A new development rhythm for AS/400 software

by R. A. Sulack
R. J. Lindner
D. N. Dietz

Synchronizing the software development process with hardware development and user involvement programs yielded a product offering that met the user requirements with a significantly reduced development cycle. This paper emphasizes the key elements of Application System/400™ (AS/400™) software development that contributed to synchronization and project success. It is intended to produce an awareness of the elements that set this project apart from most others.

The development of the Application System/400™ (AS/400™) system was a challenge from the outset. A market analysis had shown the industry needed a competitive solution in mid-1988, but development directions for future products were proving infeasible. Customers, and the industry in general, were demanding more powerful and compatible systems to replace the IBM Rochester product line led by the System/36 and the System/38. The needed system had to be developed in only two years and had to exhibit a quality level equal to or better than that which customers were currently enjoying.

The system had to provide a growth path for current System/36 and System/38 users, which implied the ability to run current system applications on the new system hardware and software. This also implied a requirement to provide programmer productivity strengths equivalent to those of System/38, ease-of-use strengths like those in System/36, and advanced characteristics for future customer needs, all in a single operating system. The operating system envisioned had to satisfy the rapidly growing range of businesses needing midrange systems. It would be used in diverse environments and, to meet customer needs, it had to improve the consistency with IBM System/370 and personal computers.

These challenges required major changes to the development process. Previously, systems were developed by building hardware, delivering it to programming personnel for software development, and delivering the completed combination to early users for evaluation before general availability. It was quite clear that this course of events would not produce a product timed with the market requirement. Hardware and software development processes had to be synchronized to achieve maximum overlap. Also, user evaluation had to take place early in development to make sure the product would meet user expectations and to give the development community time to react if changes were required before general availability.

Early sizings determined that this would be the largest programming effort undertaken at IBM Rochester to date. Even with extensive reuse of software parts from previous systems, the new code required would be about one and one-half times the historical capacity of the programming team, given the time

available. Additionally, to meet ease-of-use objectives, a significant increase in the amount of on-line information would be required over previous systems. This information would have to be translated into 25 national language versions to be available for a concurrent worldwide general availability. The project would require coordinating the efforts of over 1500 developers at multiple locations.

Even with these challenges, all the key ingredients appeared to be in place for a successful project. A clear, high-level definition of requirements based on the current products existed. The management team had the latitude to shape the organization to match the project. The project was basically autonomous within the Rochester Development Laboratory, which was responsible for both hardware and software. The project had experienced programmers, adequate tools, proven development process techniques, and personnel who were highly motivated.

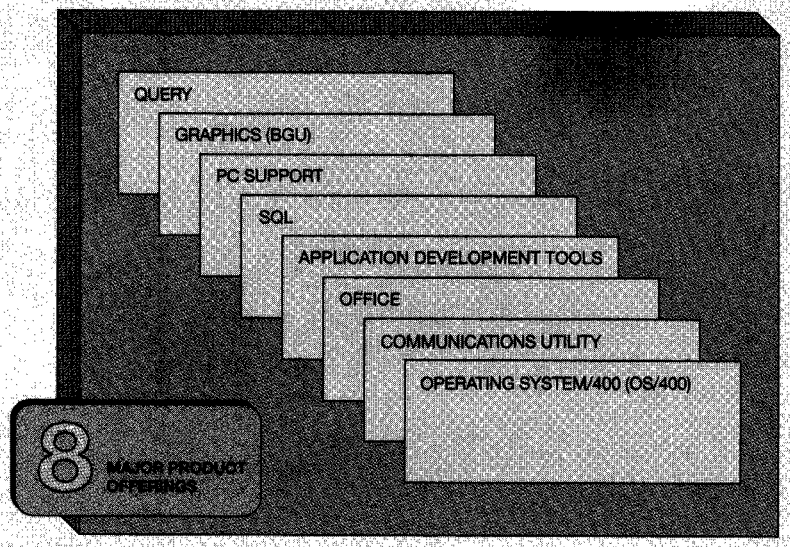
The process

Organizational considerations presented unique challenges to the software development process. System/36 and System/38 were being developed in separately managed organizations with their own tools and processes. Both organizations were committed to provide support and follow-on development activity that required continued resource allocation. The processes were quite similar in that they followed a model known as the IBM programming process architecture.[1] Both organizations conducted design reviews and code inspections, though they had different terminology and a certain amount of variation in the application of the process, especially in the early steps. In addition, another organization was working on future product development. These three organizations had to be united under a single process and tool set, all working on the AS/400 software. Each organization's process was reviewed to determine the following: (1) the common elements, (2) the most positive elements where differences existed, and (3) shortcomings that could be addressed by introducing new concepts. This resulted in a process definition that was somewhat familiar to all, thus encouraging confidence that it could be applied with minimal disruption. (See Figure 1.) A new operational process definition was documented in on-line data sets that could be accessed by all developers. This process documentation was approved by all user organizations through formal inspections and accepted by the development community as the basic method to be used for producing software. The tool set used was also a combination of the tools used in the previous organizations and was a collection of the best available tools, integrated under a common user interface. Moving to a common tool set and process

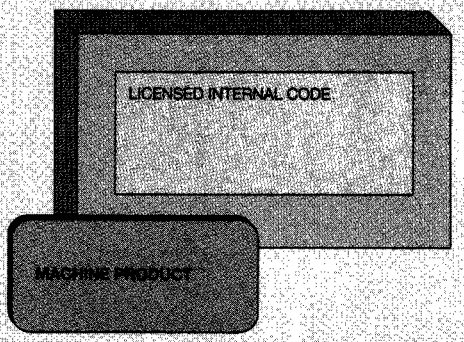
Figure 1 Process similarities

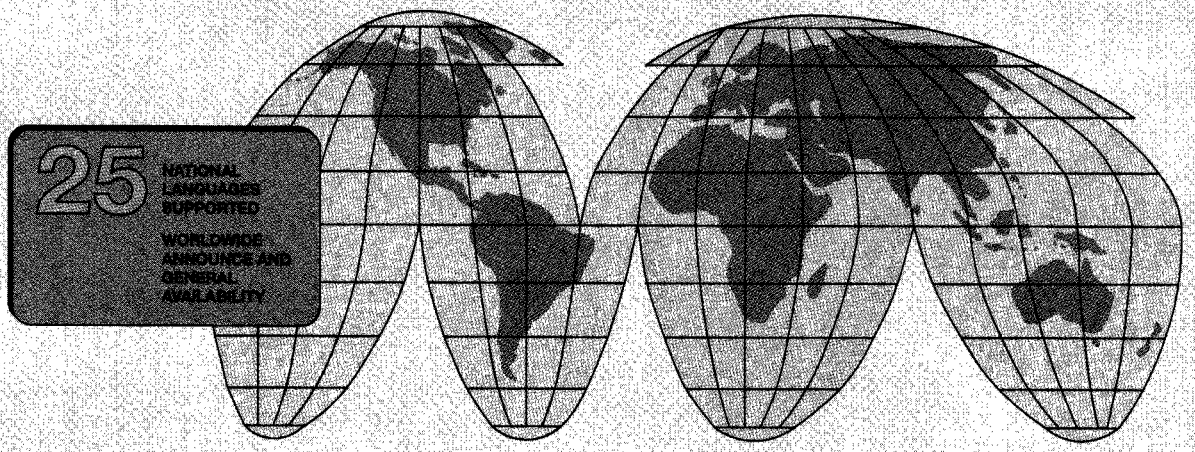
	VALIDATION			
	S/36	S/38	FUTURE PRODUCT DEVELOPMENT	AS/400
PRODUCT OBJECTIVES	REVIEW	REVIEW	REVIEW	WALKTHROUGH
ARCHITECTURE		WALKTHROUGH	STRUCTURE INSPECTION	SYSTEM/PRODUCT DESIGN INSPECTION
SPECIFICATION	REVIEW		EXTERNALS INSPECTION	
HIGH-LEVEL DESIGN	REVIEW	INSPECTION	INSPECTION	INSPECTION
INTERCOMPONENT INTERFACES	REVIEW		INSPECTION	
LOW-LEVEL DESIGN	INSPECTION	INSPECTION	INSPECTION	INSPECTION
CODE	INSPECTION	INSPECTION	INSPECTION	INSPECTION
TEST PLAN	INSPECTION	INSPECTION	INSPECTION	INSPECTION
TEST CASES	INSPECTION	INSPECTION	INSPECTION	INSPECTION

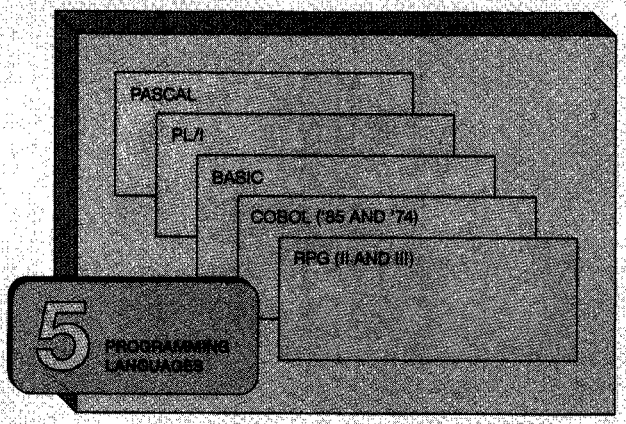
AS/400 Product Scope

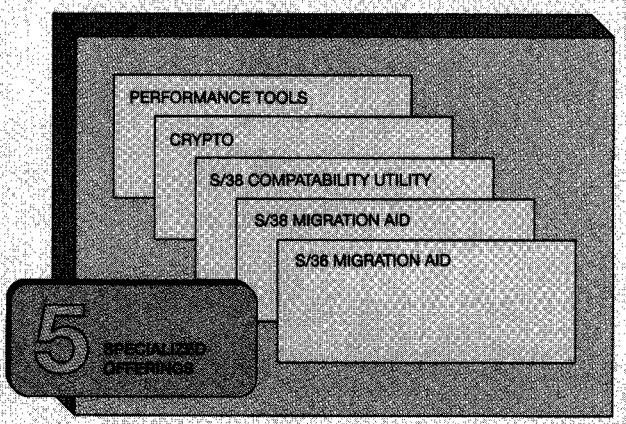


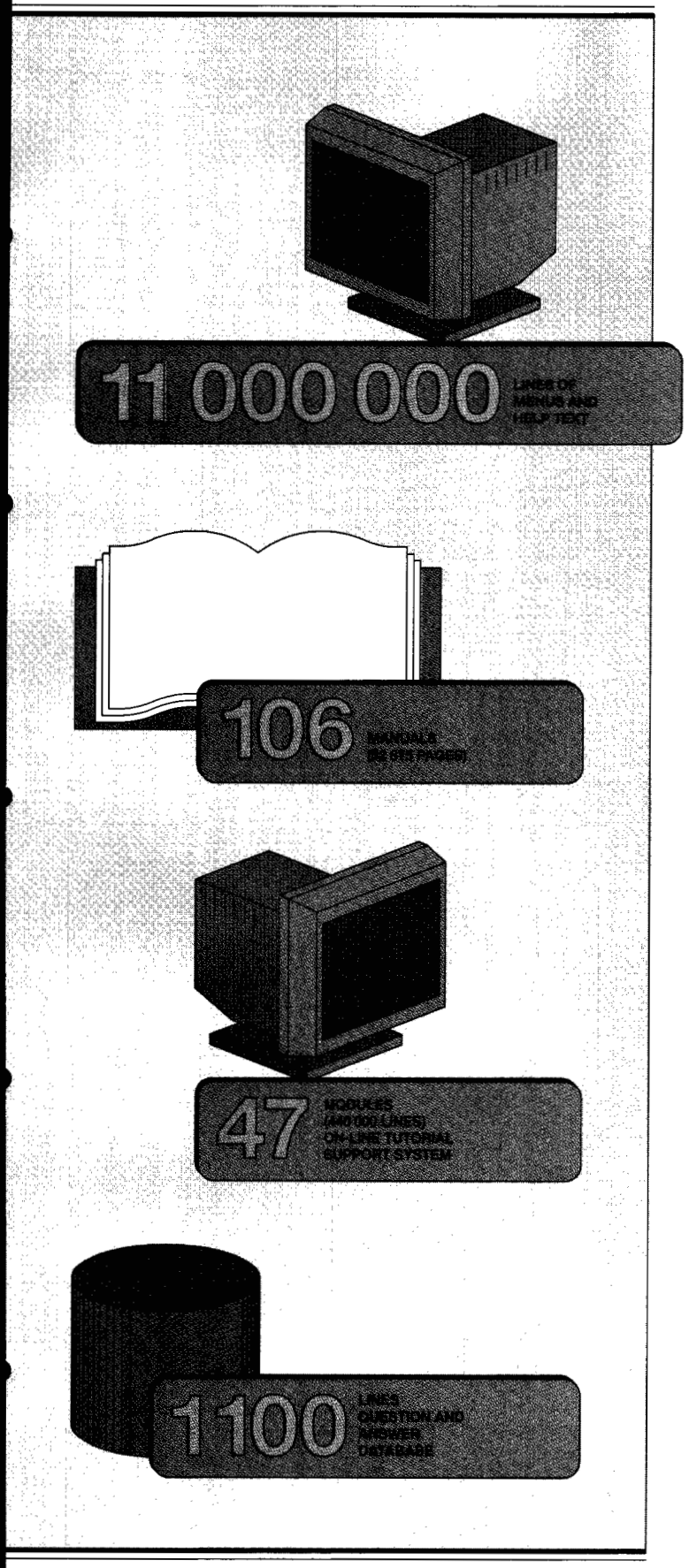
The AS/400 announcement presented a total system with a multitude of products to the market. A vast array of solutions were announced and made available concurrently worldwide, with text translated into twenty-five national language versions.











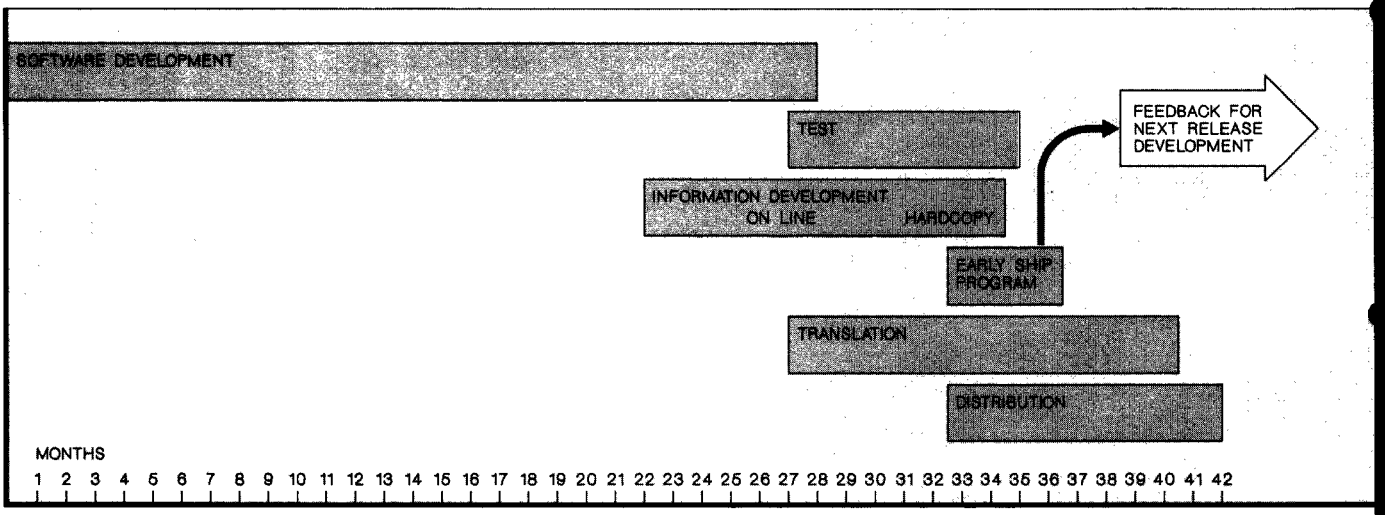
was essential to the project management control required by a project of this size and proved beneficial when the project reached the point of assembling the software parts into an operating system on a weekly basis.

The process steps. The process, as initially defined for AS/400 software development, was to begin with a walkthrough of system objectives, followed by the completion and inspection of a system design directions document that would describe the basic designs necessary to reach the objective. These activities were necessary to establish a common understanding of the end product throughout the development organization and to expose areas where further work was necessary to clarify requirements prior to implementation. Design and implementation could then begin, with the areas having firm definition proceeding under the technical direction of design control groups (DCGs), which are described in more detail later in this paper.

All new functions for Operating System/400™ (OS/400™) were introduced into the process through the DCGs. Component designs were documented along with the intercomponent interfaces and were validated by a high-level design inspection. Functions being reused from prior systems that required major development work were introduced into the process in the high-level design step where they could also be inspected to validate the redesign. All defects discovered by inspection were recorded with the point of origin identified. The resource required to repair the defect and the resource required to produce the inspection material and hold the inspection were also identified and recorded, so that analysis of the data would allow the cost of defect removal to be determined. In the low-level design step, the components were refined into implementation modules, with the code structure documented and inspected. Some modules being reused from System/38 with minor changes could enter the process in this step. Again, the inspection defects were recorded as before.

With the successful completion of the low-level design inspection, the module code could be produced. The code underwent the following three steps: (1) it was validated by inspection; (2) it was unit-tested by the developer; and (3) it was integrated into the base, which came from reusing major elements of the System/38 operating system without change. Once again, the inspection defects were recorded as before. In addition, the defects discovered during unit test were recorded to provide a complete picture of defect

Figure 2 Typical software process cycle time



injection and removal throughout the process for analysis of product quality.

Following integration, three major tests were run, all using formal test cases and test plans validated by inspections. The first test was done by the development teams as they assembled their modules into components. The last two tests were conducted by an independent test organization, which was to focus on testing the functions of multiple components assembled as program products and on testing the complete system from a user-oriented perspective. (These tests are discussed more in the product-verification section of this paper.) Defects removed by testing were also recorded with the same level of information that accompanied the defects found by inspection.

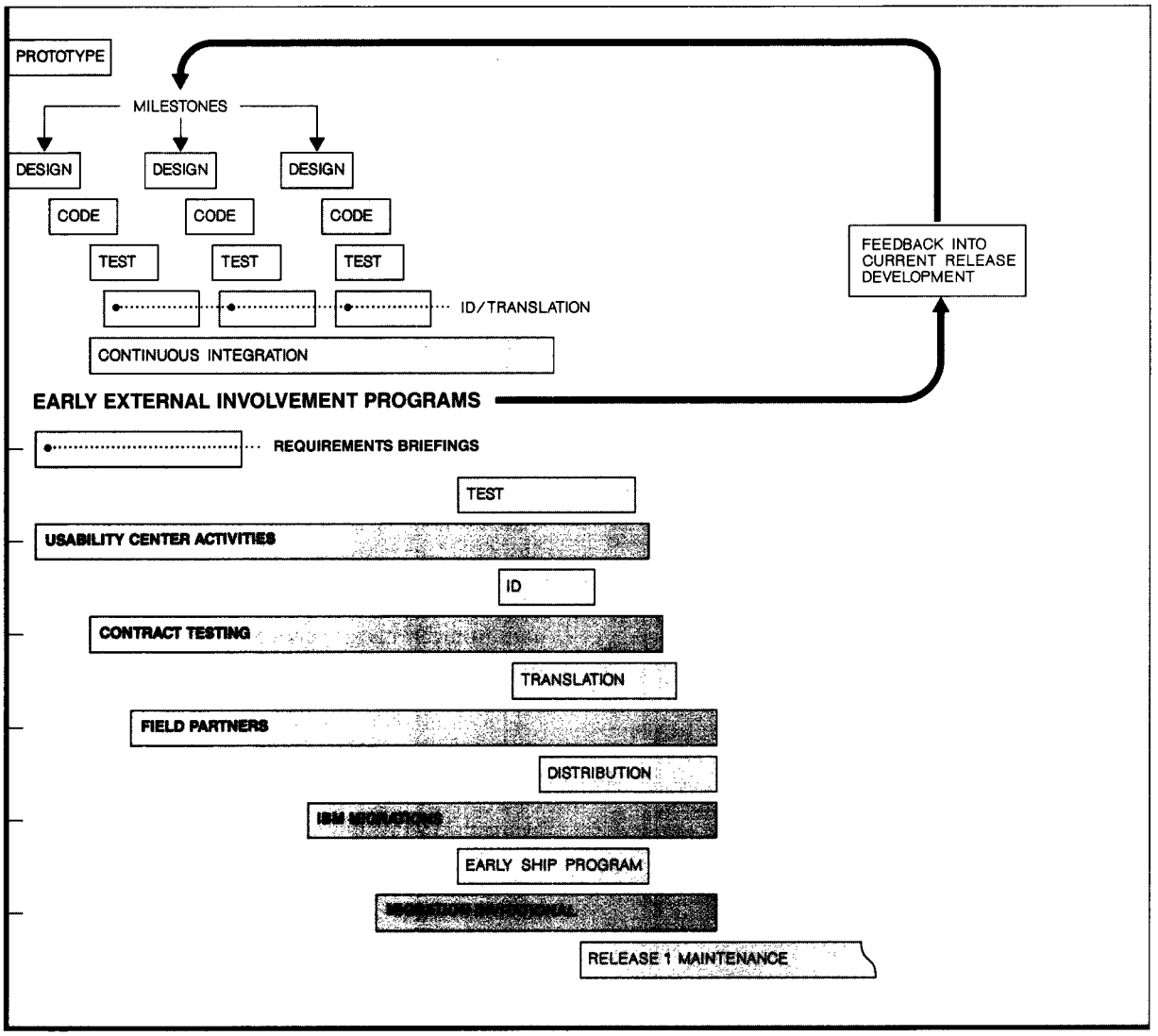
Early analysis of the software development task, given the current concepts and process flexibility just stated, produced a development cycle time that would result in the product's being delivered about one and one-half years too late. Figure 2 shows the projected cycle time, given the resource available and staffing curves, with the then current overlap of activities and productivity rates. A method of effectively practicing these proven techniques while reducing the overall cycle time had to be found to achieve product success.

The breakthrough. Even though the time-proven process for code production was well understood and accepted, changes to the overall concepts of software development were necessary to contain the required schedule. The most important change was centered around the recognition that there would be only one opportunity to provide the system that met the needs of the marketplace. Success demanded a clear knowledge of the user's needs. Thus a method had to be devised that would produce this knowledge and the method had to be timed with the development organization's ability to deliver the product. A breakthrough occurred when an iterative, or cyclic, process was integrated with the market analysis and verification programs.

The process consisted of developing the system by iterating and producing a deliverable functional milestone with each iteration. This would allow the analysis and verification of incremental functions by development support personnel, market support personnel, and end users in each cycle.

For example, the first milestone was defined to have the ability to run certain System/36 applications on the AS/400 system. This implied that major elements of the operating system had to be functional. With this milestone, a user interface prototype could be constructed for early usability evaluations. Test strategies could be defined and attempted, and information development work could begin. Translation activities could begin for the on-line information supplied with the functions of the milestone. (See Figure 3.) While these activities were proceeding with the

Figure 3 Iterative process



first milestone, additional system requirements could be stabilized for the next milestone, thus focusing resource on work activities least likely to change as further clarification of requirements proceeded. Each milestone could act as a prototype for the next, allowing more and more user involvement with development as function materialized with each successive iteration.

User involvement actually began with the definition of content for these milestones. Input from a Customer Advisory Council formed the key elements of the initial requirements. Systems engineers (SEs), who work in direct contact with end users, were briefed on the system requirements and asked for their input and assistance. Their first-hand knowledge provided additional insight into the requirements of the users, the types of applications the users would most likely want to move to the new system first, and customers or Business Partners who would be candidates for participation in future development activities. This activity served the key function of stabilizing the

requirements for first milestone development. With stable requirements for the first milestone, development could proceed at an accelerated pace.

With the nature of a system milestone being defined as a functional product or component rather than the often-produced scaffolded pieces of function, end users could be contracted to evaluate specific functions well before completion of the total system. This provided user feedback on the function being evaluated in time to incorporate changes before delivery and also stabilized additional requirements for later milestones. The system could be demonstrated to users who were interested in developing applications that could be available at the time of general availability. As later milestones became available, complete applications could be migrated and tested, thereby maximizing the application support available at the time of general availability. This supported an underlying goal for the system of providing immediate solutions for customer needs. The techniques employed to involve users in the development process are discussed in greater detail in the paper on early user involvement.[2]

The system could be demonstrated to users who were interested in developing applications that could be available at the time of general availability.

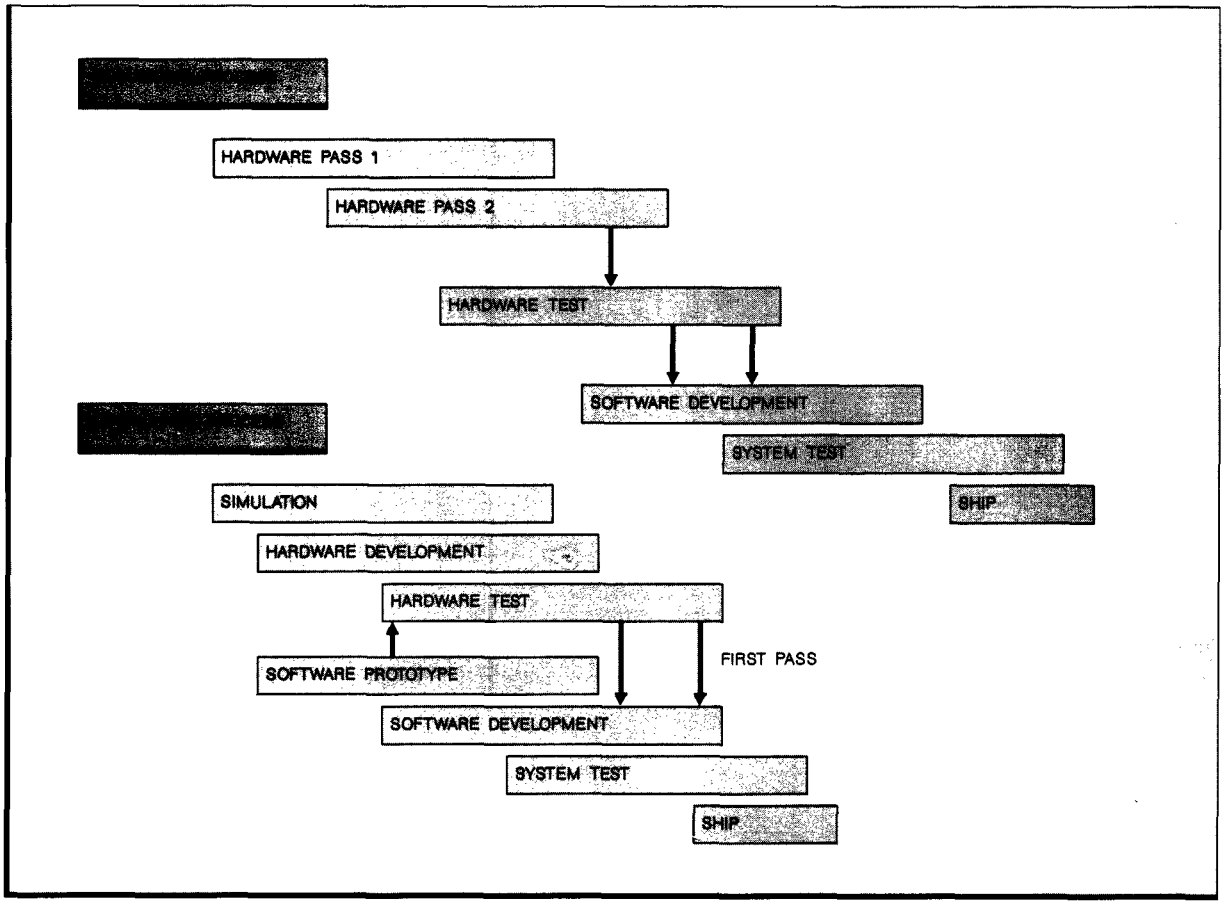
Cycle synchronization. The functional requirements stated were known from the outset as a given for the new system, but the immediate requirement was to shorten the development cycle. The engineering developers could shorten their cycle, if necessary operating-system function were available to allow engineers to test the hardware with existing test cases from previous systems. This, in addition to extensive hardware simulation work, would allow for the earlier delivery of hardware to programming personnel. Therefore, an early prototype was defined and built to synchronize software deliverables with hardware availability and meet this requirement. This interdependence significantly overlapped the hardware and software development cycles, as shown in Figure 4. Interdependence achieved greater parallelism in the development sequence and reduced the magnitude of independent hardware testing. It also allowed early evaluation of performance, usability, and serviceability characteristics of the system, thereby reducing rework late in the development cycle that would otherwise have been required to improve these characteristics. As a by-product of this activity, the integration and build procedures were created and debugged early in the development cycle.

Management techniques and controls

In addition to combining the past processes to unify the prior organizations toward a common goal, the management reporting structure was also completely realigned. Organizational management span-of-control was changed from having responsibility for many functions on a single system to specialized functional responsibility for all systems supported. Personnel with experience in a given function were collected into a single organization. For example, communications specialists on both System/36 and System/38 were organized under a single management structure, having responsibility for AS/400 communications as well as continuing responsibility for System/36 and System/38 communications. This reorganization had the effect of simplifying resource balancing during plan generation. With a single management organization having responsibility for supporting multiple system development efforts and owning all the resource available with expertise in the given function, the plan generation process could be conducted more efficiently as it balanced the resource across the various products. The organization was also constructed so as to isolate the development team from outside distractions, thereby keeping the developers focused on the task of producing the end product. Figure 5 shows this isolation, created by channeling the activities involving organizations outside the Rochester Programming Center through specific internal organizations designed to deal with them, thus reducing or eliminating the time-consuming multilocation activities from the direct development team.

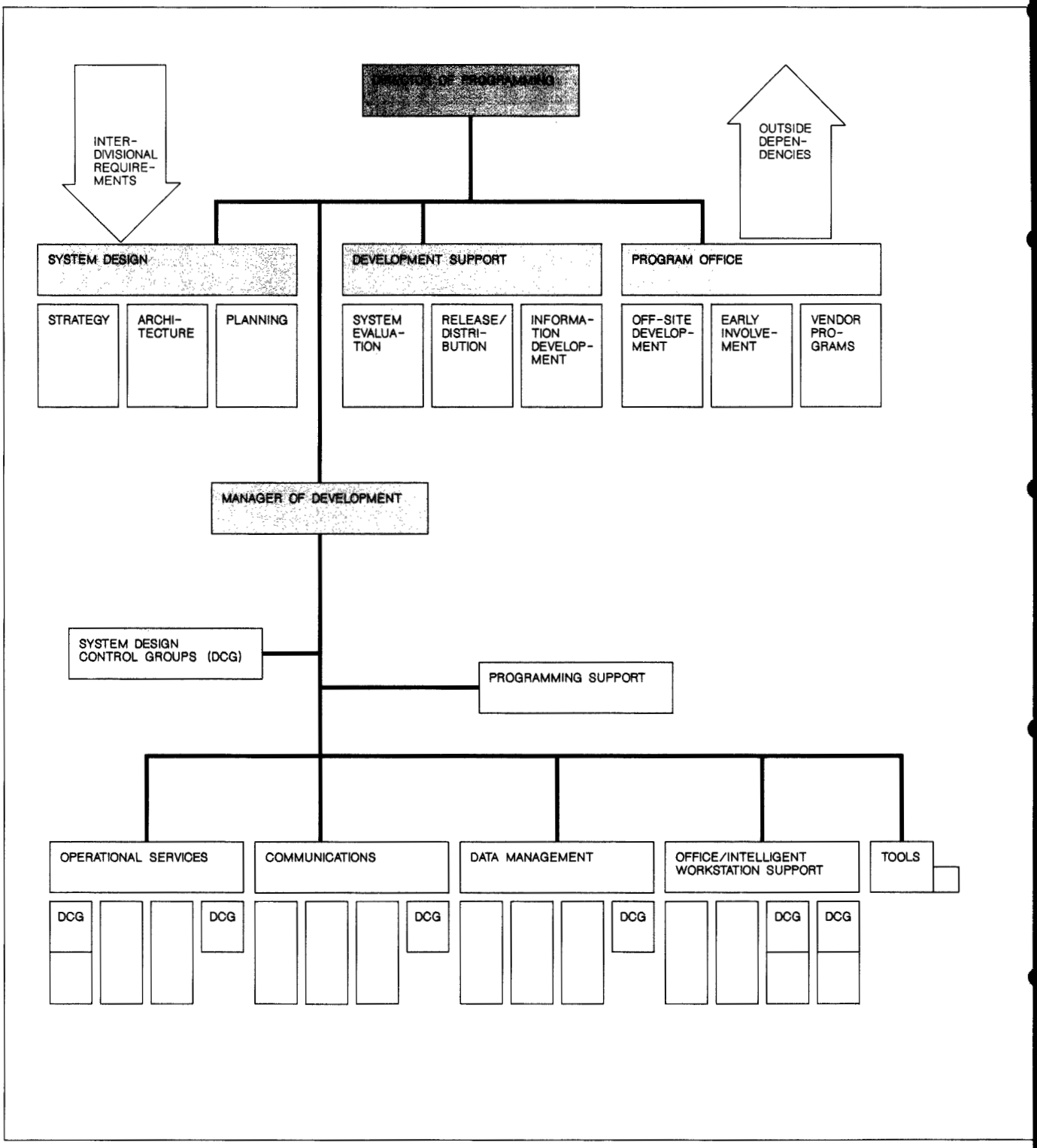
Change control techniques. The next challenge was to manage the plan content throughout the development phase. Specific tools to aid in project management had been tried before, and they all exhibited the shortcoming of failing to provide a means for

Figure 4 Hardware/software synchronization



formal communication of technical details and decisions in conjunction with schedule communication. This was viewed as necessary for effective control of change during development and was based on the project management and change control concepts used successfully on both System/36 and System/38. In those projects, the technical information and schedules were maintained in the same report for each element of development activity. Therefore, a similar technique was also established for AS/400 development. Every plan item was translated into one or more change control elements, which would be tracked and managed through every step of the development process from plan definition through delivery for testing. The change control data were structured to provide the documented technical and schedule information necessary at every step of the process. Guidelines were set up that defined the information to be completed at each step. The technical information was on line and constantly available to all developers, with the guidelines available as help text. Merely having guidelines is not sufficient to ensure change control; rather, it also requires management focus. Therefore, as was done on System/36 and System/38, a status was defined for each of the various levels of completeness; reports were defined that produced documents with appropriate technical and project management information necessary to allow constant review of progress; and weekly meetings were set up by the DCGs and by development management to coordinate technical interdependencies and monitor status transitions and schedules. This approach combined technical control with process and schedule control, and it was applied to initial design and development as well as follow-on design changes. In summary, the devel-

Figure 5 Organization structure



opment plan was under change control from definition to release and had constant management supervision.

Close control of code integration and build activity was also viewed as essential to development stability. It was desirable to maintain operational test systems from the start to the final product, so developers would not be required to build and maintain their own system versions. Continuous integration and weekly builds were established to provide timely test systems and minimize development-cycle time. Control of the software part libraries was integrated with the change control process in such a way that a specific change control element status was required before the code could be integrated. Thus all code being integrated had a valid reason that had been communicated and understood by all affected parties. This gave credibility to the change control process, in that it relied on that process to ensure that dependencies had been properly managed. The effect was to prevent catastrophic failure when new code was integrated into already existing code. Weekly builds kept development systems current and avoided rediscovery of known problems. To ensure that each build was successful, a build verification test was defined and conducted upon completion of the build, before propagation to the development systems. In addition, a special team of individuals skilled in problem solving was established to increase the efficiency of the problem-isolation tasks during development.

Similar control was also applied to integrating problem fixes during the testing phase of development. Once integrated, code could not be reintegrated without a valid reason. Fixing a problem does not require the same level of technical communication as the development of new code. Therefore, different information and guidelines were established for problem tracking, reporting, and integration. This information was focused on two-way communication of the problem symptoms and the fix solution. It, too, was integrated with the software part libraries in that a specific problem-tracking element status was required before code could be integrated.

Design control groups. Design control groups (DCGs) played a key role in development. They were composed of technical experts in each area of the system and reported to the managers responsible for the major functional areas of product development. The design control groups are the resident consultants for the first-line developers. A system-wide DCG, composed of members with system-wide technical expertise, reported to the manager responsible for all software development. This technical hierarchy provided a growth path for technical experts and provided a valuable resource for management decision making. It allowed the management team to focus on project management with confidence that technical control was being well handled. DCGs acted as buffers for the developers, in that they handled setting most of the plan-item priorities and high-level definition activity that would otherwise severely reduce a developer's productivity. They provided the initial assessment of the technical feasibility of plan items. They focused on providing timely solutions to pervasive problems and to bridging the gap between architecture and planning personnel and the developers. This ensured consistency with the system design direction and completeness of design during development. The DCGs represented about 10 percent of the development resource.

On a monthly basis, a project management team focused on problem identification and progress.

Dependency management. Although the change control techniques significantly enhanced development communication, other meetings were required to coordinate activities with other organizations supporting software development. Weekly interlock meetings, as they were called, were scheduled by software development with software support and engineering organizations. These meetings had a preset agenda of required topics and attendees. These were structured meetings that were generally held at the same time and place each week; they focused on problem identification and progress, rather than problem solving. Problem-solving activities happened outside the meetings, where the most knowledgeable people would be involved to ensure the correct decision was made. On a monthly basis, a project management team (PMT) of key development

managers met in a similar meeting, once again focused on problem identification and progress. Also, the overall system manager held a quarterly meeting with upper-level management to focus on the system-wide issues. Division-level managers did the same. At critical phases of development these meetings were held more frequently on any topic that required attention. In all cases, they served a valuable purpose of gathering the persons who would deal with problems and development issues.

By conducting a system-level review, the requirements were validated and development work began.

Design and development

Together the System/36 and System/38 consisted of over five and one-half million lines of code. To rebuild the functions with all new code, given our resources, appeared to be beyond reasonable expectation. Attention turned to an experimental project that had demonstrated the ability to run a System/36 application in an environment built on a System/38. This pilot project gave hope to the idea of reusing major portions of the System/38 software and potentially reusing significant elements of the System/36 software design. The advanced architecture of the System/38 could be used to build upon, so that the new system would inherit the strengths of that architecture. Also, adding the System/36 ease-of-use strengths for application processing would meet a major portion of the system requirements.

Development objectives. The first objective in AS/400 development was to deliver the necessary software to support hardware testing. The DCGs played a key role in translating this objective into the first set of stable requirements on which development could begin. Elements of the system were identified that could be reused from previous products, and new elements were identified that would be consistent with the new hardware interfaces and system design direction. By conducting a system-level review, the requirements were validated and development work began. We met the first objective by a series of software and hardware drivers. The software drivers had only software dependencies and could be built and tested on existing hardware, before the new hardware became available. The hardware drivers depended on the new hardware to fully test the code. When they were combined they became the development prototype that supported the hardware testing and accelerated the initial release of hardware to programming. The engineering team focused on producing high-quality, first-pass hardware through the use of extensive simulation techniques and, with the extensive testing capabilities provided by the software prototype and test cases reused from previous systems, engineers were able to deliver the first-pass hardware to programming personnel for software development and testing.

The next objective was to provide sufficient functional capabilities to allow System/36 applications to run on the new hardware. With the initial prototype as a base, additional function was added to demonstrate this capability. Process metrics became very important during this phase, as the project progressed toward its first committed milestone deliverable. This was a product that would be shown to customers through the early user involvement programs to obtain the feedback critical to the definition of remaining requirements. The metrics established were designed to detect anomalies in defect removal, module size, and code complexity, and indicate the cost of quality at each development step. Reporting techniques were established that would provide current component and department data for feedback to the developers and management. Process compliance was also depicted by the data. Regularly scheduled feedback meetings were established to evaluate data and discuss shortcuts to increase productivity without affecting quality, and to analyze defect-prone parts or components for actions necessary to ensure high quality. Performance measurements could also begin with this first system milestone.

In the next iteration of the development cycle, we focused on the ability to run System/38 applications. This cycle was designed to provide tools for an application programmer, including the system control language and an application development language, and to support an ease-of-use interface that the end user would see, as previously prototyped.

Significant system test, information development, and national language translation involvement could begin with this level of functional capability. To prepare for this, these organizations were required to be involved with the development process much earlier than traditionally expected, as shown in Figure 6. The result was that resources for these activities were spread over a longer time period in the development cycle, thereby reducing the extreme peaks traditionally experienced late in the cycle. With this milestone, external users could be shown the functional capability of the AS/400 system so they could plan the way in which they would migrate applications from the predecessor systems. Usability evaluations could also begin to ensure that the prototyped interface was correctly implemented and that they met the usability objectives.

Throughout development, progress was tracked against defect removal models.

The third milestone focused on bringing the existing function up to a deliverable product level and adding the application programming languages necessary for system level testing (milestone test) to begin. Application migration aids were added to help bring across test cases from the predecessor systems and to allow the next phase of external user involvement to begin. During this phase, IBM application developers and external users would begin migrating critical applications, in preparation for concurrent availability with the system. This activity was critical to the system objective of providing complete solutions for customer needs.

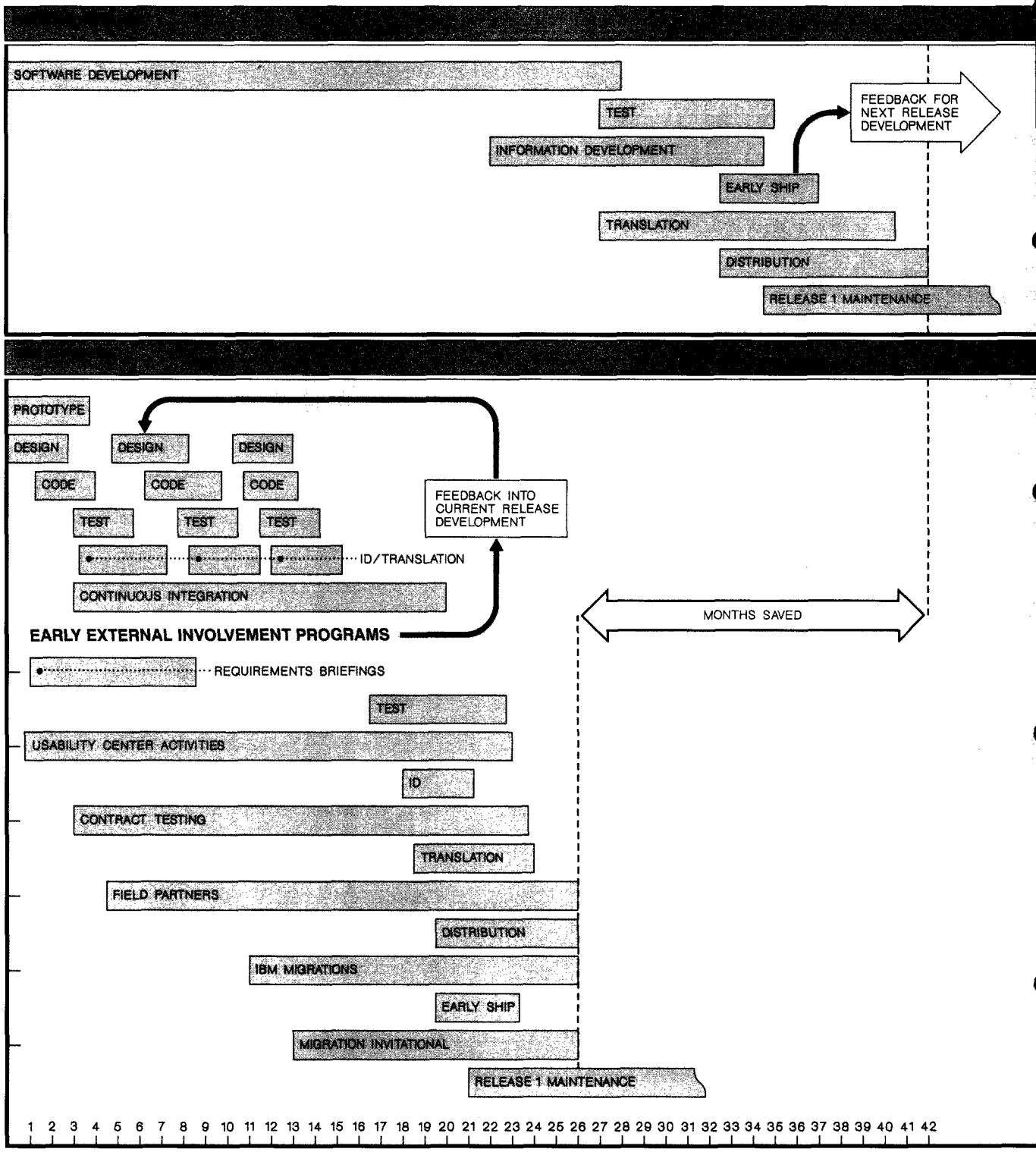
The last iteration of the development cycle produced the final product for system test entry. This included the final version of translatable text and publications that was sent to countries around the world for translation into their national languages. Attention turned from defect prevention to defect removal, as the product proceeded through the final cycle. Throughout development and especially in this last iteration, progress was tracked against defect removal models in an attempt to measure the effectiveness of the process as well as the quality of the product. Change control focused on the process for fixing problems to ensure that the code being integrated would not create additional problems. Measures like the number of problems in a backlog, problem severity, and length of time to fix a problem became very important, as the schedule closed in on the final dates.

Performance design points. An early review of the existing performance process determined that changes would be necessary. The process had evolved over many years of System/36 and System/38 development and was chiefly measurement-based. This resulted in major changes late in the cycle as performance problems were discovered, a characteristic that could potentially destroy the ability to meet the required delivery date. In response to this concern, a new second-line performance area was established to define and implement a performance process that would eliminate this exposure. The resulting performance process was called the *design point process*, because of its having a requirement to establish performance design points early in the product development cycle. The working relationships for the process were documented and agreed to in advance. Performance-critical functions of the system were identified, and an early agreement to follow the process was reached with the developers of those functions. Both performance and development personnel knew what was expected of them.

In the design point process, the end-user tasks supported by a system function were determined, and the high-use transactions were identified. These transactions were grouped according to the acceptable response time for the activity (for example, ranges of less than 1.0 to 2.0 seconds and greater than 2.0 seconds). Reaching agreement on response times brought an end-user perspective of performance to the developers.

Next, the transaction control flow was defined and design points for performance-critical components were set according to the expected response time. The design points were stated in terms of the executable instructions and objects touched while running the component's function. Development personnel estimated the corresponding path length and a series of iterations took place until both parties agreed on the time values, making tradeoffs within and across components as appropriate.

Figure 6 Cycle time improvement



As development proceeded, the design points were tracked against the estimates in the design and code reviews and were adjusted with each level of refinement. The adjustments were fed into the system-level model to ensure the end-user requirements were being met. The results were summarized and presented to management regularly.

Measurement occurred during component test, on the first milestone into which the component was integrated. Early measurement substantiated the estimates and, in the event of a problem, gave development personnel time to solve the problem before shipment.

Usability development. Usability activities were put in place early in the development cycle to ensure that the delivered product would provide superior levels of user productivity, satisfaction, and ease of use. A very high level of function was designed into AS/400 Release 1, so particular attention was paid to the development of on-line facilities for user education and information. In addition, implementing an interface that is highly consistent was emphasized. This was accomplished by writing specifications for the user interface and using screen, command, and message review processes to ensure compliance. A user interface prototype was then developed on the first milestone to test the acceptability of the initial user interface design early in the development cycle. Based on feedback from these early tests, the user interface was enhanced to reflect user needs. Subsequent prototype testing enabled further refinement of the user interface. Customer involvement in this work began at this early stage and continued throughout the development cycle. Initially, customers provided feedback on the specifications through a series of focus-session round-table discussions in which the initial design was reviewed. Later, functional demonstrations were conducted, using the prototype to provide a more realistic view of the interface and to solicit additional feedback. When functional hardware and software was available for a specific task, full usability testing was conducted using customer test subjects, recording time-on-task, error rate, attitude, and satisfaction measurements.

In addition, early in the cycle, an assessment tool was developed to determine the extent to which the design was going to satisfy the needs of different user types. Output from this assessment enabled us to set priorities for subsequent usability activities and to understand the positioning of the AS/400 system in comparison with its predecessor and with competitive products.

Toward the end of the development cycle, a series of usability certification tests were performed, again involving customer participants. The points of comparison in much of this work were the System/36 and System/38. These late-cycle activities demonstrated that we had achieved the user interface characteristics we had targeted and that AS/400's user interface was indeed a step forward.

Product verification

The size of the product and the scope of the development process were daunting for the verification and testing team. The AS/400 system was over twice the size of any previous product or release tested in the Rochester Development Laboratory, both in lines of code developed and processing power. If the traditional verification and testing process had been followed, the AS/400 system would have been shipped to its customers months beyond the date required for market competitiveness. For us to be successful, new technical and control processes had to be put in place to handle the size and magnitude of the project, without compromising the resulting product function and quality.

To meet these challenges, an independent system test organization was established from similar organizations for System/36 and System/38 testing. Little development and verification experience was available in Rochester with a project this large. To make it work, we decided first to carry over test philosophies, strategies, and the key strengths of the previous testing organizations. Then, by employing new techniques and innovations, we would meet the challenges posed by the size and schedules of the AS/400 system.

Four-phase test process. With the desire to involve users in the development process through the use of milestones, we established a test process that would verify the quality of each milestone. This had to be done from an end-user perspective, which implied product- and system-level testing on each milestone. In the *first phase* (informal unit testing conducted by the module developer prior to integration), the focus is on the individual program module (or unit) readiness for integration. Variable limits, internal interfaces, logic, data paths, and code paths are all tested before the code is made available to others for general testing. In the *second phase* (formal component testing after integration also conducted by the development organization), discrete component functions and interfaces are tested in a system envi-

ronment. Formal, repeatable test cases were used, many of which were developed with the same design review and code inspection rigor as product code. In the *third* and *fourth phases* (formal product and system test), the product and system focus began, thus providing the level of testing required for customer involvement programs. Therefore, a structured milestone test conducted by the independent testing organization was defined for each system milestone. The independent system test organization's objective was to represent the first customer as they tested each milestone and feed a customer's perspective back to development. Milestone testing had other benefits. The earlier product-level and system-level evaluation of function forced earlier involvement of the system test group in the development process. It also forced earlier stability in key system elements that were delivered in the earlier milestones.

Formal, repeatable test cases were used.

Testing cycle considerations. Milestone test evaluated the product from an end-user perspective, as with system test, but in a nonstress system environment. All components for an end-user function were tested together. Milestone test was a subset of system test but, like system test, concentrated on system (hardware and software) characteristics such as reliability, usability, and performance. With milestones, system testing could be staged, thus reducing peak resource demands late in the cycle by spreading the testing effort over a longer period of time. Where traditional sequential schedules required all the function to be available before starting the testing process, the milestone concept allowed significant overlap of development and testing activities. This high-level testing focus early in the development cycle uncovered areas requiring improvement while there was still time to react before shipment. The focus actually began during the development of the prototype. With this early software deliverable, the test teams could begin design and development of the testing techniques to be employed during the formal milestone test and system test periods.

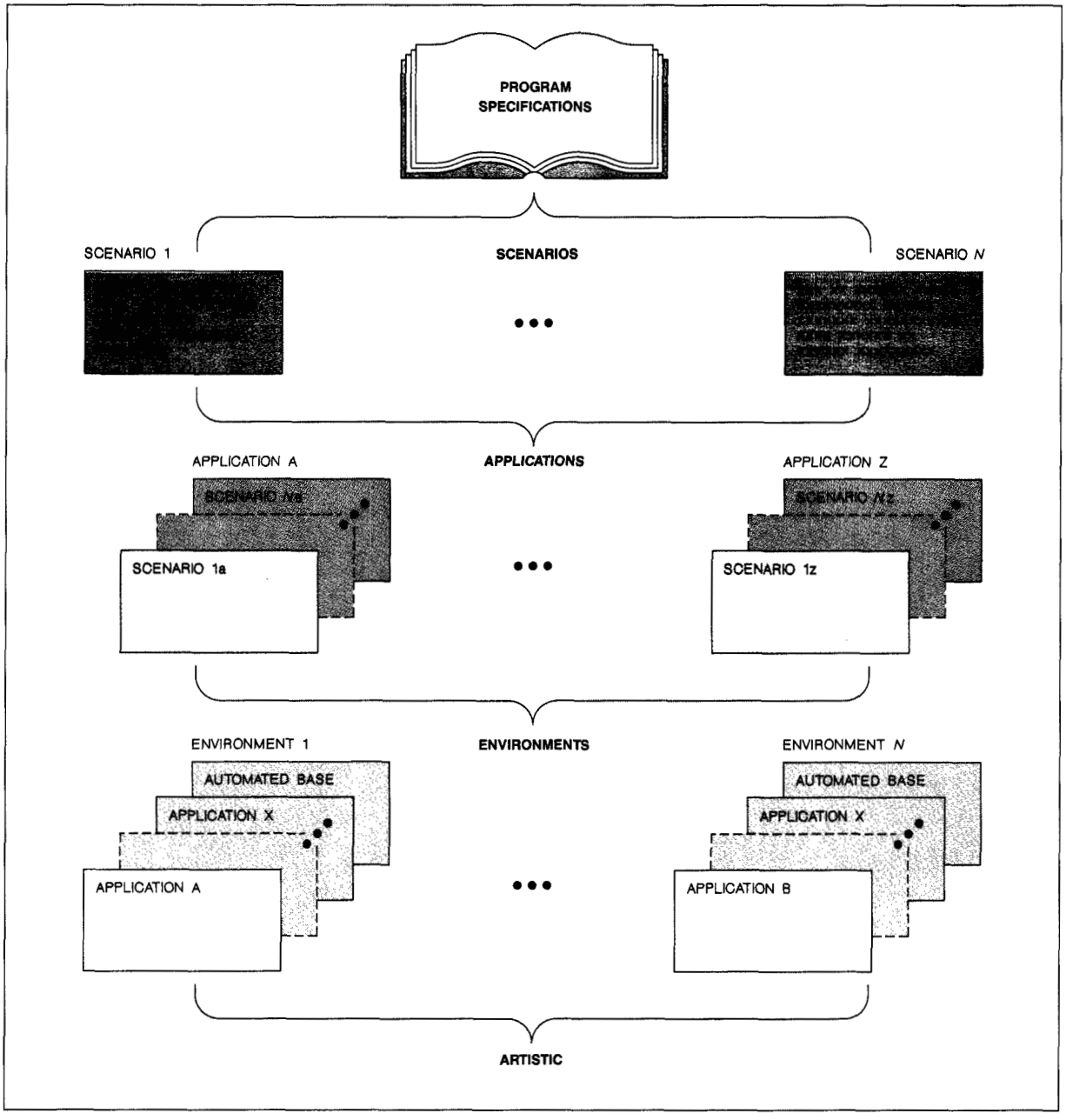
The last test phase, system test, tested all products and end-user functions together in a multiproduct, multiuser, and multisystem configuration. It started after all functional milestones were available. System stability and reliability were verified for extended periods of time under loaded and stressed conditions. Reliability and defect metrics were tracked throughout system test.

Reuse in testing. The milestone and system test strategy was based on the standard test processes from System/36 and System/38 and expanded where necessary to include the advanced functions of the AS/400 system.

A significant portion of the existing scenarios and applications were brought over to run in the System/36 and System/38 environments on the AS/400 system, whereas others were modified as needed to run the same function in the native AS/400 environment. Testing was required for stand-alone systems, systems with a peer relationship (AS/400 to AS/400, AS/400 to System/38, and AS/400 to System/36), and systems connected to a host (AS/400 to System/370), including both high- and low-end system models.

In Figure 7, the four basic building blocks used to create milestone and system tests are illustrated: (1) test scenarios, (2) test applications, (3) test environments, and (4) artistic tests. Scenarios were user-level tasks defined to test specific functions of the product, such as for example, creating, compiling, debugging, and running an RPG or COBOL program, or distributing mail and documents across a multisystem communications network. Scenarios were created and debugged during milestone test and run in several variations or versions, depending on the function being evaluated. Applications were selected combinations of multiple scenarios that were run concurrently or serially. Applications were designed to stress product versatility, because individual products may have several applications. Scenarios and applications were automated during milestone test, where possible, so they were easily repeatable in system test and could be combined with other scenarios to create new applications. Test environments were selected combinations of multiple applications that were run concurrently, such as an office test environment, a communications test environment, and a programming development test environment. Test environments were designed to stress cross-

Figure 7 Test building blocks



product concurrency and were designed to emulate end-user or customer configurations and situations. Artistic testing included all testing activities that were not easily automated as part of a scenario, application, or environment, or were unstructured tests targeted at known or suspected problem areas, such as testing error recovery and support or testing complex function and configurations. The objective of artistic testing was to drive out problems that might not be encountered with well-structured, repeatable test cases. Planning for artistic testing allowed testing personnel the freedom to focus additional time and resources on suspected problem areas and the freedom to be creative with their testing.

A large, complex network test was added to the test strategy. This was a cooperative effort between the programming development and system evaluation groups to set up a large network that simulated the larger and more complex customer networks that existed in the field, including System/36, System/38, and AS/400 systems. Testing focused on evaluating characteristics and problems, such as network reliability, error recovery, and problem isolation and determination.

Automation advantages. Two key aspects of the milestone and system test strategy were test automation and system reliability measurement. Test case automation is the foundation of the entire milestone and system test strategy. Automating the test scenarios saved both personnel resources and calendar time. Once a set of scenarios was automated, it was easily assembled into the applications to provide an automated test environment. Automation also provided a way to drive the system utilization to the maximum limits. Automated jobs were set up to provide a background system load. These automated jobs were able to run 24 hours per day, 7 days per week, providing a means for reliability measurement while using off-hours effectively. They were also run during the debugging and running of the new AS/400 test scenarios for milestone test and during the system test period to provide an increased system load while doing these activities.

A method to measure reliability was a necessary part of system evaluation. Using test data from System/36 and System/38 history, aggressive goals and expectations were created for the AS/400 system. Reliability measurement data were gathered from several systems in the controlled test environment over a week's time. There data were tracked closely against the established goals. The measurement was based on the time between unscheduled initial program loads (IPL) and defined as a four-week weighted average of the weekly system mean time to unscheduled IPL (MTI), which is calculated by the following equation:

$$\text{MTI} = 0.5\,\frac{H1}{I1 + 1} + 0.3\,\frac{H2}{I2 + 1} + 0.15\,\frac{H3}{I3 + 1} + 0.05\,\frac{H4}{I4 + 1}$$

Where:

H = Total system run hours in week(n)
I = Number of unscheduled IPLs in week(n)
Current week = 1

An unscheduled IPL was any IPL that was unplanned, usually due to a system problem or recovery from a severe problem that caused system-wide failures. Total system run hours was the sum of the hours accumulated by all the test systems operating in a live or automated testing environment. The MTI measurement was a good indicator of overall progress during milestone and system testing.

The challenges of staging. The staged delivery of product function in system milestones also provided a major challenge to AS/400 product verification. Even though a formal plan had been put in place to manage and evaluate the delivery process, many product dependencies existed, which made testing the early milestones with scenarios and applications difficult. The heavy overlap of component testing with milestone testing created a situation in which many of the intercomponent interfaces and user interfaces were changing during the test period. The changes led to frustration because the test personnel met the same problems as the developers. This made it difficult to automate scenarios. The problem backlog grew rapidly and received weekly and daily attention by development and test management. To help control the process, strict test entry and exit criteria were established for functions entering milestone and system test. This included an analysis of the problem backlog to identify problem-prone areas requiring additional test or attention by development personnel.

Another major challenge faced in system testing of the AS/400 system was one of skills availability. Additional skilled people were needed to develop the testing, and the existing skills—based on the System/36 and System/38 experience—had to be upgraded quickly. However, the product schedule did not allow for lengthy education plans. The problem was solved by getting the testers involved with the product development wherever and however possible. System test design guides and test plans were written earlier in the process, using product design information. System test personnel made themselves available to assist in unit and component testing, helping the product make its development schedule checkpoints while gaining the needed product and system education.

Earlier development by test personnel involvement helped address the current skills problem, but a resource shortfall remained a likelihood. Staffing plans to meet the challenges indicated that a threefold increase was required that would last many

months. Several methods were used to address this problem, including vending, university testing, and assignment changes.

Vendor and contract programming companies were contracted to provide skilled and experienced programmers for testing. These contract programmers worked either as an independent team working offsite on a specific deliverable or they worked directly with the IBM test personnel as an integral part of the IBM test team. The contract programmers brought a level of experience and knowledge that proved beneficial in a short time, but it represented additional project expense.

Teams of students were hired to test specific function under the guidance of a faculty adviser.

Two area universities were subcontracted to do testing. IBM supplied the needed hardware and software for a university-controlled, secure testing facility. Teams of students were hired to test specific function under the guidance of a faculty adviser from the computer science department. The student test team worked very closely with IBM to develop the test plans and test cases. The level of enthusiasm and commitment of the students was very high, because they were eager to put their education to work and gain job-related experience. The students also gained experience working in a team environment under schedule commitments and acquired AS/400 knowledge that would be valuable to them as they start their careers.

As development neared completion, the task of testing the system grew significantly. Earlier personnel transfers from the test organization to the development organization had depleted the resource and skills necessary to develop and run a thorough system test. The "full team approach to development" was then reversed and became a full team approach to testing, as development personnel transferred to the test organization applying additional resource to ensure effective testing of product function in the required time frame. This required commitment by all levels of management that was demonstrated by the willingness of development managers to transfer into the test organization. This provided the leadership that was key to the willingness of development personnel to transfer to the test team to complete the testing and avoid schedule delays.

Ensuring user acceptance

Customer acceptance was critical to the success of the AS/400 system. The only way to discover customer reaction with any certainty was to have end users provide their reactions. With this in mind, a number of related programs were set up to provide insight into the marketplace reaction prior to shipment. Customers, Business Partners, consultants, and systems engineers (SEs) from the field were contacted to participate in programs that would be mutually beneficial to the participants and the development organization.

The Field Partners program brought SEs to Rochester to obtain the field perspective. They provided additional resources for component and system test activities while learning about the system capabilities. Their individual product and field expertise was used to help design and create individual test scenarios, to review product documentation, and to do other tasks that were containable within the three- to eight-week time period they were assigned to the project.

The contract testing program involved two consultants, one for significant testing of the System/36 environment throughout the development cycle and another to provide an independent verification of system quality. The independent system test (IST) was set up using an offsite, non-IBM programming center and contract programmers. This independent programming center was provided with the required hardware, software, and tools, but was given limited direction. The programmers were allowed to proceed on their own and establish and run their testing independently. IBM reviewed their test plans and made sure that they received the information needed to proceed with their testing. Their test results were also reviewed, but no attempt was made to influence how they tested or where they focused their resources. Weekly review meetings were held to assess their progress and problems and provide them with the current driver updates. By reviewing their progress and results, development and test organizations were able to get an independent assessment of the

severity of the impact of specific problems encountered during system testing.

A communications field test was set up to extend the system testing to customer communications networks not easily recreated in the laboratory. Customers were selected on the basis of the complexity of their network, their application, and their communications knowledge. The participants were provided with AS/400 hardware and software and weekly driver updates. This field test provided a truly independent view of the products tested. It also helped identify additional problems earlier in the development cycle, by using networks and environments not found in our own network testing. This field test further provided customers with a chance to test their own applications earlier and get a head start working with new products as well as developing a working relationship with the development laboratory.

Business Partners and customers were invited to test migrating and running their applications on the AS/400 system.

IBM migrations provided interactions with the application programmers in the Application Systems Division to gain their insight on the system, while they were migrating MAPICS and DMAS applications to the AS/400 system. However, the most significant program was the migration invitational, in which 175 Business Partners and customers were invited to Rochester to test migrating and running their applications on the AS/400 system. This program provided a greater variety of test applications than what is normally achieved with internal testing.[2]

In addition to testing and improving the quality of the AS/400 software, these programs allowed other than IBM people to apply their unique knowledge, experiences, and skills to writing and reviewing online help text and printed examples, as well as to testing software function. While all of these persons were applying their knowledge to provide significant additional testing beyond the normal development process, they were all learning about the system themselves and preparing for its announcement. Whether migrating an application to be announced as commercially available at the same time as the AS/400 system or converting their own specific application for their business needs, it was an environment where all participants derived a benefit.

Worldwide announcement and general availability

The release and distribution process also received a major challenge in the objective of concurrent worldwide announcement and general availability for all products in all major countries. No system or product of this size had ever shipped concurrently with as much translatable material, translated into as many national languages. The first major challenge was the translation of displays, messages, on-line help text, and manuals into 25 different national languages. In the past, each national language required a unique version of the software. The software had to be built with the translated text included, functionally verified after the build, and distributed within the country. This process normally resulted in software shipment to countries outside the United States and Canada nearly three months before it could be generally available within that country, to allow time for the translation to be completed. To improve this process, the program messages, displays, and on-line help text were designed to be separate objects from the program source modules. This separation allowed the national language material to be packaged and managed independently of the source code. It allowed translation to begin several months before the source code had been completed, and allowed the translated material to be shipped as a separate object, selectable during system installation.

In addition to the objective of concurrent worldwide announcement and shipment, several other new factors were addressed for release and distribution. The AS/400 system had a new distribution medium in the form of quarter-inch tape, as well as the usual tape and diskette. A new ordering methodology was developed to improve the accuracy of the order definition for the customer delivery that required all ordering systems and tools to be modified. The AS/400 system was to expand the total system package (TSP) ordering option that was first introduced with the System/36. With the TSP option, the customer orders

a preloaded and customized system package that was created as one of the last steps in the manufacturing process. To address these challenges and deal with new areas of the business and new countries that were not familiar with the AS/400 product or processes, a distribution project office was established early in the development cycle. This project office was responsible for managing the translation, release, and distribution processes required for concurrent worldwide announcement and general availability. Members of the project office were able to identify problems early and often used the program management team to assist in getting problems and issues resolved.

Prior to worldwide general availability, the release and distribution process was validated through an early shipment program (ESP). Systems were shipped to selected customers or agents both within and outside the United States using the distribution processes established for the AS/400 system. Field service support was also provided, using the processes established for the AS/400 system. ESP was another program that involved customers in the development process of providing customer feedback and product verification. The primary objective of ESP was to verify the distribution and service support processes prior to shipment.

Concluding remarks

To accelerate the programming process, an iterative and innovative approach was used within software development. This approach can be characterized by three phases: (1) initial design, (2) milestone development, and (3) final evaluation. All three involved the end user as a key element of successful completion. The initial design phase consisted of producing a rapid prototype on existing hardware. This prototype became a tool for early user involvement in the process to verify known requirements and to finalize product requirements still under investigation. The development phase consisted of a series of milestones of system function that were intended to be a deliverable entity for user evaluation and feedback. The significance of this approach is that if a deliverable was not satisfying the user's needs, development personnel had time to make the necessary corrections to ensure satisfaction at product shipment. Final evaluation was shortened because of the prior testing of system milestones. This phase focused on ensuring the final product quality, performance, and usability through customer-supplied applications as well as traditional test cases.

Management techniques encouraged problem identification and change control from plan definition through final product testing. Design control groups formed a nucleus of technical expertise to ensure timely solutions to pervasive problems, consistency with the system architecture and design direction, and design correctness and completeness. Development methodologies, like continuous integration, weekly build and build verification, and expert debug teams, increased development efficiency and product quality. Together, these elements combined to ensure that AS/400 development achieved its goals.

Acknowledgments

The authors wish to acknowledge David L. Schleicher under whose direction this process was defined and followed; David G. Ness, Donald W. Van Ryn, and Leslie S. Wilkes for their cooperative leadership in the management of the Development Support, Program Office, and System Design organizations, respectively; Ronald O. Fess, Keith W. Fisher, Rodney H. Morlock, Robert J. Chappuis, Judith A. Kinsey, and Kenneth R. Johnson for their leadership of the development organizations; Beverly J. Gerzevske, Stephen F. June, Dean R. Ascheman, and Richard J. Hedger for their leadership of the development support organizations; and B. J. (Joe) Pine II who managed the early user involvement programs. We wish to thank William H. Walker, Thomas A. Hallstrom, Barry L. Smith, Diane E. Straka, and Kurt W. Pinnow for their contributions to this paper.

Cited references

1. R. A. Radice, N. K. Roth, A. C. O'Hara, Jr., and W. A. Ciarfella, "A programming process architecture," *IBM Systems Journal* **24,** No. 2, 79–90 (1985).
2. B. J. Pine II, "Design, test, and validation of the Application System/400 through early user involvement," *IBM Systems Journal* **28,** No. 3, 376–385 (1989, this issue).

Richard A. Sulack *IBM Application Business Systems, Highway 52 & NW 37th Street, Rochester, Minnesota 55901.* Mr. Sulack is the programming development manager of OS/400 and related products. He joined the IBM Rochester Development Laboratory in 1962 after receiving a B.A. degree from Winona State University and subsequently entered a programming training program. His first assignment was as a systems diagnostic programmer in the former Data Products Division, Kingston, New York, working on the New York Stock Exchange Project. In 1965, he was assigned as a systems programmer on the Time-Sharing System/360 Project

in the then Systems Development Division (SDD), Mohansic, New York. He advanced to manager of the TSS/360 Resident Supervisor in 1968. Also in 1968, Mr. Sulack attended the Systems Research Institute in New York City. He transferred to the Rochester, Minnesota, SDD Programming Development Center in 1969 where he held management positions in DOS/OS data management technology and advanced technology. He joined the System/38 programming group as manager of language/utility development in 1973 and advanced to third-line manager of the group. He then joined the System/38 Control Program Facility development group as manager of data management facilities. During 1982 and 1983, Mr. Sulack managed the System/38 Kanji Project management office reporting to the System/38 system manager. He was promoted to program manager in 1983 reporting to the IBM Canada laboratory director. As the manager of general systems programming in Toronto, he was responsible for language and utility programming for System/36, System/38, and future products. He returned to Rochester in 1985 as manager of System/38 programming. In 1986 he was appointed the third-line manager of communications programming, responsible for System/36, System/38, and AS/400 communications programming products. In August of 1986 he was promoted to his current position as development manager of all AS/400 software in Rochester.

Richard J. Lindner *IBM Application Business Systems, Highway 52 & NW 37th Street, Rochester, Minnesota 55901.* Mr. Lindner is an advisory programmer responsible for the software development process strategy. He joined IBM, Rochester, Minnesota, in 1966 and participated in both engineering and programming development of System/3 with responsibility for developing and maintaining many aspects of the I/O supervisor. He then accepted an assignment in the development of the System/38 Control Program Facility. Since then, he has held management assignments in both development and development-support areas. He became a professional engineer through training in the IBM undergraduate engineering education program in conjunction with the University of Minnesota.

David N. Dietz *IBM Application Business Systems, Highway 52 & NW 37th Street, Rochester, Minnesota 55901.* Mr. Dietz is an advisory programmer and manager of languages and utilities system evaluation in the product evaluation group. He joined IBM, Rochester, in 1977 and was responsible for software cost estimating. He has held a number of assignments in the System/36 and System/38 development support organizations where he was responsible for software quality plans, process improvement, and quality measurements and reviews. In 1985, Mr. Dietz accepted an assignment on the division and group headquarters staff in White Plains, New York, where he was responsible for quality and productivity measurements and definitions. In 1987, he returned to Rochester to accept a management position in system evaluation and was responsible for system evaluation of AS/400 office and related products. Mr. Dietz received a B.S. in computer science, physics, and mathematics from Mankato State University, Mankato, Minnesota.

Reprint Order No. G321-5366.