach to the use of information systems that is based on enterprise information system modeling concepts. This approach is primarily oriented to the enterprise expert, who is considered to be the individual most familiar with the functioning of a particular area of the enterprise information system. The approach is not primarily oriented toward the data processing professional. The paper discusses the phases of the approach and how the DevelopMate™[1] software product supports some of those phases.

Over the past twenty-five years, we have witnessed an evolution in the development of information systems by data processing professionals. First, specific programs were written to solve a particular problem. These were later combined via some intermediate step, such as a sort, into a program set (application) that operated on common data. Then, the industry evolved from building a single application to creating integrated, shared-data systems. As this evolution has taken place, it was discovered that integrated information systems are more complex, multidimensional, and highly interrelated. These systems affect an ever larger part of the enterprise than had initially been foreseen. Unfortunately, development approaches have not kept pace with this evolution in complexity. Most data processing organizations are approaching shared-data system development the same way they approached the development of single programs.

The traditional application development life cycle is shown in Figure 1. This process is not capable of supporting the requirements of integrated data-system development. For example, no provision is made to understand how the system under development affects or communicates with other information systems in the enterprise. There is no accounting for how many systems there are or what data are affected. It does not take into account the quantity of data being passed or the data standardization that might be required. Instead, the system under consideration is treated as an isolated entity that is assumed to have no association with any other system in the enterprise. Expensive design, programming, and testing are done for individual system parts, without an understanding of final interfaces and dependencies. As Figure 1 suggests, integration of the various application parts is performed only after considerable time and effort have been expended. The results of the integration activity send the developers back to the design definition phase to correct any interoperability problems that might come to light. Thus the cycle is repeated at considerable cost. This approach is analogous to building an airplane using different

Figure 1 Standard development life cycle, granular view



subcontractors. As they meet on the appointed day to join fuselage, wings, landing gear, communications equipment, and interior cabin, it is discovered that the major assemblies do not connect. The seats are too large for the available space in the fuselage, the wings cannot support the weight of the airplane, and the landing gear is too large to fit into the appropriate place for storage.

In manufacturing, this problem is avoided by incorporating the integration step into the early design stages of the product, so that subcomponent builders may build to predesigned and well-understood interfaces. In the information system manufacturing business, however, a common practice is to isolate applications from their major subsystems and treat them as if the rest of the environment did not exist. As a result, the user and the enterprise pay for the resultant rework and redesign in the form of lost time, dissatisfaction, and production loss.

This cost is manifested in successive systems because they must recover the expense of integrating previous systems. Each successive system becomes ever more expensive with the cost borne by the sponsoring organization, even though the benefit is corporate wide. The more the already-existing systems need to be integrated with the current system, the higher the cost of the current system. Therefore, it must be concluded that integration at this price is undesirable and that the old model of system development does not serve the construction of large-scale integrated systems sharing common data.

The result is that the user community feels that its requirements, which are often very straightforward,

require an inordinate amount of time to be fulfilled. When the system is delivered, many requirements and requests are not implemented or do not perform as specified, and the cost is substantially higher than initially estimated. Such systems are often delivered late as well. The data processing professionals have good and sound technical reasons for the late delivery and inadequate performance of the systems. However, these technical reasons further alienate the end users from the data processing professionals because end users are often not well versed in technical matters and do not identify with these technical issues.

For these and many other reasons, end users have embarked on creating their own solutions. Usually, they purchase a microcomputer or minicomputer in order to gain control over their own information environment. They feel that this step will allow them to have systems that meet their requirements. The time required to implement the system is under their control, and they need no longer beg and plead with others who appear to be indifferent to their problems.

As much as these users' actions may satisfy the requirements, when viewed from the enterprise perspective, other problems arise. Instead of sharing data as a corporate resource, data are being managed as organizational property by narrow groups of users. Rather than sharing a single information source from which consistent decisions can be made, we find that many information sources leave management to make decisions based on conflicting, duplicate, and contradictory data. The problem created by the dispersal of information has been made worse by the fact that the applications reside on different hardware. Standardization of terms and definition of data is nonexistent, and the processing of the same fact multiple times is widespread and results in inefficiency, duplication, and poor decision making.

To solve these problems, a development approach must allow for the creation of large, integrated enterprise information systems at much higher productivity levels than heretofore realized. The approach must also take into account the requirements of the enterprise to manage data as a resource in a standardized, integrated, and consistent way. It must also involve full participation and responsibility by the eventual owners of the system. This approach must make it possible for the designers to create overall enterprise-wide information systems as a well-functioning, integrated architecture. This requires that the method employ well-recognized design and architectural principles used for complex system building. Furthermore, the system users must be actively involved in the architectural phase and exercise overall control over each part of the integrated information system. Additionally, the users should be able to express their system specifications and business rules in their own language and test these specifications for validity and accuracy. And finally, the whole approach must be completely supported by software in order to gain significant productivity improvements.

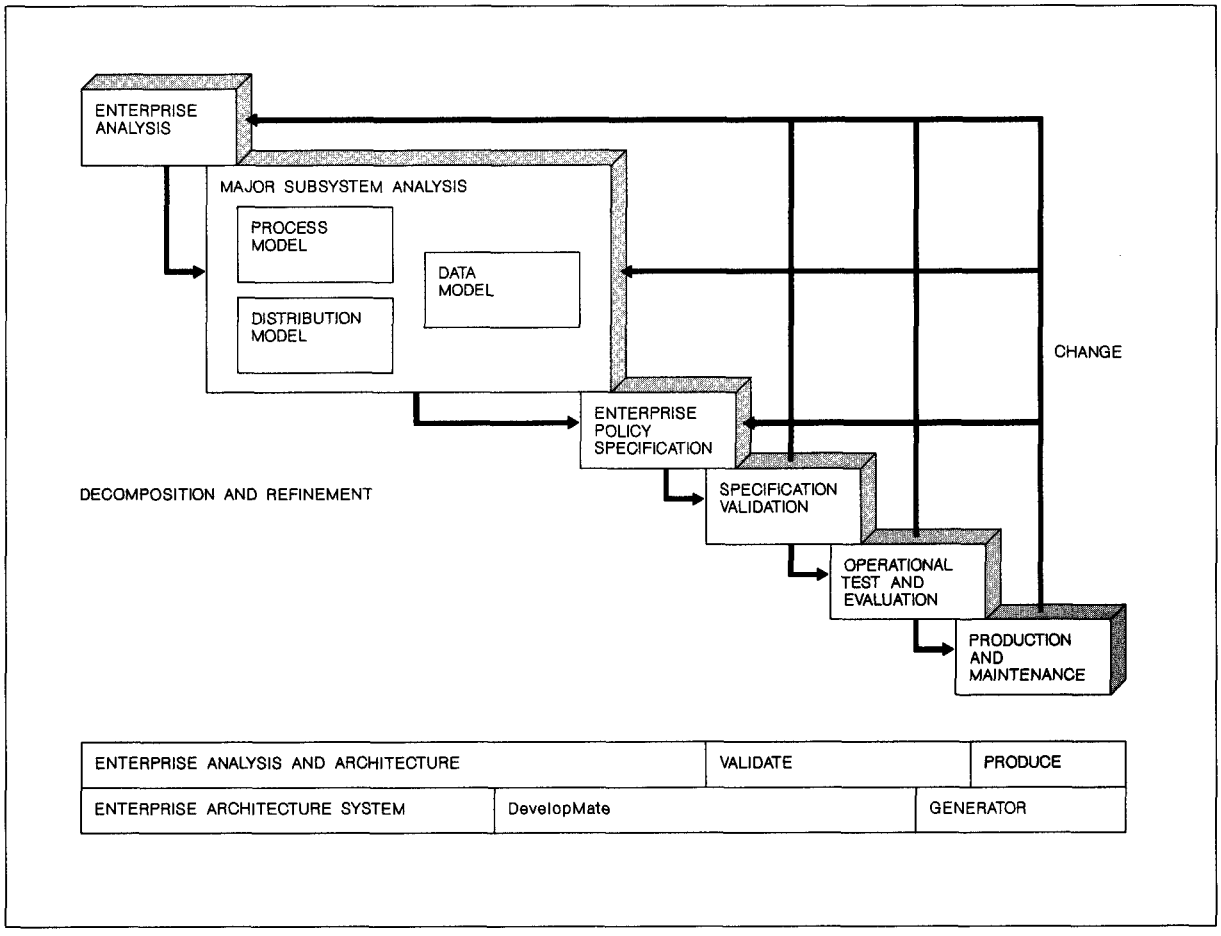
The effect of the immediate and interactive testing capability is to save expensive rework.

A new approach

A new approach to using information systems based on enterprise modeling and a program product called DevelopMate™, is illustrated in Figure 2. This approach capitalizes on the business professional's desire to be involved and to take responsibility for his or her system. The user is an extension of the total available enterprise information system development resource. The user is supported in this by a set of software support functions that store and share their information in a facility such as the IBM Repository Manager™(RM). These support functions, as embodied in DevelopMate, allow the professional to communicate in terms of the business processes, data views (such as invoices and bills of material), organizational units (such as departments or individuals), events, locations, and so forth. This information is defined as *enterprise rules* and takes the form of models, constraints, report or panel images, and enterprise policy statements.

At the start, enterprise experts define an enterprise-wide information system architecture. Following that, parallel subsystem architecture and specification begins. This step is taken with the secure knowledge that expensive rework caused by integration problems can be avoided, because the new approach,

Figure 2 New approach to development activities



as part of its philosophy, takes integration into account at the start.

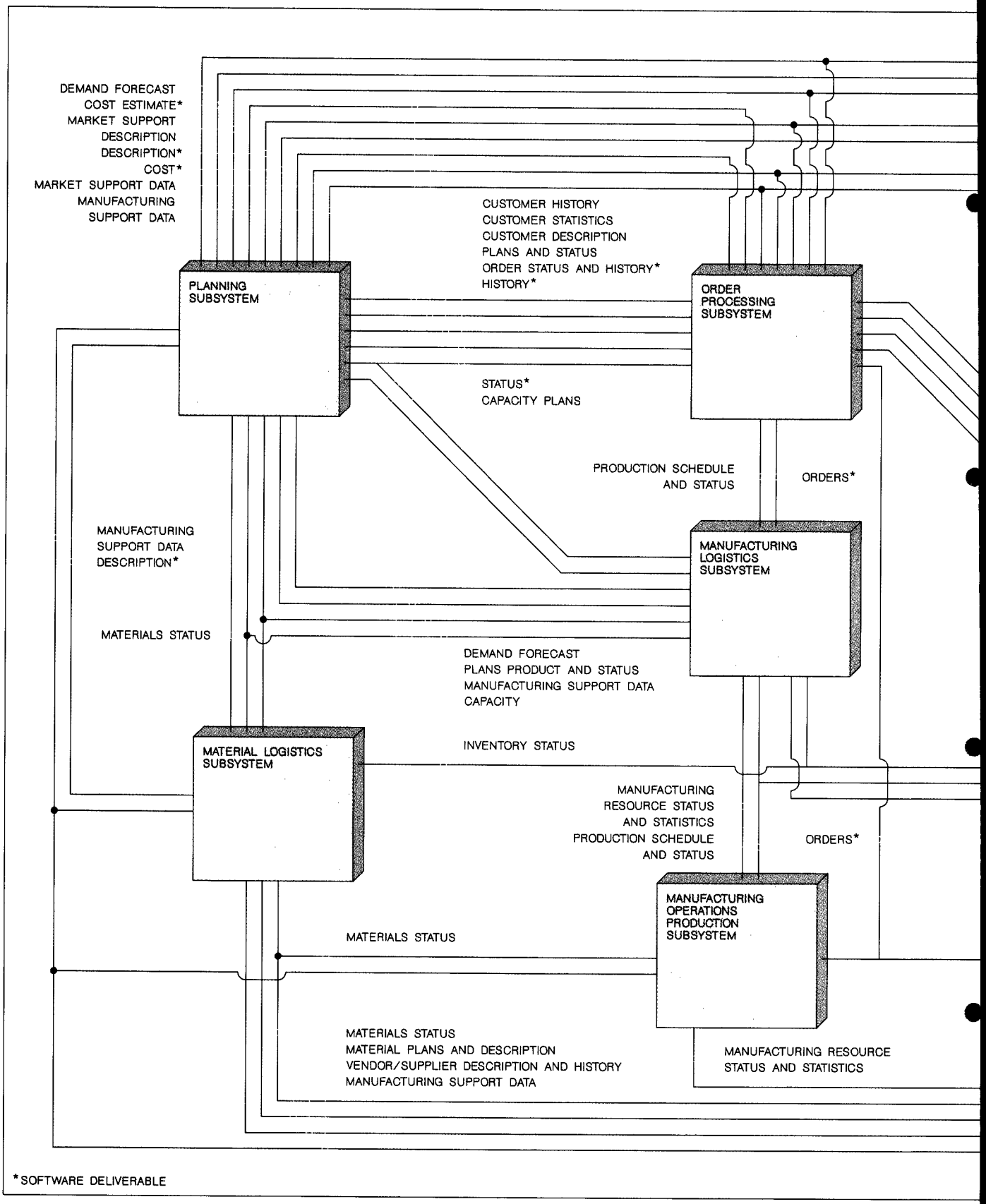
When the architecture has been captured and stored by RM, the enterprise expert can test his or her specifications interactively using DevelopMate. This provides the ability to determine how the system will apply the specifications at execution time. The system will let the user correct ambiguous statements immediately, without propagating them down to later stages of development. The effect of the immediate and interactive testing capability is to save expensive rework during later definition phases. Once this concurrent specification checkout is completed and the tested set of specifications performs as desired, the user's business subsystem is ready to be generated into target production environments, such as IMS/VS and CICS/VS, using a generator that can understand and translate the specifications into more conventional data processing terms. The information may also be used to support traditional program coding.

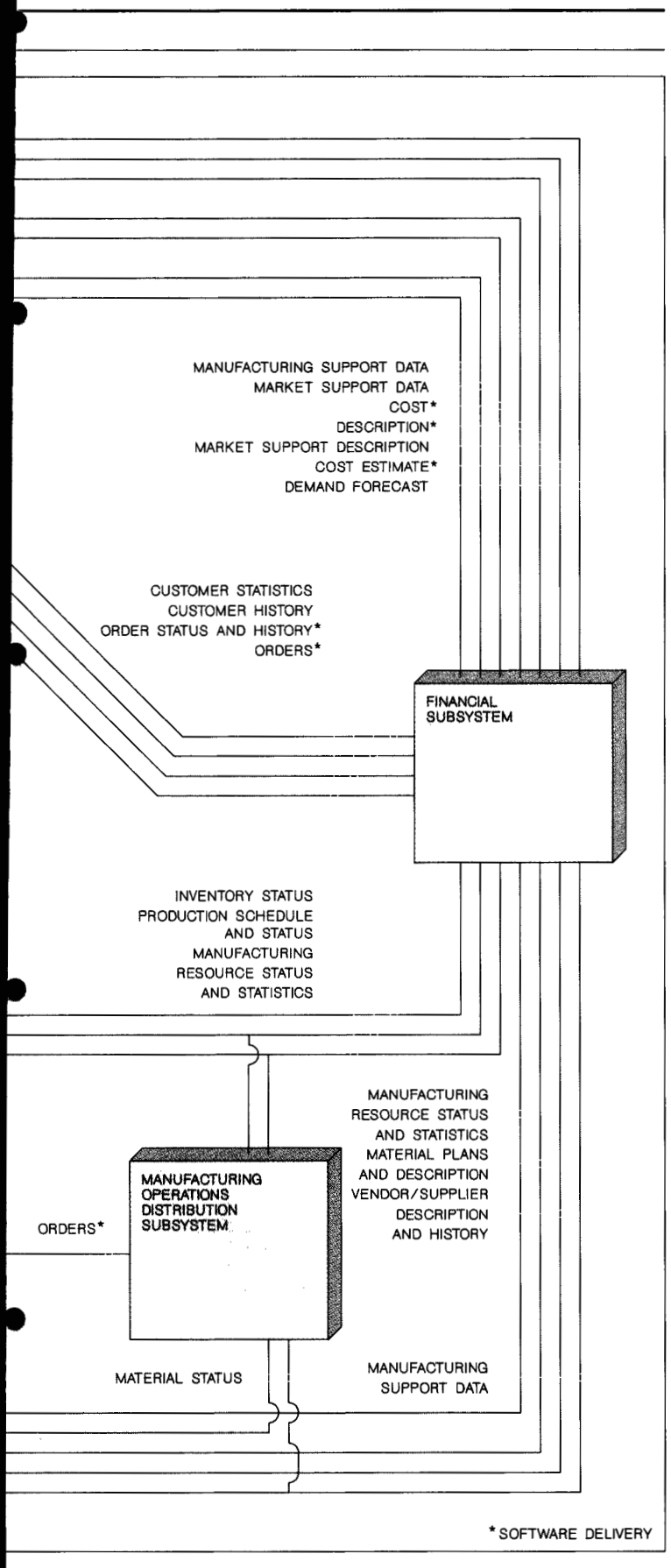
With all of this concurrent architecture and development activity supported by various software facilities, all the architecture information is integrated and shared via a common, active knowledge base stored by RM. RM-managed data contain all the information system specifications in terms of models, rules, and formats. This concept is explained in more detail below.

DevelopMate

DevelopMate allows for the definition, decomposition, and refinement of a previously created infor-

Figure 3 Example process model





mation system architecture; it is not an architecture development system. Rather, it stores the results of the architecture study phase and provides for additional decomposition and refinement of the architecture. DevelopMate also allows the user to define enterprise information policies or rules in a nonprocedural manner. It also allows for the definition of the enterprise information model in process and entity-relationship form, report formats, panel layouts, and other defining features. These definitions and refinements can be checked and tested interactively for consistency using the facilities provided and reported in printed form through predefined reports or by using Query Management Facility (QMF™). If desired results are not achieved, the user is free to make modifications to the specifications and test them again. All the information is stored by RM for later use or for concurrent sharing with other information-system facilities.

In the following sections, the various phases presented in Figure 2 showing the new approach are discussed. It is shown how DevelopMate may support these phases.

Enterprise analysis

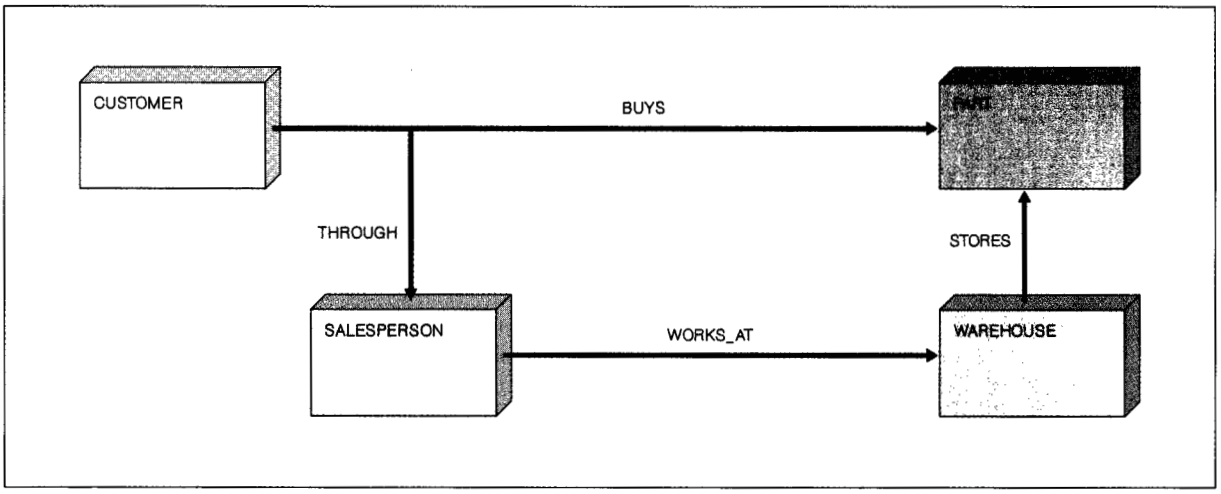
Figure 2 shows the steps involved in the new approach. The first step is an enterprise analysis phase which develops the high-level enterprise information system architecture. This step can be automated with the use of the Information System Model and Architecture Generator (ISMOD 5785-FBA), a program offering available from IBM. ISMOD can support various study methods that deal with information system architecture.[2]

The important point is that the enterprise analysis provides an overall blueprint of the information system architecture based on common sharing of data. Once this common blueprint has been established, further top-down decomposition and architecture refinement can occur concurrently in each one of the subsystems identified at the higher level. The result of this activity is an enterprise process model. When the process model is completed, it may be mass imported into the DevelopMate system, or it can be entered via a set of DevelopMate panel dialogs if ISMOD was not used.

Major subsystem analysis

Through the creation of the models defined in this phase, the enterprise expert can define the informa-

Figure 4 Entity-relationship (ER) model



tion system processes and their relationship to an underlying AD information model of the enterprise. The expert can also define how the data and processes are distributed within the enterprise. This enterprise data model is the vehicle by which system integration is achieved. A comprehensive process and data architecture are created from a very high level down to the lowest functional primitive, called a *minispec*. The minispec has some unique information defined for it in order to make it a machine-executable function. We now expand upon this summary of the approach, starting with the process model. Note, however, the user may define model types to the DevelopMate system in any order.

Enterprise process model definition. Figure 3 shows a sample process model that may be defined to DevelopMate. It shows the processes performed by the enterprise in network form. Decomposition can be performed on each process into lower-level networks until the lowest-level process (minispec) has been reached. At the lowest level, all the minispecs may have more than one input, but they must have only one output. The minispec becomes the unit of implementation either as a manual task description or as a programming specification for mechanized implementation.

The inputs to and outputs from the process are referred to as *data views*. Data views are documents that are needed to perform the process in the enterprise information system. A view is defined as a group of interrelated data arranged to allow the user to derive useful information. An invoice, check stub, bill of material, telephone message form, and computer screen format are examples of data views.

Events may also be defined to identify the conditions under which certain processes are performed. Thus, events act as triggering mechanisms that cause the process to execute when the event is satisfied. An event can be a certain time of day, end of the month, the completion of a process, or creation of a data view. In the initial implementation of DevelopMate, events are defined for documentation purposes only. Event action is performed based on trigger policies discussed later in this paper.

Enterprise data model definition. The process model is not enough to completely describe the enterprise information system. Additionally, a data model that supports the enterprise information processes is required. Through the DevelopMate data model definition dialog, the enterprise expert can express how enterprise data relates.

Referring to the example in Figure 4, it can be seen that a customer buys a part through a salesperson who works at a warehouse which stores the part. This data model not only conveys information to the user, but it also allows the enterprise expert to define knowledge of complex interrelationships in a simple, understandable form to the system. Entities and relationships are described by attributes that tell

us more about the particular entity or relationship type. This type of model is known as an entity-relationship (ER) model and was first advocated as a useful tool for data definition by Peter Chen.[3] It is created through the common efforts of enterprise experts and data administrators.

The enterprise data model connects to the process model via the data view. The entity attributes that are specified to be displayed on the data view become the connecting thread. An invoice, for example, displays various entity attributes, such as customer number, customer name, part number, part name, part price, and salesperson number. Thus, each data view is supported by information described in the data model. What is most important in this data representation is that business knowledge is imbedded in the model in the form of relationships such as: buys, stores, works at, or through. Thus, an enterprise expert can express highly technical data interrelationships in a simple and friendly way.

Enterprise analysis provides an overall blueprint of the information system architecture based on common sharing of data.

The model shown in Figure 4 can be made more expressive by applying association rules. For example, one might define the relationship type WORKS AT as M:1, i.e., many to one. This allows the user to define the rule that a salesperson may work at only one warehouse and that a warehouse may have many salespeople. Other rules for control and dependency may also be defined.

Enterprise distribution model definition. In this step of the major subsystem analysis phase, the user may define which organizations perform the various enterprise processes, where the organizations are located, where specific processes are performed, and the data required at that location to perform the process. This architecture will be of great importance to communication-network and hardware/software planners who must provide the physical environment in which the eventual information system is to perform.

DevelopMate provides the user with a set of dialogs that may be used to acquire such architecture information, but it provides no support for distribution architecture algorithms. This is left to appropriate architecture facilities.

The minispec. The minispec is defined as the lowest level of decomposition in the process model. It is at this level that processing algorithms transform input data into output data. These algorithms may be written as a job description to be performed by a person or they may be programmed as a transaction for a computer system. Another possibility is that an application-development-oriented expert system may merge the specifications and create a computer-executable function without programming.

In order for a process to be defined as a minispec, it must meet the following conditions:

- Have a superior (parent) process
- Not have any subordinate (child) processes
- Have multiple inputs but only one output

Another difference between a high-level process and a minispec process is that the minispec process may also have additional specifications that can be used by a computer system to create an executable function. These additional specifications are:

- A data submodel that tells the system the subpart of the enterprise data model to be used by the minispec to create the desired output view
- A format that defines the physical form of the output view when it is presented to the user
- An output data view to be produced by this minispec process that provides the list of entity attributes (data elements) that are to be shown or are required to compute values to be shown on the format
- Any policies that pertain to the particular minispec

Enterprise policy specification

DevelopMate allows the enterprise expert to define enterprise policies that implement his or her particular rules of operation. This is the step in the development approach that gives logical life to the process

and data models defined in the previous steps. (See Figure 3.) Policies are statements made by the enterprise expert specifying the action to be performed when certain conditions occur during the execution of enterprise processes. These policies are divided into the following four main categories:

- Integrity
- Derivation
- Trigger
- Security

Policies may be expressed as conditional or unconditional. A conditional integrity policy may be expressed as follows:

If the EMPLOYEE NUMBER is not in the range of 1000–5000, issue the message "Employee number out of range."

An example of a derivation policy is the following:

If the employee is salaried, the EMPLOYEE GROSS PAY is EMPLOYEE YEARLY SALARY divided by 12.

An unconditional derivation policy may be the following:

STATE TAX is GROSS PAY * 0.075.

A conditional trigger policy may look as follows:

If PART QUANTITY ON HAND is less than PART ORDER POINT, execute the PURCHASE ORDER WRITING minispec process, passing the PART NUMBER and the PART ECONOMIC ORDER QUANTITY on to the next step.

These policies can be globally defined for the entire enterprise and by specific, local processes. Global policies are defined once, such as the integrity policy just shown. The policies can be applied any time the employee number attribute is referenced at execution time. Therefore, when a change in the policy is made—such as a range change from 1000–5000 to 1000–6000—the enterprise expert can reference the policy and change it. From that instance, all references by all minispecs across the entire system will reflect and correctly implement the new policy. A local policy is one that is unique to a particular minispec process, and any change made to it is reflected only in the specific minispec. This allows the user to contain a change to a specific minispec process without affecting the entire system.

Note that it is possible with this concept to store enterprise policies in a nonredundant manner, making future maintenance much easier than when logic is duplicated in many application programs.

Specification validation

In DevelopMate, the above specifications—including the policies—are made independent of one another. This means that no procedural definition is made, leaving the enterprise expert free to concentrate on the specific definition rather than being concerned with detail logic specification. However,

After the specifications have been defined to the system, tested, and found to work, they can be moved into production.

DevelopMate is capable of combining the various definitions in such a way that processing is performed in the proper order to yield the required results. The enterprise expert can call for an execution of definitions at any time, and the system will produce the result. Any specification errors can be corrected immediately and then retried.

The significant point is that after the specifications have been defined to the system, tested, and found to work as desired, they can be moved into production by causing a generator facility to create programs for specific production environments. It may further be noted that the system can only execute properly when the expert's knowledge has been defined to the system. This means that without documenting the required information through the system model and rules, there is no functioning system. Documentation becomes a necessity to system operation, yet it is a by-product of the development environment and not a separate step which, in the traditional development process, is usually not performed.

Operational test and evaluation

DevelopMate implements an operational test evaluation phase of the new development life cycle by allowing the actual every-day users of the processes to test and evaluate the ease of use of the system components. If a change is required, the enterprise expert can immediately change the specification in the enterprise model definitions, and the new result can be tested. Furthermore, if the specifications and policies are global, all other usages in the system will be immediately corrected.

Documentation is an integral part of the change procedure and cannot be bypassed.

After validation by the enterprise expert, who is also the responsible user, the operational test and evaluation phase is almost nonexistent. The reason for this is that the user has worked closely with the system from the beginning and therefore has made the specifications to meet particular requirements.

Production and maintenance

DevelopMate does not affect applications in the main production environments of IMS/VS or CICS/VS. Instead, it stores the enterprise architecture and associated rules in the RM-supplied entity-relationship model, so that a follow-on generator may use it to create a production system for the required target environment.

In DevelopMate, change to the architecture and rules is easily accommodated. If business policies change or the process model needs to change, the change can be applied by the responsible enterprise expert to the DevelopMate information stored by RM. The significant advantage here is that the user must go back to the model and policy specification to affect the change. This means that the documentation step is an integral part of the change procedure and cannot be bypassed, as is so often the case today.

Independence

Note from the preceding description of the approach and from its underlying computerization, that all data about the enterprise information system is stored by RM. It is, therefore, extremely important that RM be capable of operating in various hardware and software environments. This level of independence is provided, because RM is a Systems Application Architecture™ (SAA™) component.

Some considerations for automation

The ER model. The entity-relationship (ER) model is a device that is widely used by analysts today, because it allows the analyst to describe a body of knowledge in picture form. An example ER model is shown in Figure 4. The following are definitions of some of the terms used in connection with the ER model.

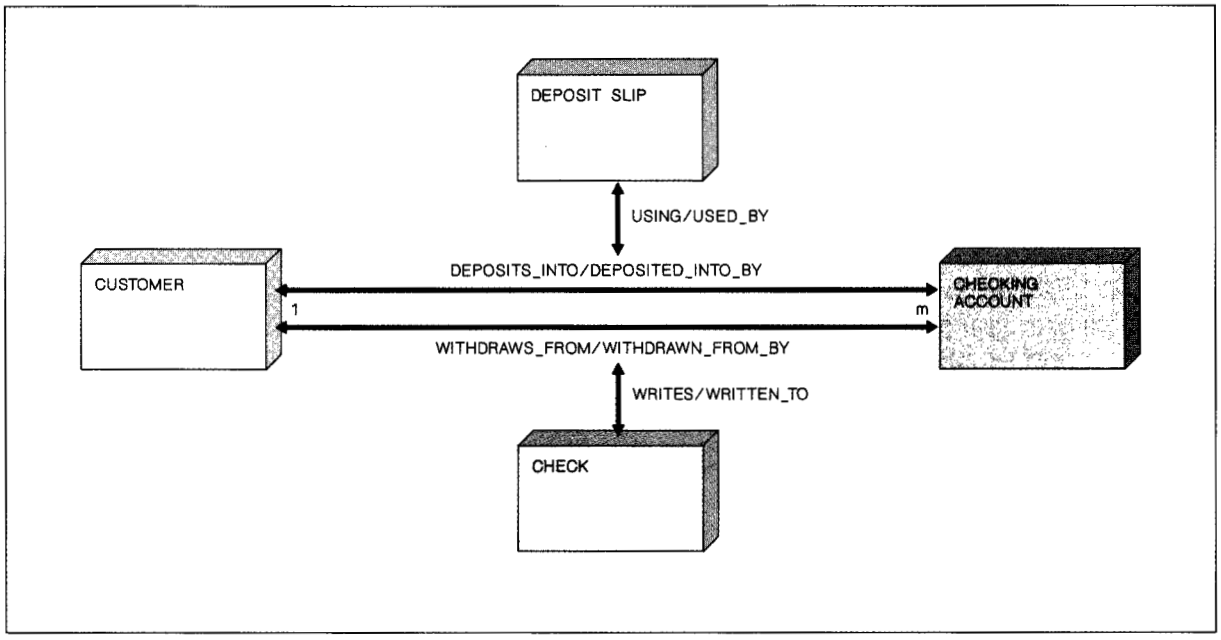
An *entity* is a person, place, thing, or idea about which the user wants to collect and maintain data in order to manage a particular resource. An entity must be uniquely identifiable and may be collected into entity sets. In a model, the description of an entity set is called an *entity type*. Examples of entities may be a customer, general ledger account, student, automobile, etc. Notice that each instance of an entity type (the entity) must be uniquely identified by one or a combination of its attributes.

Entity *attributes* are characteristics or properties of a particular entity type. For example, the entity type CUSTOMER may have the attributes CUSTOMER NUMBER (identifier), CUSTOMER NAME, CUSTOMER STREET ADDRESS, CUSTOMER CURRENT BALANCE, etc.

A *relationship* connects two entities or, if desired, an entity and another relationship. For example, the entity type STOCKHOLDER owns the entity type STOCK CERTIFICATE. In this case, the verb OWNS is the relationship. As for the entity, the relationship description in the model is called a relationship type. Its instances are the relationships between real entities. Relationships can define forward and inverse directions. Thus, we may say that a stock certificate IS OWNED_BY a stockholder. Additionally, relationships may have certain rules. For example, a stockholder may own MANY stock certificates, but a stock certificate is owned by only ONE stockholder.

With these explanations we can now see how easy it is for the enterprise expert to express knowledge in

Figure 5 Entity-relationship (ER) example



pictorial form using the ER model. In Figure 5, entity types are represented by rectangles, relationship types by lines with arrowheads, and constraint rules with 1 (ONE) and m (MANY).

Let us use the following semantic description of a banking environment:

A customer deposits money into a checking account using a deposit slip. The customer writes checks to withdraw money from the checking account.

If we use small capital letters for each of the uniquely identifiable entity types in the preceding paragraph, we obtain the following sentence:

The CUSTOMER deposits money into a CHECKING ACCOUNT using a DEPOSIT SLIP. The CUSTOMER writes CHECKS to withdraw money from the CHECKING ACCOUNT.

The words in small capitals represent entity types with uniquely identifiable occurrences and are all nouns. For example, the entity type CUSTOMER has unique occurrences of IBM, GE, AT&T, etc. Thus any noun that represents a uniquely distinguishable occurrence of its type—such as customer—is an entity type. If we capitalize each of the words that expresses a relationship between the nouns, we obtain the following:

A customer DEPOSITS money into a checking account using a deposit slip. The customer WRITES checks and WITHDRAWS_FROM the checking account.

The example could be carried further to show that adjectives become the attributes of entity types and that adverbs become attributes of the relationship. For example:

A customer who has a CUSTOMER NAME, CUSTOMER ADDRESS, CUSTOMER NUMBER, and CUSTOMER TELEPHONE NUMBER deposits money into a checking account using a deposit slip. The entity type CUSTOMER then has the attributes name, address, number, and telephone number.

Additionally, it is possible to define constraints about the relationships. To do this, expand our original paragraph and capitalize the constraints, as follows:

A customer may deposit money into MANY checking accounts, using ONE deposit slip for each deposit. The customer may write ONE check to withdraw money from ONE checking account, but may have

MANY withdrawals per account. Checking accounts may have only ONE customer. (Partnerships and joint accounts are treated as one customer for this example.)

The advantage of this type of representation can be seen immediately. In the first place, it is easy and straightforward for a user who is not a data processing professional to define complex information in very simple terms and pictures. Also, a model of this type can be analyzed by a program in order to make execution decisions. When policies are added to the entity and relationship type attributes, the power of the ER model in logic processing becomes apparent.

Therefore, DevelopMate provides this ability to allow easy definition of the enterprise data environment in ER form. Our experience has shown that the ER facilities, as provided by RM, serve this purpose in a powerful way. This allows the enterprise expert to communicate with the system at the user's conceptual level while at the same time being able to physically store and retrieve data from underlying data management servers.

Repository Manager. The ER facility is used to allow the expert to define every aspect of the enterprise information system to one common shared collection of information. The definition contains all data necessary to describe the desired information processing functions. RM is used to completely integrate every aspect of the enterprise model, including policies, screen and report formats, and enterprise analysis outputs (such as process and data models). Thus, the RM-managed data become the integration vehicle for the functional components supporting the new life cycle, the specification environment, and the verification facility.

With this facility, changes can be made simply by changing the information in RM's data store. Because RM-managed data are actively consulted at specification test time, the change can be implemented easily and swiftly throughout the enterprise information system.

Fill-in-the-blank panels are provided to allow the enterprise expert to define knowledge to the system. The panels use words familiar to the expert. This makes it unnecessary to learn a data processing language that is understood only by the computer.

Consistency checking. DevelopMate provides extensive consistency checking to ensure that all specifications required to execute a set of minispecs are met. This means that extensive checking of the process and data models is done under user control at various levels of detail. Furthermore, DevelopMate automatically invokes consistency checking at various points in the specification phase.

In the facility under discussion, checking of the process model is done at various process decomposition levels. The system ensures that processes are properly connected vertically through parent-child relationships and that horizontal connections exist through data views. Additionally, data views are checked for existing and proper definition.

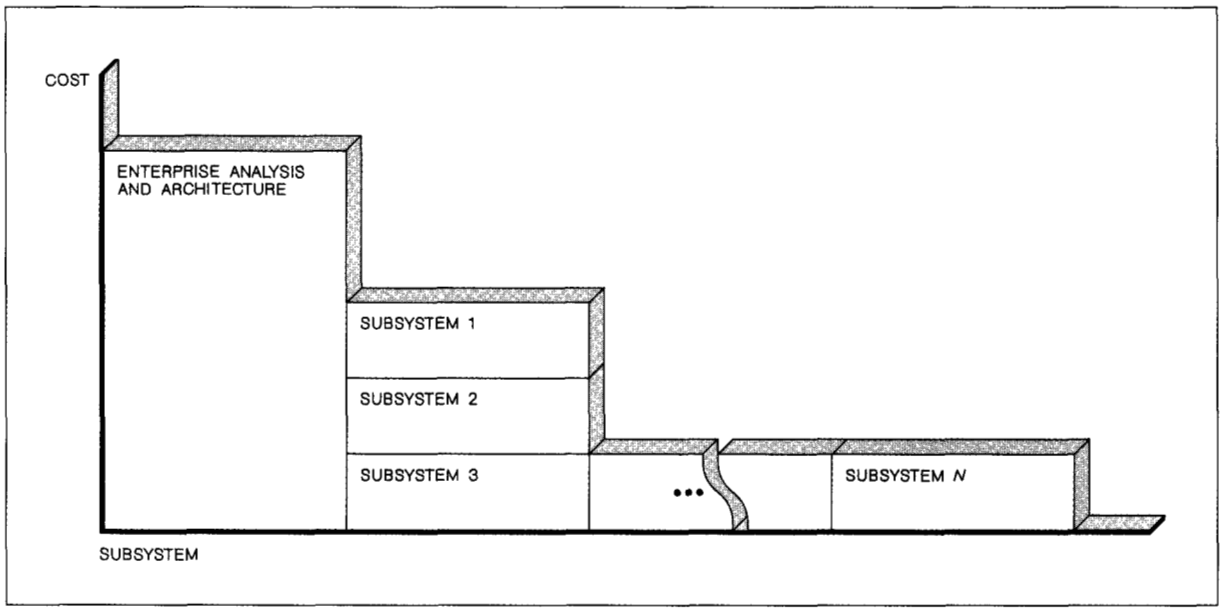
The enterprise requirements analysis phase is a highly stimulating and creative experience for those involved.

The data model is similarly interrogated to ensure that views have formats defined for them; that entity types have attributes; that the attributes defined on a view are present in the entity type specified; and that the required relationship types exist. Lists of error messages are created to allow the enterprise expert to provide additional information about the environment and make corrections.

Benefits. Based on preliminary testing, we have found that the approach presented here, with the supporting software, provides significant benefits over the traditional process of information system development. The development team (consisting of the enterprise expert, with support from an analyst and database administrator, both of whom are knowledgeable about the system) can define and prototype high-quality integrated data systems in a very short time.

The enterprise requirements analysis phase is a highly stimulating and creative experience for those

Figure 6 New approach subsystem costs



involved. The enterprise expert develops requirements and has a simple facility in which to store them and from which they may be retrieved at a later time. Computerized support (using the Information System Model and Architecture Generator [ISMOD] program offering[4]) provided at this point allows the team to tackle very large and complex information systems with great efficiency. Decisions critical to the development sequence are easily made with architecture and flow matrices, user satisfaction ratings, and simulation.

As the requirements and architecture are developed to greater detail, the system specifications emerge. They are stored by RM and are available to the designer or expert for review and modification. The usual lengthy and cumbersome system specification document takes on less significance. Because the enterprise expert and the analyst have made the definitions using RM and have tested them via the interactive validation capability, the need for a formal sign-off procedure disappears. Adjustments and modifications are part of the normal process and are therefore an ongoing activity.

Interactive validation allows the user to check the specifications immediately to determine the result at execution time. This means that when specifications have been executed and pronounced accurate, the particular function or process is ready to be generated into production. Therefore, except for very rare and complex requirements, programmer involvement is minimized.

Application execution is not possible unless the knowledge about it has been defined to the system. Thus documentation is a requirement for system operation and not an additional task. If the documentation is not precise, the resulting executable function may be in error.

Integration of the various I/S subsystems in the enterprise is accomplished in the architecture phase well ahead of any detail specifications. The definition of the data model becomes the integration mechanism for the entire enterprise information system. Using the power of ER models and the capability of relational database facilities, changes and adjustments can be made easily and at low cost. Thus the high cost of integration using traditional methods (because of rework and redesign) are reduced or eliminated.

Figure 6 indicates the enterprise-wide resources invested at the beginning of an architecture and requirements analysis cycle. Even though this may take time and the expenditure of resource, the effort is very cost effective. The reason is that individual

subsystems or subcomponents of the architecture can be implemented at a much more reduced and consistent cost compared to the traditional practice described earlier in this paper.

Maintenance is done by the responsible enterprise organization and can be accomplished in a matter of minutes or hours rather than days and weeks.

Maintenance is done in a matter of minutes or hours rather than days and weeks.

Because specifications are stored nonredundantly, the change can be applied at a precise point and shared throughout the system.

Productivity. The productivity gains are orders of magnitude greater than the traditional methods, because specifications can be directly validated and do not need to be converted to code before testing can begin. The programmer or analyst no longer is the bottleneck. Instead, that person can be trained to be the consultant to the enterprise expert and assist in making the specifications to the system. If very difficult problems arise which may need a traditional programming solution, the skill of the programmer is still available to solve the problem. This mode of operation, however, is now an exception rather than the rule.

The extension of the development process to the responsible end-user community allows a whole new personnel resource to participate in this process, while at the same time control over the resultant functions is maintained by the user. Therefore, a whole new group of people can be involved in solving the application backlog problem of today, within the confines of an overall enterprise architecture.

Summary

The purpose of DevelopMate is to demonstrate and test the feasibility of an automated development method that breaks the barriers to productivity in the traditional development approach. DevelopMate allows the user to develop large-scale integrated, shared-data information systems. Through the proper blending of various techniques and leading-edge software, such concepts as entity-relationship (ER) data modeling, logic creation through nonprocedural definitions, and various design concepts, we have found it possible to create a development environment in which the eventual user participates from beginning to end, controls the quality of the output, and prepares the application for use. Yet the enterprise-wide requirements of shared data, smooth integration of enterprise processing systems, and control of enterprise decision making are incorporated at productivity levels heretofore seldom attainable.

Acknowledgments

I would like to express my gratitude to Vivian Anderson and George Brantzeg for the great assistance they have provided in proving that this concept is viable, usable, and commercially feasible. I also want to thank Jim Sagawa and Tom Duncanson of the IBM Santa Teresa Laboratory for their excellent work in ER implementation research. Their contributions significantly reduced the time required for our efforts. Finally, I thank Robert Tabory for his intellectual, advanced technological, and moral support.

Cited references and note

1. The DevelopMate program number is 5688-36 and the manual number is GC26-4641-0, both of which may be obtained through IBM branch offices.
2. K. P. Hein, "Information System Model and Architecture Generator," *IBM Systems Journal* **24,** Nos. 3/4 (1985).
3. P. P. Chen, *Entity-Relationship Approach to Systems Analysis and Design*, University of California at Los Angeles, Los Angeles, CA (December 1979).
4. M. Veys, *Information System Model and Architecture Generator—System Guide*, Program Offering Manual, LY20-0975, published by IBM Corporation (December 1984); available through IBM branch offices.
5. K. P. Hein and M. Veys, *Information System Model and Architecture Generator—Study Guide*, Program Offering Manual, SH20-6651, published by IBM Corporation (December 1984); available through IBM branch offices.

General references

R. Ambrosetti, T. A. Ciriani, and R. Pennacchi, "An Application Analyzer," *IBM Systems Journal* **23,** No. 4, 336–350 (1984).

Arthur Young Information Technology Group, *The Arthur Young Practical Guide to Information Engineering*, John Wiley and Sons, Inc. (1987).

A. Blokdijk and P. Blokdijk, *Planning Design of Information Systems*, Academic Press, Inc., New York (1987).

Business Systems Planning—Information Systems Planning Guide, Application Manual, GE20-0527.

C. Gane and T. Sarson, *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1979).

W. H. Inmon, *Information Engineering for the Practitioner*, Yourdon Press Computing Series, Prentice-Hall, Inc., Englewood Cliffs, NJ (1988).

J. Martin, *Application Development without Programmers*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1982.

J. Martin and J. Leben, *Strategic Information Planning Methodologies* (Second Edition), Prentice-Hall, Inc., Englewood Cliffs, NJ (1989).

R. L. Nolan, "Managing the Crisis in Data Processing," *Harvard Business Review* **57,** No. 2, 115–126 (March-April 1979).

R. L. Nolan, *Restructuring the Data Processing Organization for Data Resource Management, Information Processing 77*, North-Holland Publishing Co., Amsterdam, The Netherlands (1977), pp. 261–265.

M. M. Parker, "Enterprise Information Analysis: Cost-Benefit Analysis and the Data-Managed System," *IBM Systems Journal* **21,** No. 1, 108–123 (1982).

M. Vetter, "Aufbau Betrieblicher Informationssysteme," *Leitfaeden der Angewandten Informatik*, B. G. Teubner (Editor), Stuttgart, West Germany.

M. Vetter, *Database Design Methodology*, Prentice-Hall International, Englewood Cliffs, NJ (1981).

M. M. Veys, "ISS Pour une Logique de la Productivite," *Information*, IBM Belgium **90,** Brussels (1979), p. 2.

J. A. Zachman, "Business System Planning and Business Information Control Study: A Comparison," *IBM Systems Journal* **21,** No. 1, 31–53 (1982).

Zanthe Information Inc., "ZIM EAR Data Base Management System," Ottawa, Canada.

K. Peter Hein *IBM General Products Divison, Santa Teresa Laboratory, P.O. Box 49023, San Jose, California 95161-9023.* Mr. Hein is a senior programmer with many years of experience in business systems planning, top-down system architecture, and large-scale system integration. He has lectured extensively at conferences and seminars on these subjects in the U.S. and abroad. At present, he leads an advanced technology project which deals with the automation of a complete methodology from the Strategic Information System Plan to application creation with the use of professionals who are not trained in data processing. As an advanced research project in the IBM branch office in Salt Lake City, Utah, Mr. Hein did the basic work that led to DevelopMate. In his present position in AD/Cycle external support at the IBM Santa Teresa Laboratory, he is the architect of DevelopMate. Mr. Hein graduated from the University of Utah with a B.S. and an M.B.A. degree. He is also a graduate of the IBM Systems Research Institute, the U.S. Army Command and General Staff College, and the U.S. Army War College.

Reprint Order No. G321-5397.