AD/Cycle strategy and architecture

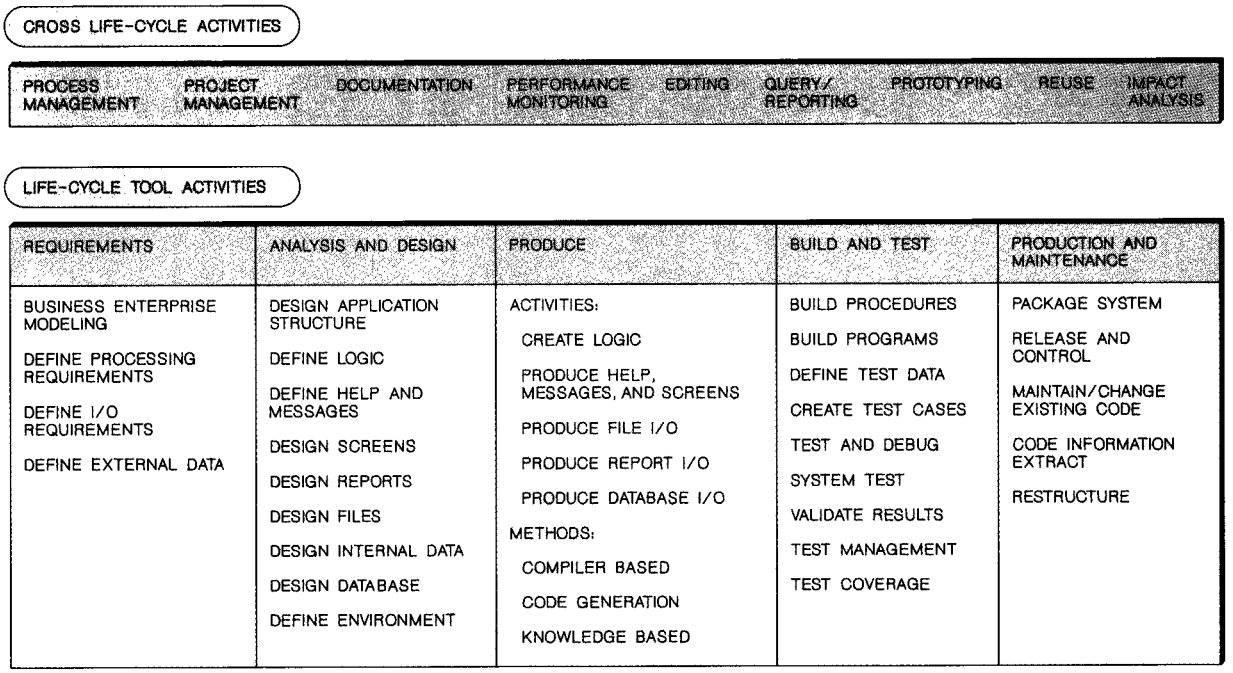
by V. J. Mercurio
B. F. Meyers
A. M. Nisbet
G. Radin

Over the years, IBM has made progress in resolving many of the issues that deal with improving application development (AD) productivity and quality. Systems Application Architecture™, together with IBM's recently announced AD/Cycle™ direction, provides a platform for even greater progress. This paper addresses the IBM strategy that supports AD/Cycle and gives an overview of the major components of the AD/Cycle architecture. This paper is an introduction to other papers that follow in this issue.

The requirement for significant improvements in productivity in the application development process has been present for several years. The importance of this requirement has been the subject of much attention, as evidenced by a recent joint project chartered by the five major international IBM user groups,[1] a top concern from the GUIDE organization,[2] a GUIDE strategy paper on integration of development tools,[3] and a SHARE task force,[4] as well as numerous research and development efforts in industry,[5–11] academia,[12] and government.[13] A recent survey[5] of over 1000 businesses indicated that the backlog for mainframe applications is approximately four years, with the demand for mini- and personal computer applications being somewhat less. In addition to large backlogs of applications, businesses are also faced with the high costs of maintaining existing inventories of applications and a shortage of experienced programming skills. Businesses are frustrated with the current state of application development and are confronted with an inability to generate the applications needed to support their business processes on a timely basis. This inability, accompanied by increasing dependency on data processing applications, puts businesses at a potential disadvantage in the marketplace and becomes an inhibitor to their data processing growth.

The approach to improving application development productivity that evolved in the late 1970s and 1980s with the emergence of computer-aided software engineering (CASE) tools, has been to focus on individual tools to process specific activities within each phase of the application development life cycle. Figure 1 is a representation of the traditional application development life cycle that groups specific application development activities conceptually into five phases: requirements, analysis and design, produce, build and test, and production and maintenance. The activities in each phase are dependent upon the completion of activities in the previous phase. For example, requirements gathering is followed by analysis, analysis by design, design by produce, produce by build and test, and so on. In addition to the

Figure 1 Application development life cycle



activities within the five life-cycle phases, there is another set of activities referred to as "cross life-cycle activities" that span or are common to multiple life-cycle phases.

The approach of focusing on individual tools to accomplish specific activities has resulted in a proliferation of unrelated tools and methodologies that have provided only incremental improvements, none of which has been significant enough to effectively reduce the backlog of customer applications. Some of the reasons for this follow:

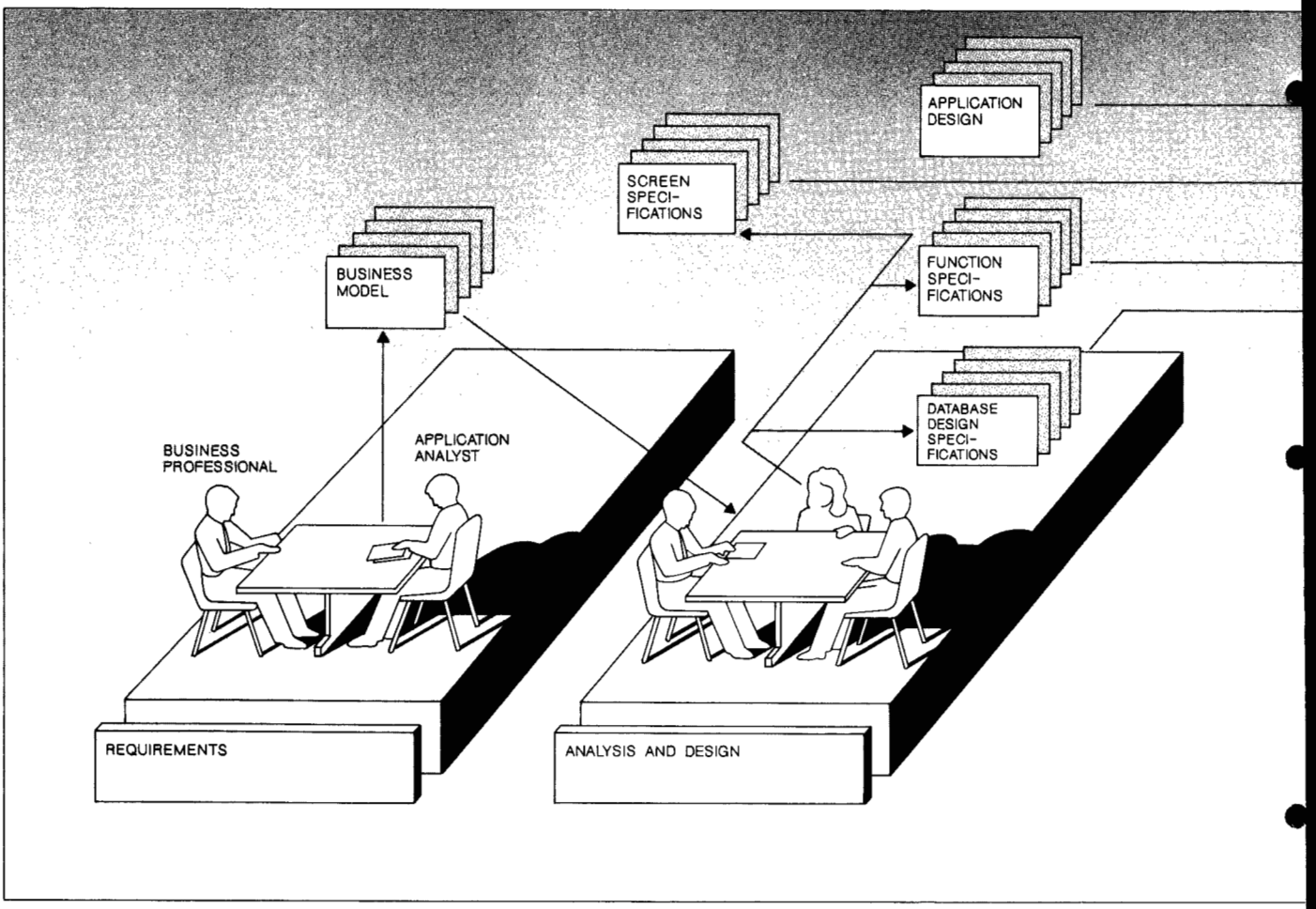
- The methodologies and tools to support the life-cycle activities usually do not share data.
- A lack of consensus on standards has made it difficult for tools to be integrated.
- Where tools have been integrated into application development systems, the systems have typically been built on closed architectures with private data interfaces, making it difficult to add tools or extend existing tools to accommodate different methodologies or technologies.

The management of the application development life-cycle process usually requires that each phase be near completion before the next phase begins. Prototypes are seldom used to provide early views of the product, and requirements and specifications are passed on paper from product planners to designers and from designers to coders, often requiring manual intervention between phases and activities. This proliferation of unrelated tools, methodologies, and manual data transformations results in a development process consisting of a series of discrete steps, with the boundaries of each step characterized by a manual or paper transfer from step to step, as shown in Figure 2. This process is very inefficient and requires intensive use of people and paper.

To address the need for significant improvement in the productivity, quality, and manageability of the application development (AD) process, a collection of offerings called AD/Cycle™ [14] was developed at IBM. Based on Systems Application Architecture™ (SAA™)[15] and supported by a staged implementation of integrated tools, data storage facilities, and an open architecture for interfaces, AD/Cycle follows an application development strategy that is intended to enhance customers' ability to better meet their data processing business needs. The goals of AD/Cycle are to:

- Bring the latest application development technology to customers through IBM- and vendor-supplied tools

Figure 2 Application development today

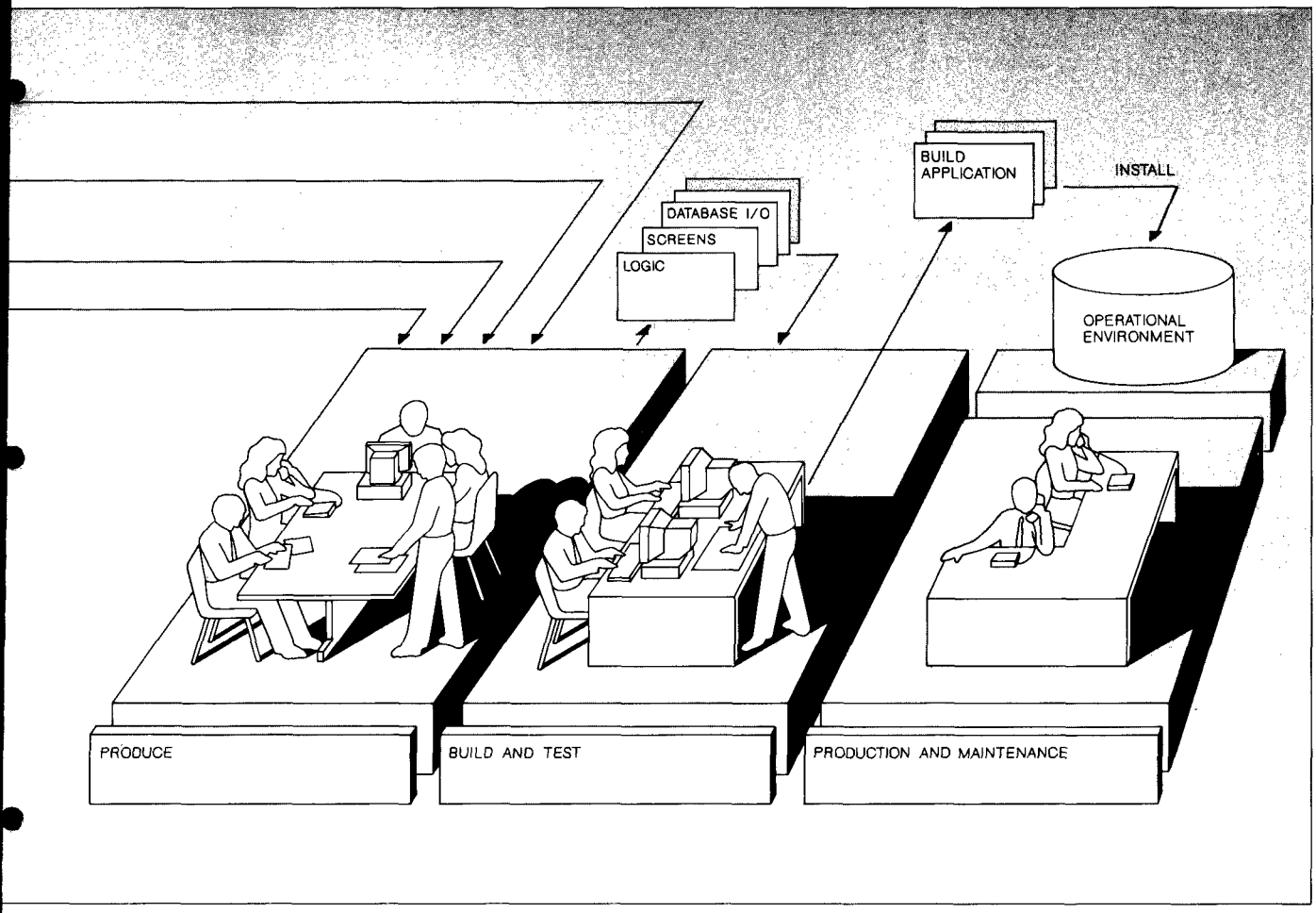


- Enable centralized control and management of customer application development data
- Establish an open architecture and a base set of tool services for integration of application development tools across the life cycle
- Complement and extend SAA

Conceptually, AD/Cycle can be thought of as a framework for, and a set of, application development tools. The framework is provided by an AD platform designed to support the integration of tools through a consistent user interface, workstation services, an AD information model, tool services, Repository Services, and Library Services that provide control for defining and sharing application development data. AD tools will offer solutions for the application development life cycle. Over time, it is planned that AD/Cycle tools will conform to the architecture as is technically appropriate for the function of each tool.

The application development tools can be from several sources: from IBM, from vendors, and from the customers themselves. IBM has entered into relationships with Bachman Information Systems, Index Technology Corporation, and KnowledgeWare, Inc. wherein selected products from these vendors will be marketed through an IBM complementary marketing program to provide offerings that will help to achieve complete life-cycle coverage.

This paper discusses requirements for AD/Cycle tool function and the AD/Cycle tools strategy that IBM



has developed to satisfy these requirements. It also describes the architecture of the AD/Cycle infrastructure and its interfaces, how it fits into SAA, and the technical constraints that have affected its design. This paper serves as an introduction to two sets of papers that follow in this issue of the *IBM Systems Journal*. One set provides more detailed discussions of some of the architectural components; the other set discusses specific AD/Cycle tools.

Application development tool technology

AD/Cycle will provide the capability for application development to focus on enterprise modeling and analysis and design—that is, toward the front end of the application development life cycle. At the same time, AD/Cycle will continue to provide continuous and enhanced tool function across the traditional application development life-cycle phases through a staged set of IBM- and vendor-supplied tools. These tools, while supporting many different technologies, including application generators, traditional third-generation languages, and knowledge-based systems, should provide significant improvements in application development productivity through integration with the AD platform. This section highlights parts of the development life cycle to indicate the tool technology direction and some recommended tool solutions.

Enterprise modeling. When businesses first started using computers, they typically automated their

processes on an application-by-application basis, with each application being a stand-alone entity having its own private set of input and output data files. As the state of data processing evolved, businesses continued to automate more and more of their processes; however, in many cases, they failed to step back and look at the overall company or enterprise structure to focus on how data are used

Early prototyping enables end-user interfaces and functionality to be evaluated.

and how data could be shared by multiple processes within the enterprise. This lack of focus has resulted in enterprises having to maintain multiple copies of the same data and maintain applications whose data underpinnings and process requirements have grown obscure over the years. Not only does this practice lead to inconsistent and unreliable values for the same data item used across different areas of a business, it also creates a situation where it is almost impossible to enhance, modify, and maintain existing applications.

For the above problem, AD/Cycle offers an enterprise modeling approach supported by tools that will assist in the creation of an enterprise model to be validated, analyzed, and then used to generate applications. Enterprise modeling is a technique that enables business professionals and business analysts to define, relate, and validate the data and processes that are used. As changes occur in the enterprise, the model can be updated or related applications can be viewed, and change can be effectively managed. In the AD/Cycle implementation of enterprise modeling, the enterprise model data, i.e., data and process requirements, are centrally stored as entity-relationship data by the Repository Manager™ (discussed below), thus making that data available for subsequent life-cycle tools. This activity also provides a permanent record of the process requirements and the data underpinnings for each application to facilitate future enhancements needed to support a growing enterprise.

The enterprise modeling method of defining processes also enables prototyping of those processes early in the development life cycle. Early prototyping enables end-user interfaces and functionality to be evaluated at the beginning of the life cycle, eliminating much of the rework that characteristically faces developers after code has been developed and an application is in an executable state. Rework at this later stage is many times more costly and time-consuming than at the prototype stage.

The enterprise modeling information is created by business professionals and analysts rather than data processing professionals such as programmers and systems analysts. These business professionals are the experts in their respective processes, e.g., accounts receivable, finance, billing, order entry, payroll, etc. This role change in application development provides two major benefits:

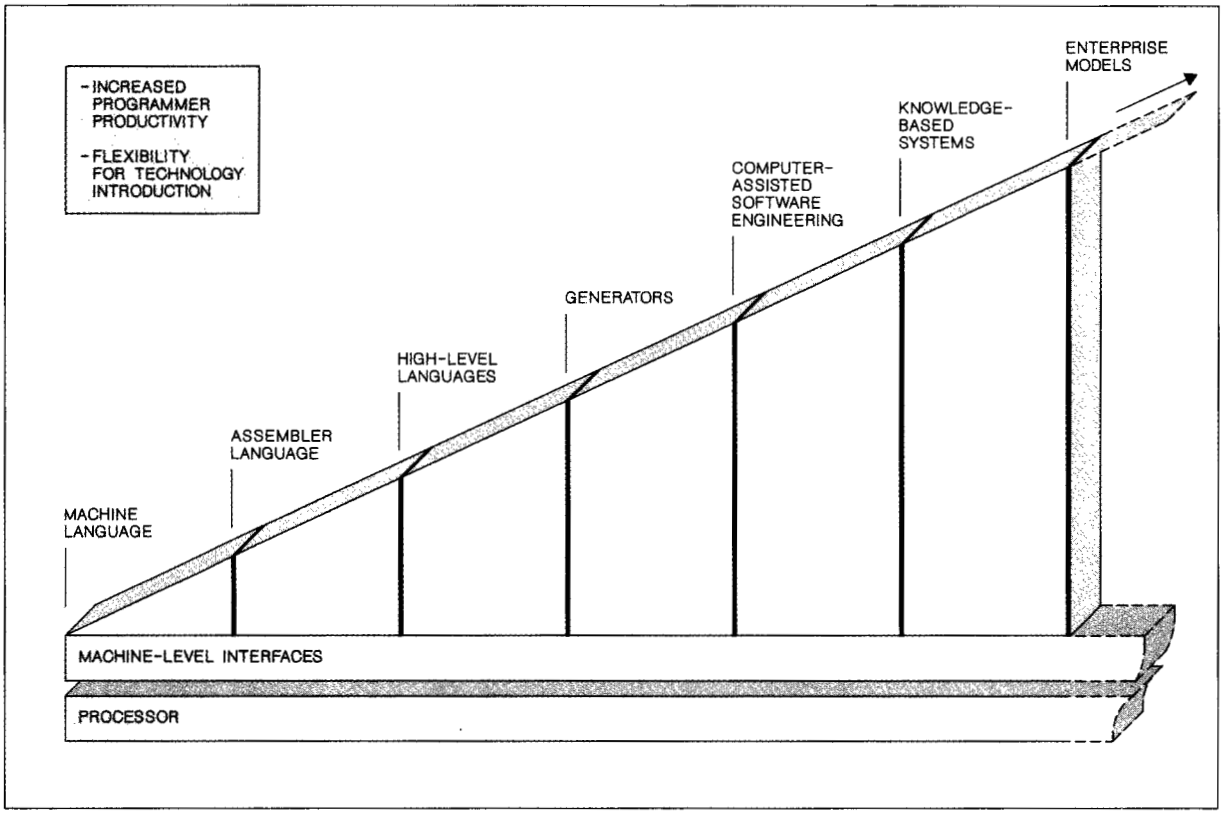
- Knowledge of business processes does not have to be transferred to data processing professionals to initiate the application development process. Costs are reduced significantly and errors resulting from insufficient business process knowledge are minimized.
- Cycle time for application development can be reduced through the direct initiation of application development and changes.

Additionally, assimilation of a process into a sequence of processes that includes input, process steps, and output—as in a payroll calculation and check statement printer operation—requires significant transformation and automation from the enterprise model definition steps to production application generation. Transformations will initially involve discrete steps of function requiring the collective effort of both business and data processing professionals; however, as the integrated tools base develops, more of the transformation should be replaced by automatic tool function.

Figure 3 shows the evolution of programming technology from the use of machine language technology when computers were first introduced, to the current leading edge technology incorporating enterprise models.

AD/Cycle tools offered by IBM that can be used for defining enterprise models and for validating models through prototyping are:

Figure 3 Evolution of programming technology



- IBM's DevelopMate™ that focuses on the definition, refinement, and validation of an enterprise model through prototyping
- Information Engineering Workbench®/Planning Workstation from KnowledgeWare, Inc., for capturing, modeling, and analyzing data about an organization and its use of information
- PC PRISM™ from Index Technology Corporation for helping organizations develop and analyze enterprise models and other planning models so that they can better align their business and systems objectives

Additional information on enterprise modeling can be found in Reference 16.

Analysis and design. In the analysis and design phase, business requirements defined in enterprise modeling can be used to help develop application design information.

The selection of analysis and design tools is dependent on the design methodology established by an enterprise. The AD/Cycle open framework supports the integration of analysis and design tools that use current software engineering principles, modern design methodologies, and common diagramming techniques. AD/Cycle tools offered by IBM that can be used for analysis and design are:

- Information Engineering Workbench/Analysis Workstation and Design Workstation products from KnowledgeWare, Inc., for refining and analyzing end-user requirements and for logically defining information systems based on the analysis of end-user requirements
- Excelerator® from Index Technology Corporation for developing process and data models, validating design information, prototyping screens and reports, and generating system documentation

- The BACHMAN Re-Engineering Product Set™ from Bachman Information Systems that includes tools for advanced data modeling and database design

Generators. The use of a generator is frequently seen as a more productive way to produce an application. Applications created by generators may not be as error prone as coding in a procedural language, and the level of application specification is often less detailed. Whereas IBM's Cross System Product/Application Development (CSP/AD) is the strategic SAA application generator, it is planned that AD/Cycle evolve to provide more automatic creation of generator input. An initial step in this direction is provided by the external source format function of CSP/AD that allows CSP/AD to be used along with other application development tools such as Index Technology's Excelerator and KnowledgeWare's Information Engineering Workbench/Analysis Workstation and Design Workstation products.

See Reference 17 for additional information about CSP/AD.

Languages. Today's application development world has significant financial investments in procedural languages, sometimes called third-generation languages. The vast majority of existing customer code has been written using third-generation languages, and recent use of the C programming language indicates the continuing importance of procedural languages for developing sophisticated software.

In AD/Cycle, third-generation language products will be enhanced to provide integrated functions to increase the productivity of the development programmer. Increased programmer productivity should be achieved through integration of language-sensitive editors, preprocessors, compilers, front-end tools, static and dynamic debuggers, and other support functions. This integrated function should give cohesive edit/compile/debug capability and will facilitate the integration of code from generators and knowledge-based applications with traditionally produced code. It is planned that information captured from the front-end analysis and design tools, such as prototypes, code skeletons, data structures, screen definitions, and report formats, will be accessible by users of tools supporting the third-generation languages.

Most large third-generation development projects rely heavily on library management systems to control source code versions and synchronize programmer access to code. In AD/Cycle, Library Services will provide the function of library management as well as the function of configuration management that automatically keeps track of module interdependences and synchronizes compilations and link-edits when necessary.

Integration will extend to the execution environment. A common run-time environment would streamline calls between programs written in different languages. With communication among languages, customers will be able to take advantage of the special strengths of each programming language, without having to write function all in one language. It is also planned that language debugging and test analysis tools will be integrated with editors and compilers to provide a consistent context for programmers as they test applications.

Process management is one of the key functions provided in AD/Cycle.

This capability will evolve to take full advantage of the productivity of the programmable workstation, from program creation through unit testing. Over time, it is planned that AD/Cycle will give the development programmer the capability to edit/compile/debug and unit test applications on the workstation, thus offering improved productivity and usability. The SAA languages will continue to support system test and execution on the host for workstation-developed applications.

Knowledge-based systems. Strategic application-enabling tools for knowledge-based systems will participate in the AD/Cycle tools strategy. Many application solutions can be developed with knowledge-based systems alone, but many can be more effectively developed using a combination of knowledge-based systems and other approaches such as procedural code. The intent is that developers of procedural and knowledge-based code applications will be able to develop entire applications with tools that work consistently and that provide for seamless

tool-to-tool transition. (IBM's The Integrated Reasoning Shell, Expert System Environment, and KnowledgeTool™ are application-enabling tools for knowledge-based systems.)

Reference 18 describes the role knowledge-based systems will play in AD/Cycle.

Testing and maintenance. Testing new applications and maintaining existing ones are key activities in the development life cycle, along with the process management facilities to manage and control these activities. It is our plan that new tools for analysis, creation, management, and coverage of test data will be provided as part of a comprehensive verification environment. IBM will provide aids such as the COBOL Structuring Facility to help users understand and maintain existing applications. Additional integrated tools will help programmers assess the impact of making changes to existing systems, which is the most time-consuming part of application maintenance.

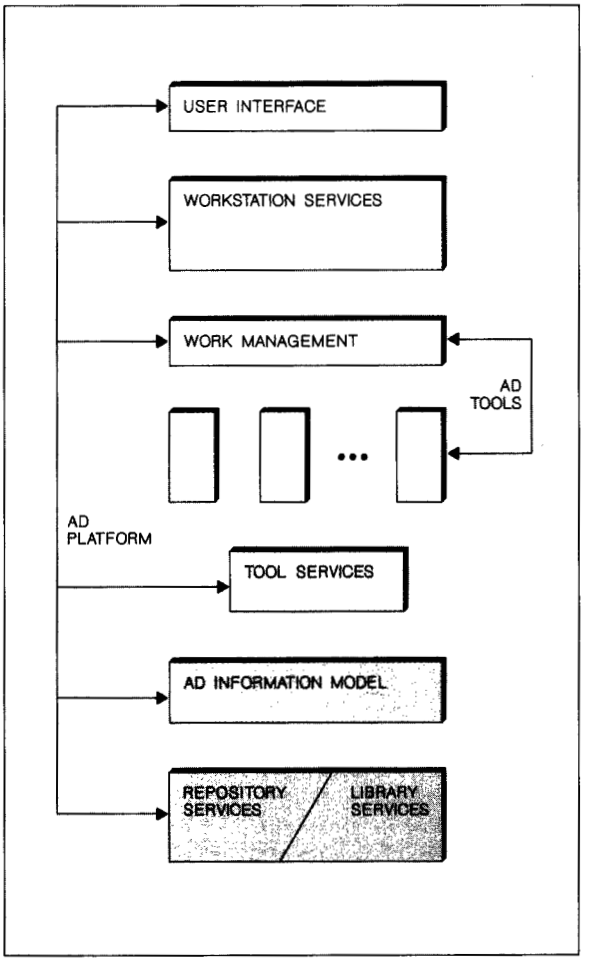
Support for program testing and debugging is aided by IBM's INSPECT for C/370 and PL/I. INSPECT is a single product that allows application developers to control the execution of C and PL/I programs, inspecting and modifying variables as they execute. INSPECT runs in either an interactive mode with multiple windows and extensive on-line help or it will operate in a batch mode. Over time, it is planned that program debugging will be extended to support all of the SAA languages from a single source-level debugging tool.

IBM also provides the Software Analysis Test Tool for test coverage analysis of PL/I and COBOL programs, and the Workstation Interactive Test Tool for regression testing support of interactive applications in SAA environments.

Cross life-cycle tools. Whereas some application development tools provide functional support for a single phase of the application development life cycle, management of projects, the process itself, documentation, and changes (impact analysis) extend across all phases.

Process management (discussed in Reference 19) is one of the key functions provided in AD/Cycle. This function, implemented as an AD/Cycle tool itself, is based on the IBM product, Application Development Project Support. The product enables the user to define a model of the application development process, to save that model, and then to guide the process by execution of the activities defined in the model. Control over the application development process is expected to improve the quality, as well as decrease the time required for development.

Figure 4 AD/Cycle architecture



The direction of AD/Cycle is to also provide the ability to identify and locate components across the life cycle (for example, enterprise models, designs, programs, etc.) that can or must be integrated in new application definitions.

AD/Cycle architecture

The AD/Cycle architecture has been defined as an infrastructure of services and interfaces to enable integration of application development tools across the life cycle. Figure 4 illustrates the layered archi-

tecture of AD/Cycle. The technical objectives, requirements, and constraints that influenced this architecture are discussed first. Following this discussion, each of the architectural components is described.

Having consistency and integration at the end-user interface is a major factor in the structure of the AD/Cycle architecture.

AD/Cycle technical objectives. Meeting the objective of accelerating and improving the delivery of application systems requires not only a full complement of AD tools that apply existing and emerging development techniques and methodologies to activities across the development life cycle; it also requires a supporting architecture that delivers and fully utilizes both well-established and evolving technologies and facilities within the operating systems and their application-enabling components. The technical objectives that influenced the design of the AD/Cycle architecture are now discussed. Many of these objectives and architectural fundamentals are identified in Reference 9, which is a description of the internal IBM application development architecture that evolved into AD/Cycle.

A major objective for AD/Cycle is to provide the user or tool developer with the capability to incrementally extend functional capabilities and to allow selective substitution of existing functions. This objective dictated that the architecture would clearly define interfaces to be published so that others could become part of our solution. The Repository Interface provides the functional interface to the Repository Manager, and the AD information model provides the model of shared data so that an AD tool conforming to the architecture can work in concert with other conforming AD tools.

The architecture must define the central control facility for enterprise administration. The scope of such a facility is necessarily larger than just application development. The Repository Services, which uses the recovery, integrity, and data-sharing capabilities of an underlying relational database management system (DBMS), and the Library Services, which controls storage of multiple versions of flat file data, provide this facility for data used by AD/Cycle tools.

A set of generic and reusable AD services that will be used by AD/Cycle tools and developers is required in order to ensure consistency in AD tool implementation and semantics, and to avoid redundant programming. The AD tool services and AD workstation services provide this capability.

The objective of having consistency and integration at the end-user interface is a major factor in the structure of the AD/Cycle architecture. It is necessary to give a common "look and feel" to the various AD tools. Consistency is especially important because AD tools from various sources will become part of the solution. Initially, AD/Cycle user interface consistency is based on the use of Presentation Manager™ of Operating System/2™ Extended Edition (OS/2® EE)—the operating system on a workstation—and the graphical model guidelines of the Common User Access (CUA).[20] This is the base from which the user interface will evolve into a common workplace model. This user interface component will be extendable so as to make tools accessible in a consistent way by using *work management*.[19] A significant prototyping activity developed the user interface[21] and ensured that the interface would be well-designed from the human factors point of view.

Technical requirements and constraints. In addition to the major objectives discussed above, several other technical requirements and constraints affected the design of the AD/Cycle architecture.

Cooperative SAA development environment. A requirement of the architecture of AD/Cycle was to merge the advanced graphical presentation and interactive characteristics of the intelligent workstation with the ability of mainframe facilities that centrally control and share application development resources. The Personal System/2® (PS/2®), with its supporting software in OS/2 EE, provides the application developer with an interactive, graphical user interface and provides AD tools with a multiwindowed, concurrent activity display needed to integrate multiple AD tool functions at the end-user level. The Repository Services and the Library Services will provide the ability to centrally control, admin-

ister, share, and recover application development data across the enterprise. Additionally, merging the workstation and host capabilities permits optimizing the activities of an individual developer (such as editing a single source program) and the activities of an administrator (such as building a complete application system comprised of numerous parts).

To support a cooperative environment between the PS/2 workstation and the Application System/400® (AS/400™) or System/370 hosts, the AD/Cycle architecture permits optimum placement of data and function between the processors. In other words, the architecture has the ability to route function requests to data location, bring data to the function location, and route single data requests from the workstation to the host. Additionally, these facilities are designed so that the location of function and data, on workstation or host, will be transparent to the end user.

It was required that the AD/Cycle architecture specify the application development solution for SAA. This requirement led to a cooperative structure with a PS/2 workstation running OS/2 EE as a front end and an SAA host—either the Multiple Virtual Storage/Time Sharing Option-Extensions (MVS/TSO-E) or Virtual Machine/Conversational Monitor System (VM/CMS) operating system on the System/370 hardware architecture, or Operating System/400™ (OS/400™) on the AS/400 hardware architecture—as a back end.

Multiple data storage facilities. To support shared, centrally controlled data, the AD/Cycle architecture was required to support application development data, within a single framework, with a wide range of storage requirements. Data captured at the front end of the development cycle, such as enterprise or design models, can be most naturally structured into entities and relationships between the data elements. Much of the data at the back end of the development cycle, such as source programs, are typically stored in flat file forms. The AD/Cycle architecture supports both storage forms with control information and relationships between the data being maintained with the entity-relationship model.

Data integrity and library management functions. The AD/Cycle architecture was required to merge the data integrity and recovery requirements traditionally supported by database management systems with the version, configuration, and build facilities traditionally offered by library control systems. The AD/Cycle data handling architecture addresses these requirements with a mixture of protocols that will be provided by the underlying relational database managers, the Repository Services, and the Library Services. Over time, it is planned to have enhancements defined and delivered to improve the level of consistency among protocols regardless of the storage form of the data.

Software technology. The AD/Cycle architecture was itself required to use, as well as allow AD tools to fully utilize, currently existing software technology so as to bring it to bear on the problem of improving both productivity and quality within application development. The existing technologies thus supported include advances in graphical displays, interactive techniques, connectivity and networking capabilities, database management and control facilities, configuration and library control functions, and mainframe capacity and performance. In addition to these technologies, it also implements other important technologies such as the entity-relationship (ER) model[22] and the object-oriented paradigm for persistent data.

Each component of the AD/Cycle architecture plays a role in enabling tool integration.

Technology extensions. Finally, the AD/Cycle architecture was required to provide an infrastructure that is extendable and able to accommodate advances in technology within the AD/Cycle structure itself, within the underlying SAA enterprise system, and within the AD tools and methodologies that operate within this environment. AD/Cycle architecture is required, for example, to be extendable to a local area network (LAN) server and fully distributed configurations, incorporate artificial intelligence technology, and enable the use of extensions to SAA.

Components of the AD/Cycle architecture

Each component of the AD/Cycle architecture plays a role in enabling tool integration. The totality of

the base consisting of the user interface, workstation services, AD information model, tool services, and Repository Services is referred to as the *AD platform.* AD tools, which supply specific AD function, are built using the services of this platform. The following discussion parallels Figure 4 from the top to the bottom, giving a descriptive overview of the AD platform and AD tools and relating their roles in the AD/Cycle architecture. More detailed descriptions of some of these components are found in other papers in this issue to which the reader is referred.

User interface. The AD/Cycle user interface implements the CUA rules and guidelines and will evolve to the workplace model[20] as the consistent point of display integration for all AD tools and support functions needed by an application developer using the workstation. CUA, complemented by work management facilities discussed below, will result in these independently developed AD tools presenting a reasonably consistent interface to application developers.

More information on the use of CUA in AD/Cycle can be found in Reference 21.

Workstation services. Code running on the workstation requires the availability of services for displaying information on the screen and for providing access to application development data. The workstation services component uses the SAA Presentation Manager[23] for screen display and a set of user interface services[21] that enhances tool builder productivity in building displays and consistent display formats. Workstation services also provides OS/2 services that allow Repository Manager data and library data to be accessed by workstation-resident AD tools. They materialize the data on the workstation either as a flat file or in entity-relationship structures. Workstation services also allows routing function invocation to a host function and direct Repository Manager access by workstation-resident AD tools.

Work management. Work management is the facility in the AD/Cycle architecture that provides for consistency of AD tool invocation for all AD/Cycle tools. This facility is a common one that allows the user to list possible activities and to select one on which to work. This consistent task invocation capability includes how lists are displayed, selected from, manipulated, updated, and refreshed.

In a given enterprise, the methodology preferred to develop applications may be encoded using a process manager. Such a methodology will embrace a preferred set of AD tools for application development and a specific sequence of process steps. The process management facilities of work management can be used to encode the methodology to accomplish AD tool initiation automatically. The use of process management within application development is given in Reference 9.

Architecturally, work management is classified within AD tools. It is in the tool category known as cross life-cycle tools, since it is used across the entire AD life cycle. However, it has a special architectural role since it controls the invocation of the other tools and has thus been singled out here.

Repository Services provides centralized, shared management of all application development data.

More information about work management can be found in Reference 19.

AD tools. On the AD platform base, IBM and IBM's Business Partners are developing AD tools that either are applicable across all of the development tasks or are specific to a single activity. In addition, IBM encourages vendors to enable their AD/Cycle tools to operate with this architecture to complement the IBM offerings. AD tool function can be delivered as a combination of OS/2 programs, objects, Repository Manager functions, and host programs. These tools, and the categories into which they have been classified in AD/Cycle, were discussed above.

Tool services. Tool services is common function that multiple tools require regardless of whether they execute on the host, the workstation, or cooperatively. Examples are profile and error message services. Tool services can be implemented as a combination of OS/2 programs, objects, Repository Manager functions, and host programs. Definition and delivery will be staged over time.

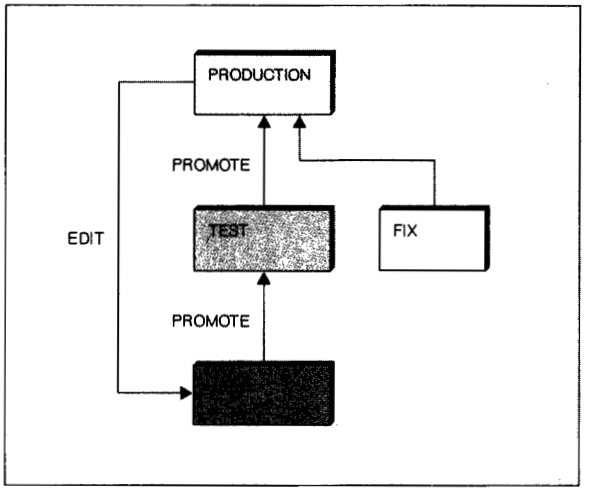
Repository Services and Library Services. The Repository Services component provides centralized, shared management of all application development data. It is the point-of-control for defining, administering, and controlling access to AD data shared by users through multiple workstations. It is built on top of the SAA relational DBMS, which provides facilities for safe, shared access to all of the data that define applications in the enterprise. It extends the SAA Common Programming Interface (CPI) by means of the SAA Repository Interface. On MVS, Repository Manager/MVS™ is the product that implements Repository Services.

Repository Services presents an entity-relationship (ER) model of these data. The ER services allows the specification of relationships among these definitional data, constraints on the data and on the relationships, and views that different programs may have of these data.

For example, an entity called PROGRAM and another called PROGRAMMER can be defined, with a set of attributes on each. A relationship called OWNS between PROGRAMMER and PROGRAM (and an inverse relationship called OWNED_BY) can be defined. It can be specified that a PROGRAM may be owned by only one PROGRAMMER, but that it must always have an owner. (So, if an attempt is made to delete a PROGRAMMER, it is prevented if either he or she owns any PROGRAMS or these PROGRAMS are deleted.) It can be specified that if an attempt is made to update the SALARY attribute of PROGRAMMER so that it exceeds a specified amount, a message is to be sent, and an authority check is to be made. A view available to a set of AD tools that does not include the SALARY attribute can be defined.

Some uses of the Repository Services require manipulation of many entities and relationships. Because of this, a need for a higher-level interface to AD/Cycle data has been identified. To address these problems, the Repository Services has provided object capability whereby a set of programs, called *methods*, can be registered in the Repository Manager to present a high-level abstraction of the data to the AD tools. If objects are used, the AD tool code need not see the data model directly, but rather see a set of object types and the method programming interfaces for these types. Note that objects can be used by tool services as well as by individual tools that want the benefits of the higher level of abstraction provided by objects.

Figure 5 Example of a library hierarchy



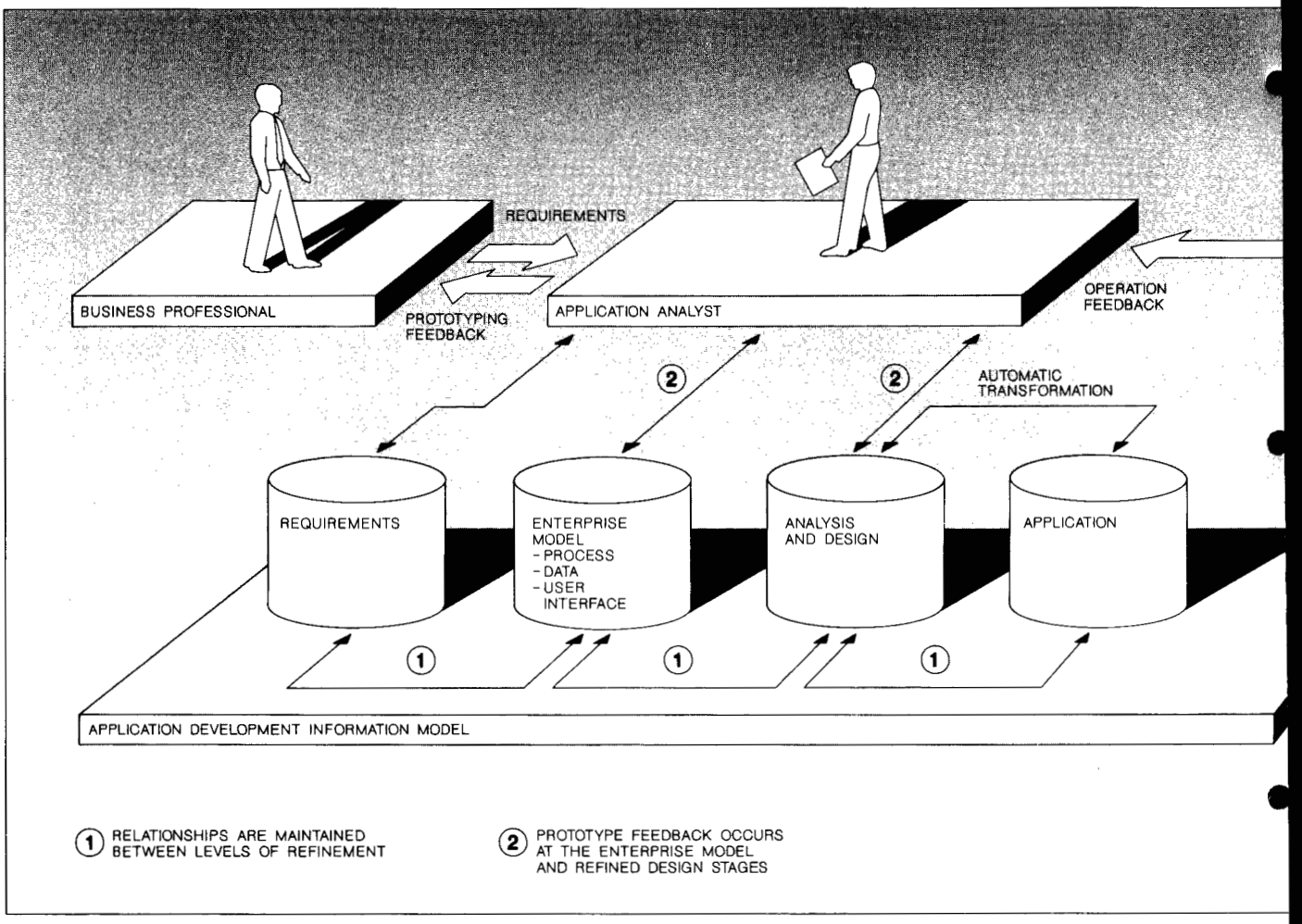
More information about the Repository Services component is found in Reference 24.

AD/Cycle has populated the Repository Manager with a set of entities, relationships, and constraint definitions that represent the data that may be shared between AD tools at various stages in the development life cycle. A standard model specifying, for example, the representation of COBOL data structures and Information Management System (IMS) database definitions necessary in building an application has been created. This model of AD/Cycle data that may be shared is the AD information model. As enterprise modeling, design, generator, and editor AD tools populate the Repository Manager with these elements, the tool data from each tool will be integrated with data from other tools.

It is planned to have the interfaces to the Repository Services available to AD tools whether they execute on the host or on an attached workstation.

Library Services provides support for versioning and automatic application build. It allows an installation to set up a project database in which data reside in flat file data sets. The project database is organized into groups, each group being subordinate to the one above it. This form of data organization is known as a *hierarchy*. Hierarchies are allocated from bottom to top, each level being a different version of the application. Thus, when data is referenced, the lower positions in the hierarchy take precedence over members at a higher position. Figure 5 shows an

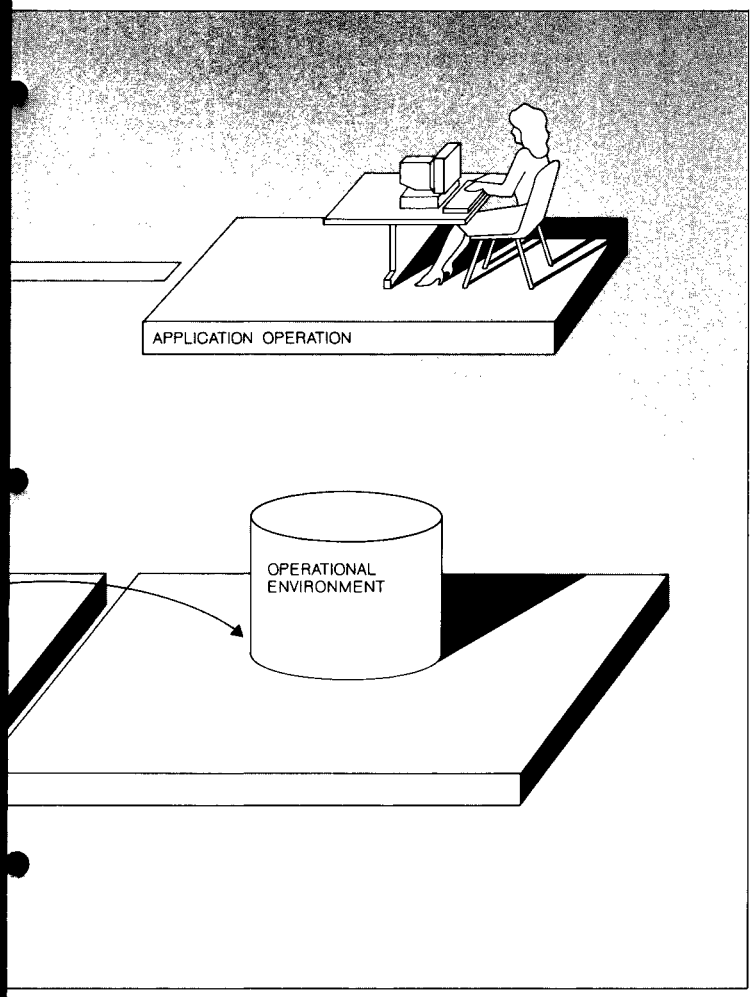
Figure 6 Application development under AD/Cycle



example of a library hierarchy, with the root PRODUCTION. When a user wants to edit a file that is not currently at the USER level of the hierarchy, the file is brought down to the USER level and locked at the PRODUCTION level. When the USER level has completed its changes, it is moved back up the tree, one level at a time (e.g., USER to TEST, TEST to PRODUCTION). The upward process is called *promote*. Library functions allow the user to browse, create, update, delete, compile, link, build, promote up the hierarchy, and report on data stored in the database of a project. The build function, sometimes called *configuration management*, ensures project integrity by verifying that all components are present and complete. It keeps track of data set interdependences and will automatically perform necessary compilations and links so that out-of-date components are replaced with up-to-date ones.

On MVS, Library Services is implemented by the Software Configuration and Library Manager (SCLM),[25] which is a component of the Interactive System Productivity Facility/Program Development Facility (ISPF/PDF) Version 3.

Over time, it is planned that AD/Cycle define and deliver additional support to enhance the integration of tool services, Repository Services, and Library Services with the goal of providing consistent versioning across all AD/Cycle data, maintaining relationships between modeled and file data, and ensuring a single point of control for all operations.



The AD information model: the data architecture of AD/Cycle. One key to both opening the AD/Cycle architecture to other vendors' and customers' functions and providing an integrated solution is delivering and publishing consistent data specifications used across the life cycle. These data specifications can capture the structure of data used by applications and the application specification itself. As discussed above, the AD/Cycle architecture defines the data specifications that will be populated and used by AD/Cycle tools throughout the development of applications. It contains definitions of the entities, attributes, and relationships of the Repository Manager data.

The data architecture is also characterized by three points of data integration:

1. An enterprise model, which represents the point of data integration after an enterprise-modeling AD tool has completed its task. A good industry consensus on this model has been obtained, and its implementation and use are planned by various enterprise-modeling tools from IBM and its Business Partners.
2. A design model, which represents the point of data integration after a design AD tool has completed its task. Initially, there is no industry agreement on a common design model; so, although it is part of the architecture, it will not be standardized in early versions of AD/Cycle.
3. A technology model, which represents the point of data integration after a "produce" AD tool has completed execution. ("Produce" AD tools consist of such things as compiler, generator, and knowledge-based system AD tools, i.e., AD tools that "produce" user application code.) The technology model reflects the types of tools that use it, and thus it is expected to grow over time as new tools are added.

It is planned to have these points of integration met by all applicable AD/Cycle tools in an AD/Cycle system, thus allowing data to be shared among AD tools from many different sources, both within and outside of IBM. They are used regardless of the methodology being followed or the style of user interaction being offered.

More information about the AD data architecture can be found in Reference 26.

AD/Cycle complements and extends SAA

AD/Cycle is based on IBM's Systems Application Architecture and will evolve using the same elements. The functions provided by AD/Cycle will be SAA-conforming applications and will support the range of SAA environments. An example is the incorporation into AD/Cycle of the graphical model of the SAA Common User Access support[19] provided by the OS/2 Extended Edition Presentation Manager for use by the AD/Cycle application development tools and tool services.

With AD/Cycle based on SAA, the skills learned in one implementation of AD/Cycle will be transferable to the other SAA environments. Training costs and migration costs to move to a new application development environment, to add tools to an existing one, or to change methodologies will be substantially reduced.

An example of SAA extension introduced by AD/Cycle is the programming interface to Repository Manager, defined as a new element of the SAA Common Programming Interface (CPI). This component of the SAA CPI will be important to AD/Cycle by providing for controlled communications and sharing of data between the tools.

AD/Cycle tools conform to Common User Access guidelines.

AD/Cycle utilizes an evolutionary strategy. It is designed to build upon existing IBM and vendor products in a staged manner, allowing users to develop their own timetable for moving into new application development technologies and methodologies. The AD/Cycle implementation is based on enhancements to existing application development tools and on the introduction of new tools by IBM, customers, and software vendors. Existing capabilities will be enhanced and new functions will be added in stages in the different SAA environments. This evolution will result in consistent, integrated application development tools and services across the SAA environments.

Application development under AD/Cycle

The key difference in application development as it emerges under AD/Cycle (portrayed in Figure 6), as opposed to application development today (see Figure 2), is in the following four areas unique to AD/Cycle:

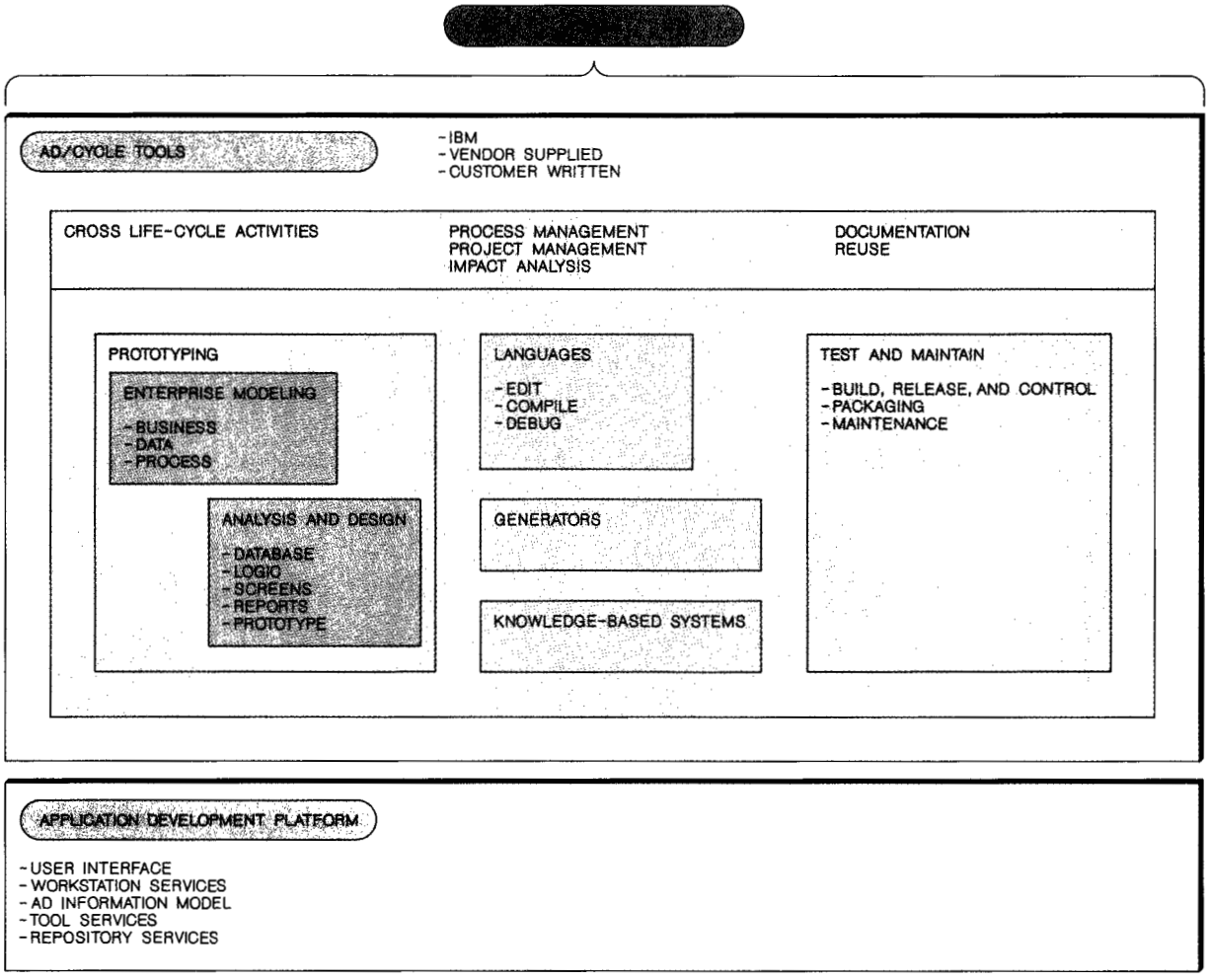
1. AD tools share data through common use of the AD information model.
2. Tool-to-tool user transition will be simplified as a result of having AD/Cycle tools conform to the CUA guidelines and the standards associated with a tool developer's guide to be published for application development.
3. Reduction in the necessity for unique skills is achieved in the application development process by enabling application development through refinements of the enterprise model. The analysis and refinement tools can be invoked by the same individual who creates the application model, and this individual is assisted by automated transformation operations that create the executable application from the refined application design. These transformations occur by using tools such as application generators or compilers.
4. The open architecture of the AD platform provides a mechanism through services and standard definitions to enable IBM, customers, and tool vendors to add and extend the capability of the application development process. For example, new technology emerging in user interface facilities can be easily enhanced through ongoing tool additions to this foundation.

Figure 7 shows the AD/Cycle application development function model that provides the foundation for an integrated and efficient application development process. At the top of the figure, SAA is shown as the umbrella under which AD/Cycle serves as an SAA application and the IBM SAA solution for application development. The AD/Cycle tools encompassing the application development life-cycle model will provide support for the entire application development life cycle. They can be provided by IBM, customer-written, or vendor-supplied because of the openness of AD/Cycle established by the AD information model. Supporting the life-cycle tools at the bottom of the figure, the AD platform provides access to the Repository Manager for application development data, an interface to the Repository Manager for the access and manipulation of the data, and an information model that defines the structure and format of the data. The platform also provides a CUA-conforming user interface and a tool invocation and execution environment located at the workstation. This function model defines the key tool areas required for full life-cycle support of application development. It also includes the key AD platform services that provide the base for all tools and ensures their integration and sharing of appropriate application development data, processes, and parts.

Summary

The application development process under AD/Cycle is markedly different from current application development processes. Tool integration, both data and functional, along with the relationships established between parts of applications and the successive stages of application refinement, combine

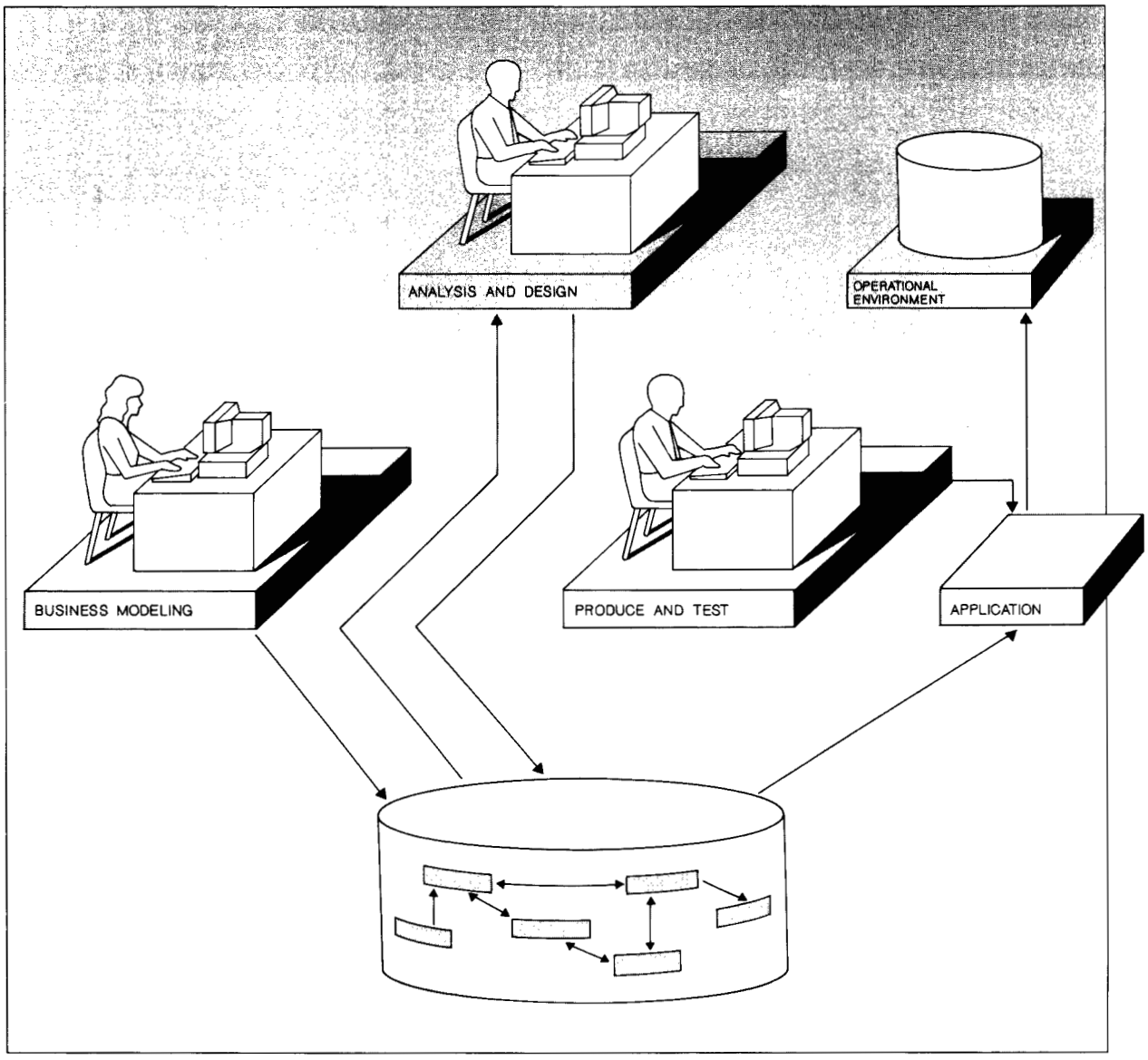
Figure 7 AD/Cycle life-cycle function model



to enhance and simplify application construction and control. Figure 8 portrays this simplified application development environment. (Contrast Figure 8 with Figure 2, showing the current application development process as an inefficient people- and paper-intensive process characterized by manual intervention and paper transfers of information from step to step.) Integration and efficiency are supported by AD/Cycle, which provides multiple methodologies and enables tools to share data throughout the life cycle, and by the centralized Repository Services for the control and management of application development data.

Although the AD/Cycle tools strategy and architecture has focused on tool integration and tool technology, the overall net effect is a strategy that will eliminate the application development inhibitors to customer data processing and overall customer growth. Tool builders are able to focus more on tool specifics without having to be concerned about implementing underlying data facilities. Thus they can provide a wealth of high-quality, integrated tools for tool users. Tool users in turn are able to move easily from tool to tool and are assured of a high degree of integration and data sharing. Application data can be defined and entered once and can be easily shared and reused at different steps along the development process. Applications can now be defined at the source of the requirements to improve responsiveness to end-user requirements; they can be evaluated earlier, reducing costly and time-consuming rework

Figure 8 Application development is simplified under AD/Cycle



at later stages. As information concerning the underlying processing and data requirements of an application are stored in the Repository Manager, applications are able to grow as customers' businesses grow.

Acknowledgments

The AD/Cycle strategy and architecture summarized by this paper represent the joint work of a team of people throughout IBM. The authors would like to acknowledge the work done by Dave Harvey and his staff in Cary who worked with the Santa Teresa and Toronto Programming Systems development laboratories in the formulation of the AD/Cycle tools strategy. We would also like to acknowledge the work on the AD/Cycle architecture done in Santa Teresa in Bob Costain's and Steve Uhlir's departments working with the Cary and Toronto development laboratories, with internal tools efforts, and with

IBM's Business Partners. We also wish to acknowledge the leadership of Jim Archer, the IBM director of integrated development environments in Santa Teresa, for his overall AD/Cycle product and strategy coordination.

Cited references and notes

1. *Application Development Productivity Strategy*, worldwide IBM user group made up of Australasian SHARE/GUIDE, GUIDE International (U.S.A.), G.U.I.D.E. (Europe), SHARE European Association (SEAS), and SHARE (U.S.A.) Application Development Joint Project (ADJP) (May 1989).
2. *Application Development Productivity, A GUIDE Top Concern*, GUIDE (December 1986).
3. *MP-1409 Integration of Development Tools Project Strategy Paper*, GUIDE, Montreal, Canada (November 1986).
4. *SHARE Interactive Systems Task Force Report*, SSD #223, SHARE (1983).
5. CASE87, a survey conducted in August 1987 and published by Software News' Sentry Research Division.
6. J. Martin, *Application Development Without Programmers*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1982).
7. W. S. Humphrey, "The IBM Large-Systems Software Development Process: Objectives and Direction," *IBM Systems Journal* **24,** No. 2, 76–78 (1985).
8. R. A. Radice, J. T. Harding, P. E. Munnis, and R. W. Phillips, "A Programming Process Study," *IBM Systems Journal* **24,** No. 2, 91–101 (1985).
9. G. F. Hoffnagle and W. E. Beregi, "Automating the Software Development Process," *IBM Systems Journal* **24,** No. 2, 102–120 (1985).
10. "Programming Aids," Section D80, *Datapro Directory of Software* **15,** No. 11, D80-000-001–D80-900-005, Datapro Research, McGraw-Hill Information Services Company, Delran, NJ (November 1989).
11. *ICP Software Directory, Mainframe & Minicomputer Series—Systems and Utilities*, 62nd Edition, International Computer Programs, ICP Incorporated (Autumn 1989).
12. T. Teitelbaum and T. Reps, "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment," *Communications of the ACM* **24,** No. 9, 563–573 (1981).
13. D. E. McConnell, *An Investigation of the State of the Art Trends in the Life Support of Complex Embedded Computer Systems*, Naval Weapons Center, Dahlgren, VA (September 1979).
14. *AD/Cycle: Application Development for the SAA Environments*, Programming Announcement 289-456, IBM US Marketing & Services (19 September 1989); available through IBM branch offices.
15. *Systems Application Architecture: An Overview*, GC26-4341, IBM Corporation; available through IBM branch offices.
16. K. P. Hein, "DevelopMate: A New Paradigm for Information System Enabling," *IBM Systems Journal* **29,** No. 2, 250–264 (1990, this issue).
17. M. E. Dewell, "Cross System Product Application Generator: Application Design," *IBM Systems Journal* **29,** No. 2, 265–273 (1990, this issue).
18. D. M. Hembry, "Knowledge-Based Systems in the AD/Cycle Environment," *IBM Systems Journal* **29,** No. 2, 274–286 (1990, this issue).
19. G. Chroust, H. Goldmann, and O. Gschwandtner, "The Role of Work Management in Application Development," *IBM Systems Journal* **29,** No. 2, 189–208 (1990, this issue).
20. *Systems Application Architecture, Common User Access—Advanced Interface Design Guide*, SC26-4582, IBM Corporation; available through IBM branch offices.
21. J. M. Artim, J. M. Hary, and F. J. Spickhoff, "User Interface Services in AD/Cycle," *IBM Systems Journal* **29,** No. 2, 236–249 (1990, this issue).
22. P. P. S. Chen, "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Transactions on Database Systems* **1,** No. 1, 9–36 (March 1976).
23. *Systems Application Architecture, Common Programming Interface Presentation Reference*, SC26-4359, IBM Corporation; available through IBM branch offices.
24. J. M. Sagawa, "Repository Manager Technology," *IBM Systems Journal* **29,** No. 2, 209–227 (1990, this issue).
25. *Interactive System Productivity Facility/Program Development Facility (ISPF/PDF) Software Configuration and Library Manager (SCLM) Guide and Reference, Version 3 for MVS*, SC34-4235, IBM Corporation; available through IBM branch offices.
26. R. W. Matthews and W. C. McGee, "Data Modeling for Software Development," *IBM Systems Journal* **29,** No. 2, 228–235 (1990, this issue).

General references

D. R. Barstow, H. E. Shrobe, and E. Sandewall, Editors, *Interactive Programming Environments*, McGraw-Hill Book Co., Inc., New York (1984).

Common APSE Interface Set (CAIS), Proposed Military Standard, Version 1.3, Report AD-A134825/9, Office of the Secretary of Defense, Ada Joint Program Office, Washington, DC (August 1984).

C. Davis, et al., Editors, *Entity-Relationship Approach to Software Engineering*, North Holland Publishers, Amsterdam (1983).

Proceedings of a Conference on Language Issues in Programming Environments, Seattle, WA, June 1985, *ACM SIGPLAN Notices* **20,** No. 7 (July 1985).

Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Pittsburgh, PA, April 1984, *ACM SIGPLAN Notices* **19,** No. 3 (May 1984).

Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Palo Alto, CA, December 1986, *ACM SIGPLAN Notices* **22,** No. 1 (January 1987).

Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, MA, November 1988, *ACM SIGPLAN Notices* **24,** No. 2 (February 1989) and *SIGSOFT Software Engineering Notes* **13,** No. 5 (November 1988).

Systems Application Architecture: AD/Cycle Concepts, GC26-4531, IBM Corporation; available through IBM branch offices.

Vincent J. Mercurio *IBM Programming Systems, P.O. Box 60000, Cary, North Carolina 27512-9968.* Mr. Mercurio is currently an advisory planner in the AD/Cycle systems planning and architecture department in the application development products business area at the Programming Systems Development Laboratory, Cary. Since joining IBM in 1970, he has been involved in the development of Multiple Virtual Storage in Poughkeepsie, New York, store systems and point-of-sale control programs in Raleigh, North Carolina, and the Realtime Programming System and Event Driven Executive Series/1 operating systems in Boca Raton, Florida. In addition to his involvement in systems development, Mr. Mercurio has also worked in telecommunications market analysis and in product assurance as the product administrator for application development tools. Mr. Mercurio obtained a B.S. degree in mathematics from Ohio State University and an M.S. degree in operations research from Union College.

Barbara F. Meyers *IBM Programming Systems, Santa Teresa Laboratory, P.O. Box 49023, San Jose, California 95161-9023.* Dr. Meyers was educated at the University of California, Los Angeles, obtaining her Bachelors, Masters, and Ph.D. degrees in computer science in 1969, 1970, and 1975, respectively. She joined IBM in 1975 as a performance analyst and worked on the performance of disks and control units, databases, transaction management, and data management. In 1980, she started and managed a performance group for the programming language area at the Santa Teresa Laboratory of IBM's General Products Division. Dr. Meyers has held technical staff positions in the programming language and application development area in the Santa Teresa Laboratory, with responsibility for helping to determine strategy, plans, and advanced technology directions in these areas. In her current position as senior programmer, she is a member of the AD/Cycle architecture department.

Al M. Nisbet *IBM Programming Systems, P.O. Box 60000, Cary, North Carolina 27512-9968.* Mr. Nisbet is currently manager of application development products—systems planning and architecture at the Programming Systems Development Laboratory in Cary, involved in AD/Cycle plan and architecture. Mr. Nisbet joined IBM in 1964 in Chicago where he worked in program field support. He then moved to Raleigh to participate in research and development for many software support programs as programmer, planner, planning manager, and development manager. Some of the systems and products he supported include the Telecommunications Test Center, the Retain System, and the Branch Office Support System. He was development manager for IBM's Cross System Product and ISPF/PDF.

George Radin *IBM Programming Systems, Santa Teresa Laboratory, P.O. Box 49023, San Jose, California 95161-9023.* Mr. Radin, an IBM Fellow, has Masters degrees in both mathematics and English literature from Columbia University. He joined IBM in 1963. Mr. Radin has held several key technical positions throughout his career including: leader of the PL/I language definition, member of the design team of both the OS/360 and TSS operating systems, systems design and architecture manager in the former Systems Development Division, member of the IBM Corporate Technical Committee, Director of Architecture in the System Products Division, and manager of several projects in IBM Research (including the 801 minicomputer project which defined IBM's RISC architecture). He is currently Chief Architect of the AD/Cycle project in Programming Systems at the Santa Teresa Laboratory.

Reprint Order No. G321-5392.