Process automation in software application development

by K. D. Saracelli
K. F. Bandat

Over the years, the field of application development (AD) has evolved from that of an art form to being more of a science, hence the emergence of concepts such as information engineering. In engineering and scientific fields, the value of process definition and management has long been known. This paper discusses the requirements for managing the AD process and establishes the need for automated assistance for these management activities. Considerations for an automated system to manage the process are presented, and the benefits to be realized by such an implementation are then discussed.

The development of software applications is certainly one of the more complex human activities. For large projects, it requires the close cooperation of many people with distinct responsibilities. In an effort to bring more structure to development efforts and increase productivity and application quality, current trends in development have leaned toward an engineering approach using computer-aided software engineering (CASE) tools and more rigorous methods. CASE tools offer a more structured way to do data and function modeling and development by offering tool support in these areas. However, in order to achieve higher quality products and to increase productivity, activities must be planned, managed, and evaluated. The people on the project team represent a critical factor toward achieving the goals of a project. If we can create a synergy between the project team and a managed process, and enable people to perform creative work more productively, our outlook for predictably successful projects is considerably enhanced.

As in other industries, managers of software application development are faced with the challenge of increasing the productivity of their development teams, and at the same time, improving the quality of the applications they deliver to the end user. Factors that greatly influence their ability to affect these challenges include: reliable prediction of project costs, tracking development projects already in progress, meeting committed schedules, securing and keeping adequately skilled designers and developers, and being able to handle changes in a timely and controlled manner. Faced with these management and process-related problems, they find themselves reacting to, rather than being in a position of proactively managing, the day-to-day work. The methodology for a development effort of 6 to 12 months often consists of 400 to 500 identifiable tasks. Performing and managing such a large number of tasks does not leave project managers and teams much time to look at improving productivity and quality. Thus these important areas can quickly become obscured by daily pressures.

Historically, there have been many tools and techniques that were intended to increase productivity and quality. These tools and techniques have been used in application development with some success. However, the benefits have been

incremental. What is needed is a more holistic approach, one that can be used by developers, designers, analysts, and testers alike, to yield repeatable, predictable results and alleviate the churn that dealing with chaos can cause. Watts Humphrey (formerly of the Software Engineering Institute) felt that to attain this approach, we would need to provide more advanced tools, techniques, and management, but that it would take sound process management to achieve required quality objectives.[1] Applying process management principles without a great deal of automated assistance can provide the essential basis for process improvement, including repeatability, predictability, and a greater awareness of the activities and deliverables. However, an application development (AD) solution that includes an automated process management system can provide a more consistent and repeatable basis for *managing* application development efforts and for improving them.

A methodology solution for application development. A *methodology* is a collection of methods and tools, designed and arranged so as to provide guidance in achieving a specific objective. Whereas a CASE environment provides a set of tools for the AD work, a methodology defines a consistent and efficient way of applying these tools, and adds support for the aspects of project planning, administration, and evaluation. A methodology includes the enactment of a defined process. It provides the guidance for the entire life cycle of application development, using what exists in AD: deliverables (the output, or work product, resulting from a given activity or task), tools, and methods, to name a few. In addition, it must provide the framework for a logical and smooth progression of work from one activity to another, eventually delivering the final application to the end user. The value of the methodology is that it provides predictability and repeatability, encourages adherence to organizational standards for quality, and uses the accumulated experience of the developers. Thus, the developers are not left to figure out how to do development on their own but can capitalize on this knowledge and experience.

Components of an AD process solution. A well-thought-out AD process is an integration of process management, project management, and methodology. These three form the infrastructure upon which development projects are performed. Metric collection and analysis must be done for each specific project. The activity-related information, such as completion date, duration, resources required, and the deliverables produced, needs to be tracked and analyzed so that any out-of-line conditions can be handled and the project replanned as appropriate.[2]

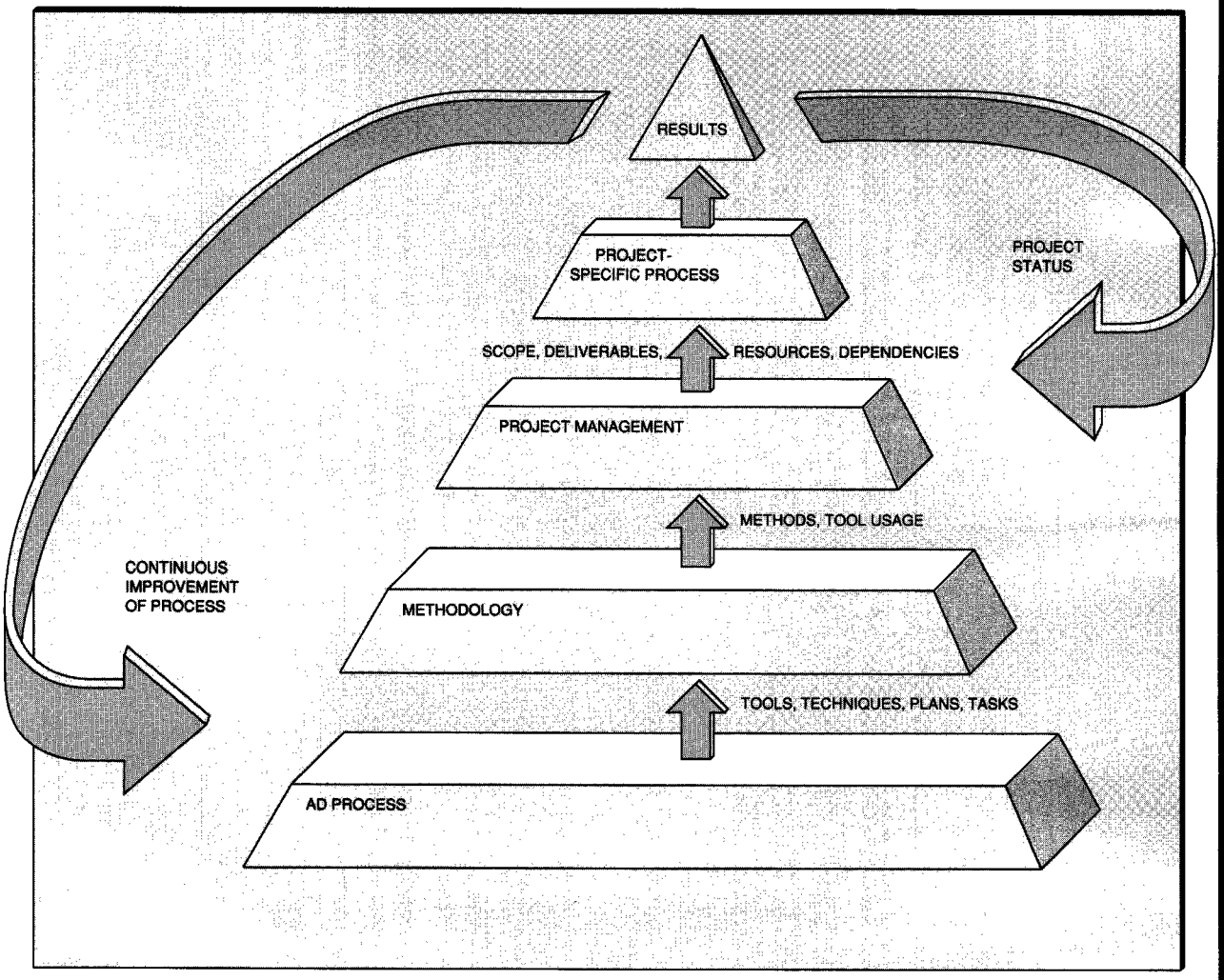
Figure 1 illustrates the interrelationships of these components. First, within a specific project (represented by the top three tiers of the component pyramid), project results are continuously fed back into the project plan. This in turn is replanned as required, appropriate changes are made to the project-specific process, and the project continues until complete. Second is the improvement of the overall AD process based upon the collective results of actual projects. As we shall see, this is at the heart of all total quality management (TQM) programs. Predictability and repeatability are possible through this system of feedback and continuous process improvement.

Although formal education is required for building the skills needed to reach proficiency as a software developer, the methodology facilitates on-the-job training by guiding the developers through the activities and deliverables, helping them to see the progression of events, while alleviating extraneous work. The methodology should provide guidance with tool usage, with organizational standards (such as common user access and documentation standards), and with deliverable samples (for example, a developer pulls down a window and gets immediate help for any step in the process). This kind of guidance or on-line help reinforces learning, as it enables the developers to see a practical application of the new information while doing the work, making it more meaningful and relevant in the context of their job. One example of an on-line help capability is a hypertext application, such as SAA* (Systems Application Architecture*) BookManager*.[3] A hypertext application typically provides text in a display-oriented fashion, providing additional navigation features such as links and searches through the document.

Requirements for process management

The need to implement an AD solution that includes process management is derived from several sources. First and foremost are the direct requirements from development organizations that are seeking a new approach to AD to help them to re-

Figure 1 AD solution components and feedback loops



alize increased productivity and product quality. Second, we consider the environment within which we find ourselves in the 1990s, where *total quality management* is having a major influence on the products we create and the processes we use to produce them. Intrinsic to the quality of a process is its usability, and we consider the implications of human factors in process management.

Requirements from development organizations. The authors have gained their experience in this area by previous work in the field, particularly with the IBM process management tool Application Development Project Support (ADPS).[4] The feedback from customers using ADPS, along with several reviews and analyses of the product, validated the general approach taken and brought out requirements for future extensions of functions originally offered by the ADPS products. Prototyping work in this area is progressing.

In 1987, the Application Development Joint Project (ADJP) was formed by the International User Group Council (IUGC). The ADJP members represented Australasian SHARE/GUIDE, GUIDE International (U.S.A.), G.U.I.D.E. (Europe), SHARE Eu-

ropean Association (SEAS), and SHARE (U.S.A.) in both the commercial and scientific and engineering fields. This group's mission was to advise IBM of requirements for functions that would result in a "significant increase in application development productivity."[5] The resulting *Application Development Productivity Strategy* paper represented a compendium of needs from over 15 000 member organizations worldwide. The paper pointed to an application development environment (ADE) as the way for IBM to "enable its customers to produce efficient, high-quality applications with an order of magnitude improvement in application development productivity."[5] One of the key features of the ADE is the automation of process management that would provide many functions, including: automating work flow, invoking tools, triggering tasks at appropriate times, reporting process status, and describing states and state transitions of deliverables. Other desired features were intertask communication, a tool interface, and a method to validate the work produced with respect to correctness, or an indication of which part of the process was responsible for the error.

This set of requirements developed by the ADJP was further expanded by the subsequent work of the Application Development Management Project (ADMP).[6] This follow-on IUGC project included a focus on cross life-cycle services (those services that affect more than one area of AD), including: project management, process management, and metric requirements, and emphasized the need for an information model to contain business and project-related information. Their work emphasized that the current processes are poorly defined, difficult to manage (enforce, automate, and measure), and do not allow for formal change control to be implemented. There was also an acknowledgment that the present situation left their AD organizations unable to deliver products of consistent quality and cost. One of the points made was that automation was required to do effective process definition and measurement collection, and further, that links to project management were necessary. Their work discussed the need for project and process synergy and proposed that enabling these cross life-cycle facilities was the only way to achieve the order of magnitude productivity improvement.[6]

The environment. In the 1990s, we find ourselves in a worldwide movement to improve quality in every industry. Application development is not immune to the problems that poor quality products cause. To better understand the environment, we first need to differentiate between the two areas of the quality emphasis.

We are concerned about product quality and process quality.

The focus on quality is twofold; on the one hand, we are concerned about *product quality*—the end result or deliverable of a series of activities. Product quality is typically thought of in terms of numbers of defects per production run, mean time to failure, or other tangible characteristics that are readily measurable. Product quality has been a focus of manufacturing for years, and in application development we have successfully used various methods of inspecting for product quality. These methods include peer reviews and code walk-throughs, in which the objective is to validate the logical flow and determine if the code will perform the desired function. Application quality is typically measured in terms of defects, response time, and end-user satisfaction.

Contrasted to product quality, but complementary to it, is the quality of the process that is employed to produce the deliverable. *Process quality* has to do with the ease of navigation through the required activities, the ease of executing selected activities, and the predictability and reliability of the end result, or product. Traditionally, we have focused on product quality. Only recently has the importance of the process by which we do our work been acknowledged as a major contributor to the quality of the product. The emphasis on quality, of both product and process, is of worldwide concern, as shown in the requirements from ADJP/ADMP.[5,6]

ISO 9000. The International Organization for Standardization (ISO) has published a set of standards for developing processes to establish reliable contracts for the delivery of quality products

between supplier and purchaser. Since developing software was seen as a problem with very specific properties that differed from manufacturing hardware products, ISO has issued a specific standard ISO 9000-3 on "Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software."[7]

This standard requires that suppliers establish a *quality system* for the life-cycle activities of a product, beginning prior to the development life cycle, at the contract stage. Here, the purchaser identifies requirements and product parameters, and the supplier makes contractual commitments to deliver a product that conforms to these requirements. The overall objective of the quality system is to ensure that both supplier and purchaser have the capability to meet their contractual obligations. The ISO 9000 standards require the supplier to submit a well-defined description of their quality system for certification.

A quality system, as defined by ISO, consists of many elements that must be present and auditable, including: product identification and traceability, inspection and test status, special controls for nonconforming products, documentation and records, training, and use of statistical techniques for quality measurements. Automation is not a requirement for a quality system. In the ISO 9000 sense, *system* (typically thought of in computer terms) refers more to a comprehensive program with feedback loops for error correction and improvement.[7]

In addition, ISO 9000 suggests what major functions a quality system for the life cycle of software development should support and requires a process with certain properties. The process of the quality system must be well documented, must analyze causes of nonconforming products, and analyze and improve all processes and their components. The term *quality* is used here in the sense of conformance to specifications in general. The standard also requires well-defined management authority and responsibility for ensuring that the requirements of ISO 9000, as expressed in the certified process, are implemented and maintained throughout a project.

The components of a life-cycle quality system recommended by ISO 9000-3 represent the known aspects of good software engineering techniques. The process definition must be available for independent, external inspection by purchasers, thus requiring a particular emphasis on complete, well-structured and accessible documentation for all aspects of the quality system. While executing, the process must be monitored and recorded so that adherence to the certified process can be audited throughout the project. The documentation and audit files must be retained for a defined period after conclusion of the project. The mode of implementation of a quality system is left open to the supplier. But many aspects of documentation, recording, and auditability will strongly suggest computer support for a quality system.

Malcolm Baldrige Award. The Malcolm Baldrige Award was developed as a result of the Malcolm Baldrige National Quality Improvement Act of 1987. This act called for a national quality award with the intent of increasing the quality of U.S. companies, as part of a national quality improvement program.[8] The act established award criteria with which companies could assess their quality improvement efforts. The Malcolm Baldrige Award represents the highest rating for quality attainable in the United States. The criteria for assessing an enterprise according to Baldrige standards emphasize a number of key process concepts.

The award system has seven major categories that are separately assessed, contributing to a total attainable point value of 1000 points (in all categories combined). These are:

1. Leadership (90)
2. Information and Analysis (80)
3. Strategic Quality Planning (60)
4. Human Resource Development and Management (150)
5. Management of Process Quality (140)
6. Quality and Operational Results (180)
7. Customer Focus and Satisfaction (300)

The fifth category, Management of Process Quality, includes: process quality, continuous improvement of processes, quality assessment, and documentation, among others. Each of these has a corresponding point value that contributes to the overall 140 points for the category. The assessment looks for any approach to manage and control the process that will yield verifiable, repeatable results that meet customer needs.[8] Although a process is explicitly assessed in this category, most other categories focus on management systems and processes as well.

The focus on information quality is pervasive throughout the Baldrige assessment and accounts for approximately 30 percent of the total score. An example of an individual category's emphasis is that of database quality, within the Information and Analysis category. This focus has resulted in the category point value being increased from 70 to 80 points. "The database that is used to measure the progress of the organization in meeting its goals is one of the most important factors in the quality system."[9] In fact, the point value for collecting and using company data has doubled, from 20 to 40 points. This emphasis on metric collection and analysis places additional requirements on an automated quality system.

Within IBM, which uses the Baldrige model as the basis for its market-driven quality program, the Management of Process Quality category emphasizes the importance of the definition, management, and improvement of processes. Assessment is based upon three factors: approach, deployment, and results, with a percentage of the total possible points given for various stages of attainment.

Deming philosophy. W. Edwards Deming is an international consultant and a member of the National Academy of Engineering. Deming's management philosophy was adopted by Japan in 1950 and is credited with greatly influencing the Japanese transformation into a world-class industrial nation. In 1980, the American Statistical Association established the annual Deming Prize for improvement in the areas of quality and productivity.

The Deming approach[10] focuses on statistical process control and process measurement, resulting in quality being built into a product, rather than requiring postproduction inspections for defects and errors. Deming's 14-point program for transforming the industry, based on improved quality and productivity, addresses areas such as minimizing cost, instituting training on the job, leadership, and eliminating production quotas.

Deming, in his discussions about manufacturing processes, asserted that the definition, scrutiny, and control of the process (process management) would provide opportunities for process improvement. Process improvement would, in turn, lead to manufacturing improvement because it forces people to look at the design and manner in which something is produced. This scrutiny provided insight into the individual tasks performed and made evident areas where greater efficiency could be achieved, or error-prone activities upstream could be changed so that a higher-quality product resulted the next time through the process. Deming felt that to perform reviews and inspections was simply not sufficient. Even though these approaches were excellent error-detection methods, they still occurred *after* the error had been made, and the final product of that stage then required correction.

The concept of product improvement is predicated upon first being able to obtain predictable and repeatable results in development efforts. Improvement can only be accomplished by a systematic, disciplined approach to production, whether it be in manufacturing or in application development. For example, the term *information engineering* implies a more structured and rigorous method of application design and production. This transformation of application development from an art form to a more defined set of activities is an indication of the recognition that process management is required in order to further mature the field. The additional requirements of ISO 9000, Baldrige, and Deming justify the need to define, measure, and improve the process. A discussion of automated process management systems brings to bear additional factors that must be considered during development of such a system.

Other requirements on process management automation. Paramount to the acceptance and use of a tool or system are its human factors considerations. The most efficient and effective tool will quickly be cast aside if it is not perceived as useful and easy to use by its intended audience.[11] The value of ease of use and good human factors can be measured by productivity gains, facilitating timely product delivery, satisfaction of developers with their jobs, reduced training costs, and protection of investment in tools. Ease of use includes providing a high degree of intuitiveness in operation and navigation, so that when presented with an activity, the users (in this case the developers) can easily connect the function to be performed with the relevant deliverable and then accomplish the activity in the most straightforward manner. We have worked with tools that neglected this premise and found it interesting that users would take a more circuitous route (bypass-

ing a tool) to accomplish a task, rather than use a tool that they perceived as cumbersome.

In addition to ease-of-use considerations, if users should need assistance, an on-line, context-sensitive help facility would be valuable. During familiarization with a new tool, a primer and reference guide in document format would enable a user to browse through the material while using the tool. A good primer helps the novice user achieve proficiency quickly and thus feel greater satisfaction in using the tool. Ease of use in a multitool environment includes passing the output from one tool to the subsequent tool downstream in the process.

Paramount to the acceptance and use of a tool or system are its human factors considerations.

Proficiency levels will be achieved in varying amounts of time due to differences in usage. Some may use a given tool only occasionally, requiring more time to relearn functions with each use. Those who use a tool regularly will become familiar with the basics quickly and will likely learn the more subtle nuances the tool has to offer. These differences in usage and proficiency require a tool to be easy to learn, but also to be usable in both a basic and a more sophisticated, fast-path manner. Failure to provide these two perspectives can result in decreased usage by experienced people, simply because it now takes them too long to perform a function, and the tool becomes an impediment to productivity. These considerations present an additional challenge to the developer of any automated system but pose particular challenges for the automation of a process management tool. The tool must provide for both occasional and frequent users and must be friendly to the novice, while not becoming an impediment to those who are very experienced.

Other sources of expertise and knowledge. Watts S. Humphrey, in his book *Managing the Software Process*,[12] provides a general analysis of the challenges and approaches in application development, with the goal of improving deliverable quality and process productivity. In his introduction of the concept of process maturity levels, Humphrey distinguishes five levels of maturity for software development processes and also identifies what is needed to promote a process to the next higher level. The five levels are listed in Table 1. Using such categories, AD organizations can assess themselves and determine what is needed for them to continually mature. The goal, of course, is to attain the optimized level, level 5.

Humphrey's concepts guide one through the functions that will have to be implemented to achieve certain characteristics of the process. As he explains, a quality process is a prerequisite for the development of quality products. Implementation can range from manual disciplines to a maximum of computer support. However, as we shall see when we approach higher levels of maturity, the need for computer support will become progressively more apparent.

It should be noted that the discussion of optimizing processes is occurring while the industry is still on a learning curve about processes and their attributes and properties. Humphrey's concept of a continuously optimizing process clearly needs automated support in order to be successful. The implications of how process behavior reflects the behavior of the people that are an integral part of the whole development system is included in this learning curve, as are their acceptance and reaction to the process.

Requirements summary. In summary, the requirements for a defined and disciplined approach to application development are being voiced from several areas. The true challenge for automating process management comes not only from being able to satisfy these requirements, but also from doing so in a manner that leaves the developer using the process with an increased sense of satisfaction from the job. The proposed process management solution should enable creativity and innovation and not make the development process a rote and mundane job. Productivity increases and higher product quality are tightly linked with job satisfaction; thus a process management approach must be ever mindful of its intended user.

Table 1 The process maturity levels

Level	Characteristics	Process Enhancement
Initial	Ad hoc, on occasion even chaotic. No formalized procedures or project plans. No management understanding of relevance of key process issues.	Establish project management system, including quality assurance and change control.
Repeatable	Can repeat tasks mastered in previous project. Process dependent on accumulated experience of individuals. New tools and methods cannot be incorporated without risk, due to lack of process framework. Inability to capitalize on innovative approaches.	Establish a process group and a software development process architecture or development life cycle (methodology) that describes all activities in the process.
Defined	The process is established and well understood. The process can be applied to most normal and crisis situations. The process does not encompass a sufficient collection of data about the process to analyze efficiency.	Establish process measurements to identify quality and cost parameters. Provide sufficient resource to monitor process data to assess relative product quality and advise management to take action when targets are not met.
Managed	The process is measured and controlled, measurements are meaningful and well defined. Measurements are kept in a process database.	Demonstrate that measurements are accurately and correctly gathered, aggregated, and evaluated. Due to the volume and type of data involved, Humphrey felt that this would require automated support.
Optimized	Focus is on systematic process improvement	The data gathered about the process must be fully utilized toward identifying the places in the process that can be improved. Product quality becomes a natural by-product of the high-quality process.

Note: Based on the Software Maturity Model by Watts S. Humphrey[12]

Automating process management

The terminology in the area covered by this paper has not yet stabilized. In particular, the semantics used in this paper for the terms process management and project management need to be defined.

Process management is a term with varying definitions in the literature. The broader meaning, and the one used in this paper, encompasses all work of the development team (in any role) throughout a project, including items traditionally thought of as components of project management, metrics, quality management, and others. This larger scope more accurately reflects the concepts of *process maturity*[12] and the ISO 9000 standards.[7] Historically, process management has been basically concerned with the application-oriented deliverables in a project (e.g., data flow diagrams and modules), with their status (e.g., in progress, completed), and with their related activities. This narrower view of process management does not consider management aspects, such as who does which task when (traditionally thought of as a project management activity), or who controls the correct execution of the process (typically not a consideration in development projects).

Project management, in the traditional sense, manages the details and constraints within a project: who has to perform which activity in the context of a task, and when. By use of project management techniques, task plans for a project are established, including estimates for resources and effort needed, schedules to be met, and quality to be achieved. The aspects of planning, tracking, measurement, and recording are facets of project management.

Separation of such facets into independent management domains (process or project) is the traditional response to the lack of an integrated development environment with respect to the definition of its process. This integration from a process perspective will enable us to achieve

the goals of producing higher-quality deliverables and increasing productivity by having development teams consider their work efforts from a broader perspective, including quality assurance, tools and techniques, and training.

Historically the procedures in application development (the methodology) were implemented by sets of books and standards that were to be read and used by the developers. One of the earlier methodologies of this type was developed in IBM Germany under the name "Verfahrenstechnik,"[13] which advised IBM customers how to approach and master the challenge of developing data processing applications. These early methodologies were exclusively documentation-based. The requirement to aid the developers with more advanced tools and services always existed. Only when improved price-performance hardware and user-friendly interfaces to computers via displays became economically feasible did the first comprehensive computer-based support environments for the AD process become available. An early example is the IBM Program Offering VIDOC,[14] developed in cooperation between IBM Germany and IBM customers, which provided computer-based support for the above-mentioned "Verfahrenstechnik" and integrated a set of AD tools available at the time under the process. Based on VIDOC concepts, the set of IBM Program Products Application Development Project Support (ADPS)[4] was developed and became available in 1987.

Although ADPS assisted in the management of development projects and introduced the concepts of process management by invoking tools, it left the developer with a certain amount of manual day-to-day routine activities and workplace management tasks. In this sense, it actually supported process management more than project management. To gain the additional benefits from a process management system that can fulfill the requirements of Humphrey's metric collection and analysis, and support project management activities as well as various other quality requirements, calls for a more highly automated system.

A quality system for software development can involve computer support to a high degree. Ever-increasing levels of product quality and process productivity will demand even higher degrees of computer support by automated processes with integrated tools. We frequently talk about the *automation* of software development, a term to be used with caution because it can lead to a misinterpretation of the role of developers in a fully automated development environment. Only routine activities can be automated. The work remaining for the developer may require higher skill levels, enabling the developer to concentrate on the creative aspects of development. Work environments can be defined for specific user roles, providing guidance to activities and deliverables eligible for work, as well as extensive help support, automated metric collection, and fully automatic tool initialization and invocation. These work environments can then be automated to enable developers to achieve higher productivity and quality of work by reducing the probability for errors and by reducing the amount of administrative manual work.

For a process supporting the concepts of a specific application development methodology, we must incorporate tools and tool functions in a way that satisfies all requirements of the methodology: the right set of deliverables and of corresponding tool functions for activities in the process, the ability to integrate them into a comprehensive seamless process based on the methodology, and support for the pragmatic requirements of process maturity, the ISO 9000 standards, Malcolm Baldrige, or similar quality and productivity approaches. Many of the aspects of process definition and process implementation have already been discussed in the literature.[15–17]

Components of a process management system. There are several components of a process management system that are needed to support the requirements for mature, certifiable processes for multideveloper projects. For every role in a project (e.g., process planning, technical development, quality assurance, resource management), developers can be offered customized workplaces. A workplace customized for a role enables the developer to concentrate on the specific activities and objects relevant to that role. We can provide developers with *units of work* they can perform in a role in an undisturbed and uninterrupted way for substantial periods of time (e.g., several days or weeks). Developers may have a choice between several units of work that, at a given point in time, could be assigned to them, either in one single role or in multiple roles within a project.

When analyzing the software development process representations with conventional means of functional decomposition and data analysis, one can observe that the functions in these processes depend, to a large extent, on the existence of *deliverables* and *deliverable states*. As a practical example for these aspects, consider a deliverable "source-module" that could have a status of "in-work" or "code-complete," where the function "inspect-source-module" would, according to the rules of the specific example, only be executable if the deliverable has reached the status "code-complete," as indicated by the developer.

Considering all deliverables, deliverable status, and activities incorporated in a specific methodology, we can design a generic process model for all instances where the methodology would be used in specific projects. In the process model we define activity and deliverable *types*, whereas specific *instances* will be defined in the execution of the process for a specific project.

When automating process support, various options are available for presenting the units of work and their contained activities to a developer, who is performing one or more roles in a project. A system can be designed to present a developer with a single activity, or with many activities in units of work. A system could also present a developer with a list of deliverables that are eligible for work, according to work assignments. The amount of work presented to a developer may be restricted to those activities or deliverables that, according to the progress of the project, are eligible for work by roles or for the whole project. Work can also be presented in a broader view of already completed and future work. Choosing among the available alternatives, the designer or administrator of the development process will have to consider a balance between a high degree of enforced control versus a higher amount of user choice in selecting work, following informal disciplines.

A methodology-based application development process essentially supports and coordinates the work of designers, coders, planners, administrators, and managers of the development team. Most plans, schedules, and cost assumptions are centered around the work of people involved in the process. Thus it is most important to understand how the work of people is planned, coordinated, and supported. For this purpose we assume that *elementary activities* are executed by the members of a project team in their assigned roles. Today, we find that most of these elementary activities in a process are supported in a variety of CASE tools. Elementary activities in most cases depend upon *input deliverables* and provide *output deliverables* in defined states.

Process catalog of activity types and deliverable types. Earlier work of the authors with ADPS customers and subsequent prototyping work have verified the need for a catalog of activity types and deliverable types, as well as for an inventory of deliverables throughout the project (these items appear in the ADPS products under different terminology). A process model supporting a selected software development methodology employs a set of elementary activity types, requiring defined input and output deliverable types. The *instances* of activities and deliverables will only be defined when the plan for a concrete project is established. It is convenient for the developer of a process model to be able to use a catalog of all activity types and deliverable types that play a role in the selected methodology.

For each activity type, the catalog also defines the types of input and output deliverables and their associated states as required for input, and delivered as output. Elementary activity types in most cases are supported by tools or tool functions. It is advisable to define activity types in a generic way and to implement a catalog of activity types such that the association of tools and tool functions can be changed throughout execution of a process model in a project. This facilitates tool upgrades for new tool releases or replacements, and will enable reuse of catalog items used in defining a process model for execution in multiple projects where different sets of tools may be employed.

Deliverables inventory. Throughout a quality system, the status of individual deliverables and of collections of deliverables (lists, sets) plays an important role in the navigation decisions in the control flow, whether done manually or automatically. Individual deliverables and deliverable collections are also the objects to which numerous measured and assessed data are attributed. Automated processes require the concept of an inventory where all such information can be maintained and retrieved. Such an inventory does not necessarily contain the actual data of the de-

liverables, like the source code of a module or the text file of a document, but may facilitate access to these deliverable data, in addition to maintaining the navigation-relevant and quality- and productivity-related attributes. Auditability requirements of the ISO 9000 standards require that much of this information remains available for several years after a project has terminated.

Automated processes require the concept of an inventory where information can be maintained and retrieved.

Today, subsets of these data may be found in databases, libraries, repositories, and encyclopedias, embedded in, or used by, a variety of tools. It is a particular challenge to provide an integration concept for all of these existing partial solutions into one single and consistent framework, enabling access to data and attributes of all deliverables in an automated, integrated, and seamless way.

Tools and functions available for activities. Activity types defined in the catalog are implemented by tools and tool functions. Any realistic approach toward a *computer-aided quality system* must make the best use of existing investments in CASE tools, standards, and education. Only an *evolutionary* process for upgrading development environments toward a quality system is economically feasible. Many application development tools exist today, supporting functions from enterprise analysis, through requirement analysis, design, implementation, verification, installation, and maintenance. It is not reasonable to assume or require that an organization replace its current tool base in order to realize the benefits of automated process support.

The tools and tool functions integrated into the process for executing specific activities will be the environment where developers spend most of their working hours, and achieve, or fail to achieve, the projected goals of productivity and quality. Most of the relevant metrics to be recorded for a process and used for its evaluation and improvement must originate from this environment, either collected automatically by the tools or provided by the developers.[2] The integration of tools into the process and their automatic invocation during process execution has been identified as one of the most important requirements for process automation.[5]

Suitability of tools and functions for integration. Guidelines will have to be defined for an improved generation of tools that will support new requirements and that will evolve over time. It is conceivable that the requirements for full integration of all aspects of a quality system pose substantial requirements beyond the capabilities of many of today's tools. Enhancements to existing tools or the development of new tools may be needed.

Tools could be imbedded in tool shells that could present a consistent interface to the user, provide interface adaptations for invocation and parameter passing, and provide access to additional help information for the process context. These shells could also support dialogs for situations in which developers have to provide additional information required by the process, but not provided by some tools.

Some aspects to be considered in determining the suitability of tools and tool functions are:

- Adequacy of the function set according to requirements
- Ability to invoke tools with parameters for specific functions
- Ability to accept information about input deliverables
- Ability to deliver information about output deliverables
- Provision for measurement data about output deliverables
- Provision for measurement data about function execution
- Provision to return information about function completion
- Ability to provide text output for project documentation

Seamlessness of operation. A quality system assembled from components that exist today has to

be composed of elements from various sources: existing application development, quality assessment, and process administration tools with their own dialogs and help information. Methodologies need to make the right selections from these sets of tools for complete support through all steps in an application life cycle. Developers should be supported with methodology-specific help and guidance, providing workplaces that support their roles, and with documentation that follows the process, recording progress at various points. Throughout the process, a process navigator should provide the individual workplaces with references to selectable activities and deliverables. This complex system would appear to the developer as one single, consistent environment, providing consistent user interfaces and "look and feel." The higher the degree of automation in an integrated development environment, the more visible the faults in seamlessness will become. Some of the aspects required for seamless operation are:

- Consistent dialog standards as documented for Common User Access* (CUA*)[18]
- A consistent way of exchanging information between developers
- Smooth flow of deliverables through the process activities
- Comprehensive coverage of all functions required by a methodology
- Overall concepts for measurement and validation across all phases of a project

Migration and convergence. A quality system will have to integrate a wide variety of tools that are in use today and will also have to import existing deliverable sets from existing software projects into the new environment. A critical factor in moving to a quality system will be the transparency with which a migration can be made, and how much initial effort may be needed to migrate an existing application portfolio. Migration and conversion tools will be needed to logically link or physically transport existing deliverables into the context of the quality system.

Tools continuing to use their current deliverables environment must provide dynamic exchange of information about their activities and must be able to execute requests to perform functions on specific deliverables for which they hold the deliverable data.

The process from the planner's perspective. A model for the flow of activity execution in a project can be designed from the catalog of activity types, deliverable types, and deliverable states required for activity execution. Such a *process model* can represent a schema for many projects that follow the same methodology and use the same, or an equivalent, set of tools. A convenient way to present process models is to represent them as control flow graphs.

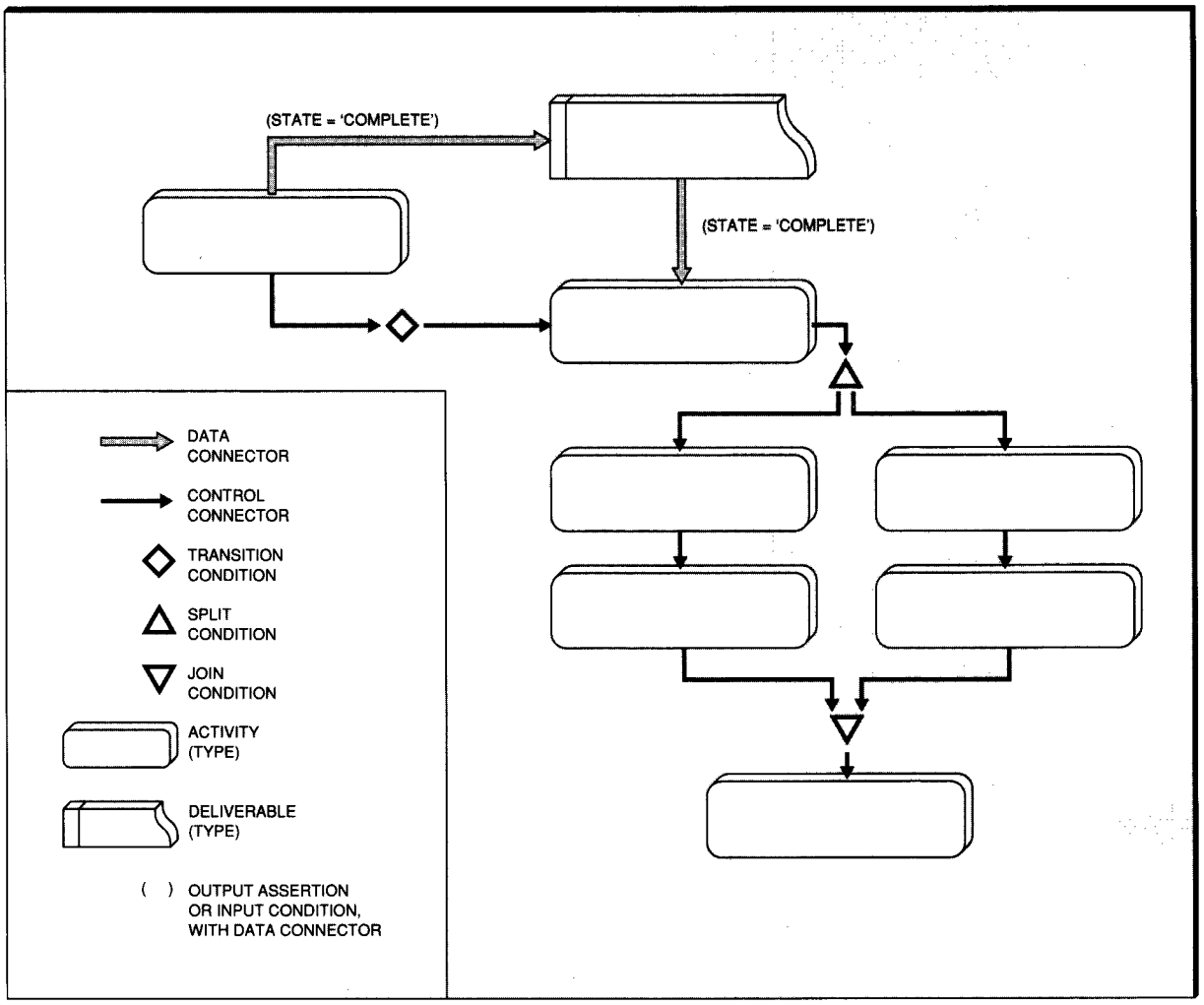
Navigation logic. The sequence of activity execution in a process plan is controlled by navigation conditions. Figure 2 shows a sample section out of such a network of activities. An activity can be modeled, showing a successor activity dependent on the truth value of a *transition condition* in a *control connector* between the activities. A completed activity may also be followed by more than one successor activity, according to the truth values for the outgoing control connectors in a *split condition*. Several activities may also have one common successor activity, where a *join condition* determines the conditions under which the successor activity becomes enabled for execution.

The control flow connectors shown in Figure 2 carry values to control the flow of execution logic. The control values carried by a connector leaving an activity or a navigation condition can be unevaluated, true, or false, depending upon the status of its execution.

Navigation conditions can be based on such flow control values and on values obtained from deliverable status. Thus, the flow control value leaving a navigation condition can depend on the flow control value carried by the control connectors entering the condition and on the status values of deliverables referenced by this condition.

When designing navigation conditions, reference to deliverables entering the control flow may be shown by data connectors. Since these deliverables are referenced by name, showing data connectors enhances the graphical representation of the conditions. Optionally, data connectors may also be used to denote the status of an output deliverable from an activity or the status of a deliverable that is input to an activity. This information (associated with data connectors) serves only as commentary. The actual status has to be checked in a navigation condition, and the status

Figure 2 Example of a control flow network



for the output from an activity must be set by the activity.

Many navigation conditions will not depend on deliverables produced by predecessor activities, but rather on other data obtained from the project environment. Such data could come from trigger events, like "end of week," or "calendar date," or from the change of certain deliverable attributes that are utilized as triggers. Navigation decisions in process execution could be evaluated manually by a developer investigating the execution status of activities and the status of deliverables. Navigation decisions based on documented rules and disciplines would then be made by the developer (in a manual process) or automatically by a navigation engine (in an automated process management system), based on the conditions in a control flow network.

Abnormal navigation. We have to expect cases in which the flow of execution as it has been modeled for a project cannot be done as planned. When an error is detected, it is necessary to determine which of the previous activities caused the error. As a result, parts of an already executed process may have to be reopened and re-executed. It will remain the job of an expert planner

or process analyst to identify which deliverable status values have to be reset, which of the already executed activities in a project have to be reopened and re-executed, and even what new subprocesses may have to be defined and attached to the original process in order to repair the impact of defects. This case shows how important it is to avoid errors and, if they occur, to detect them as early as possible so that the planned process can incorporate corrective actions.

Advance navigation. In many cases in a real project, developers may want to start activities even if the required input deliverables have not yet reached the required completion status. This type of *work ahead* navigation can be precisely defined, identifying the possibility of starting an activity even though the input deliverable is not yet ready. A sequence of such *loosely conditioned activities* must be synchronized at a predefined point, verifying that all deliverables in the sequence have reached the status required for the start of each activity before the synchronization point. In a less rigorously controlled or less automated process such decisions may be left to the developer, relying on the discipline of the developer to obey rules that may be documented but are not enforced.

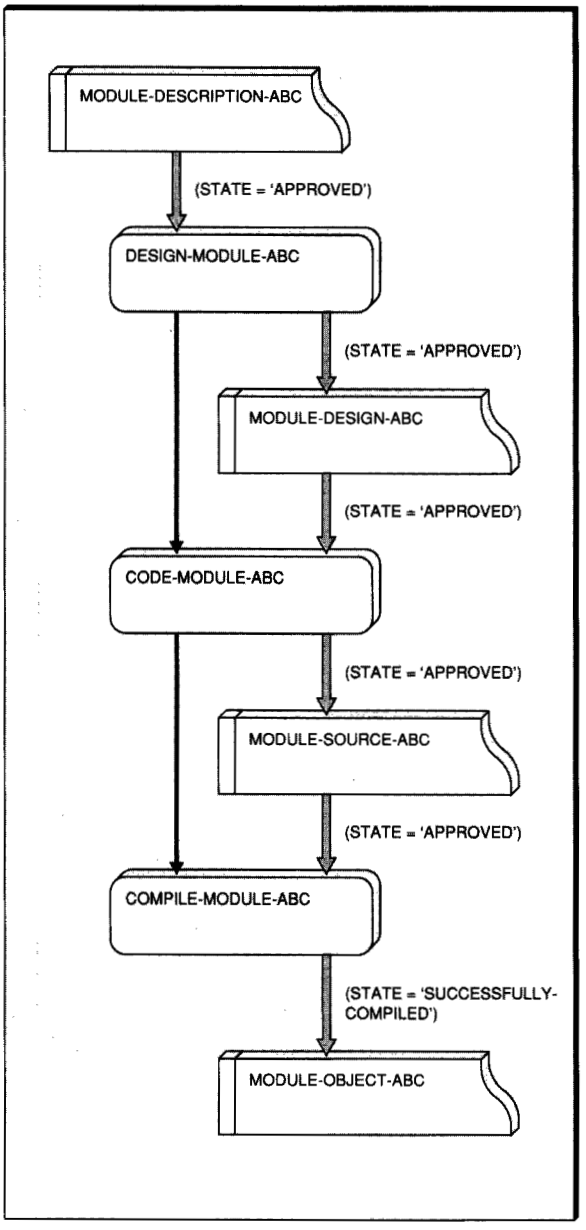
Transformations and state transitions of deliverables. Before one can design a complete model of a development process, assuming a specific methodology, one has to understand the detailed role of deliverables in the process. Many deliverables undergo state transitions and are subject to transformations. Let us, as a practical example, talk about a specific function of an application to be implemented as a module. Let us further assume that we have already obtained a deliverable of type "module-description" for the module ABC as the result of an analysis phase. This module-description will now be transformed into a new deliverable of type "module-design" for the same module ABC by an activity of the type "design-module." This will be executed with the input "module-description-ABC," the specific description for the module. Further transformations will, for the specific example, create new deliverables, "module-source-ABC" and "module-object-ABC." Figure 3 explains this sequence of activities, transforming the conceptual object "Module ABC," as it may be reflected in the analysis and high-level design for the application, into the concrete deliverables of the sequence of transformations, each qualified by the identifier ABC from the conceptual object.

In this example, the first part of the name of the deliverables is formed by the type of the deliverables, whereas the second part is the name of the conceptual object. Thus, we observe a sequence of deliverables of different types being created for the same conceptual object name. Such transformation sequences are common in development processes where conceptual objects are carried through multiple steps of refinement. Each individual step involves an activity to be performed by a person in the developer role. Each of these steps of refinement may involve both the use of tools and creative work by the developer.

Figure 3 also shows the status value of deliverables, either as output from activities or input to activities. A specific deliverable in our example also passes through a sequence of states, as shown in Figure 4. The deliverable "module-source-ABC" goes through the status values "in-work," "completed," "inspected," and "approved." This transition of states is again performed by a sequence of elementary activities within the subprocess activity "code-module-ABC" (from Figure 3). Thus, the state of an object changes from an initial state to a final state, as defined by the development methodology.

Process planning for a project. Once the enterprise has established the model that is applicable for a specific project, a person in the role of process planner will have to produce the process plan, making decisions about the definition of work units as aggregations of activities to be assigned to the individuals in the development team. (It should be noted that new roles may emerge over time, such as that of *process planner*, which may evolve from the traditional role of project manager, or may become an additional, complementary role. Role definition and delineation will most likely occur at an organization or project level.) Process planning dynamically continues throughout the progress of the project. In this planning activity the constraints of limited resources may require the planner to employ human resource and time management functions to determine the most effective use of the development team to achieve the given targets. Such planning activities themselves are part of the process plan that has to be executed at defined project checkpoints or whenever one of the critical

Figure 3 Transformation sequence of activities



parameters monitored throughout the project deviates from the target plan.

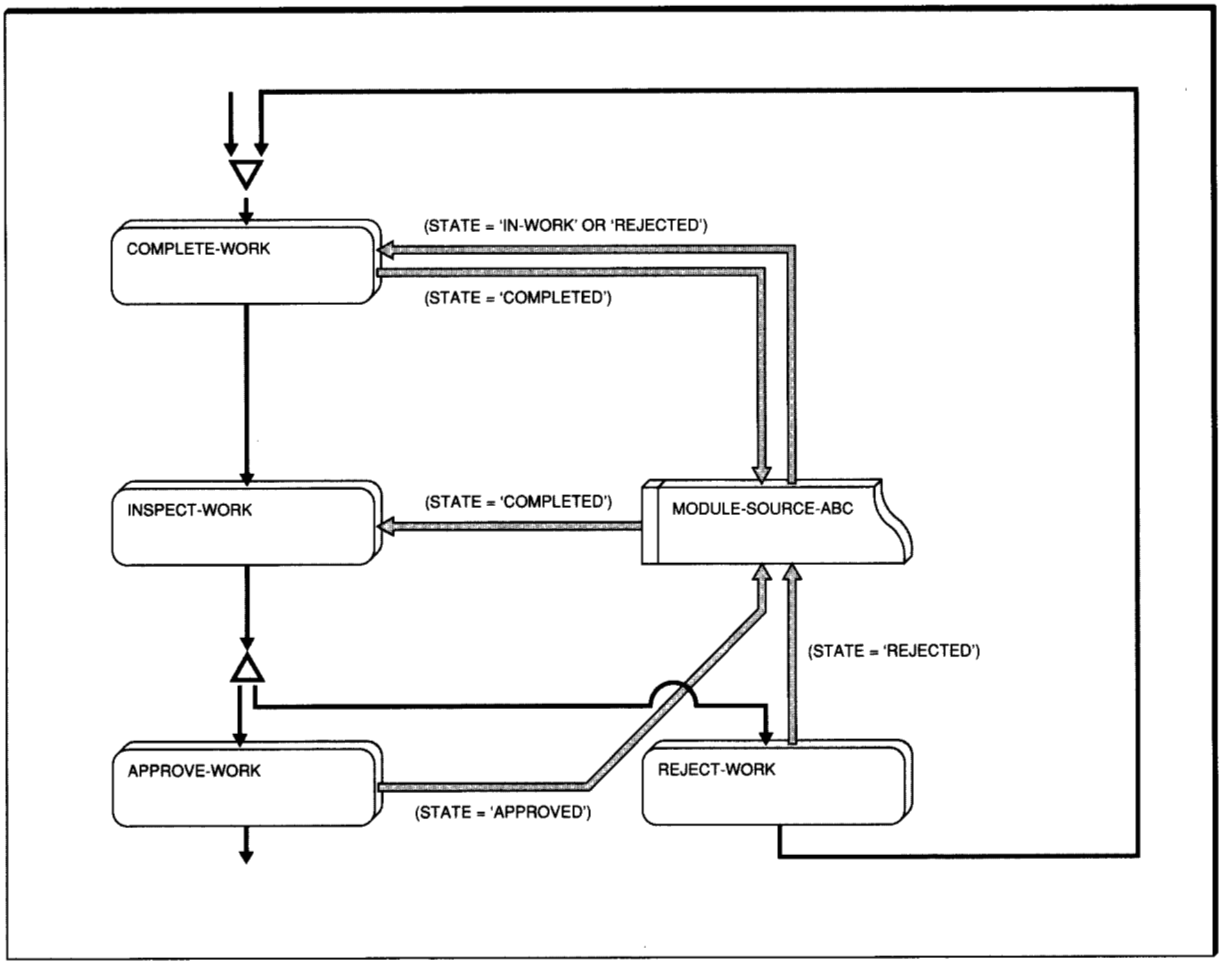
For a specific project, a process plan can be derived from a process model. In such a process plan, the specific instances of deliverables of a given type for the project will be defined as shown in Figure 5. In the various places in a process model where deliverable types are involved, the names for deliverables in the project have to be provided by a planner or designer. Thus the process model will be populated with the instances of deliverables and activities for the specific project.

The refinement of a process model into a process plan will, in most projects, be a dynamic planning process, progressing to the planning of instances of deliverables and activities in steps when parts of the project have already been executed. Both process models and the corresponding process plans can be represented by a control flow graph, as shown in the example in Figure 5. The stepwise refinement of the process plan will have to take into account the fact that the exact number of instances of a deliverable of the project, and the corresponding instances of activities to be executed, will only become known as the project progresses.

As part of the planning activity, at the earliest logically feasible point, the *estimates* or *projections* for the *effort* needed to perform an activity, and for the *size* of the deliverable to be created, have to be provided by a planner or by a tool and have to be recorded. Such data will be subject to later revision and to eventual recording of the final values when completed. The planner, acting in the role of project manager, will consolidate the logical process planning with the given resource constraints and with the relevant business factors for the project. This consolidation will continue throughout the project.

In this task of process planning, the accumulated knowledge from previous projects and from skilled team members should be accessible and should be used. Numerous factors come into play in planning that are difficult to formalize and optimize, because they involve both judgment of human behavior and a strategy of risk assessment and risk taking. Measuring the performance of people touches on an area of legal consideration: depending on the laws of the country where a project is performed and on the practice of the industry, it may be illegal or unusual to record detailed information about the performance of individuals. This situation makes estimating and planning more difficult and less transparent. The art of estimating and projecting will always involve a good amount of human judgment and represents a key critical function in software development projects.[19]

Figure 4 Sequence of activities changing deliverable status



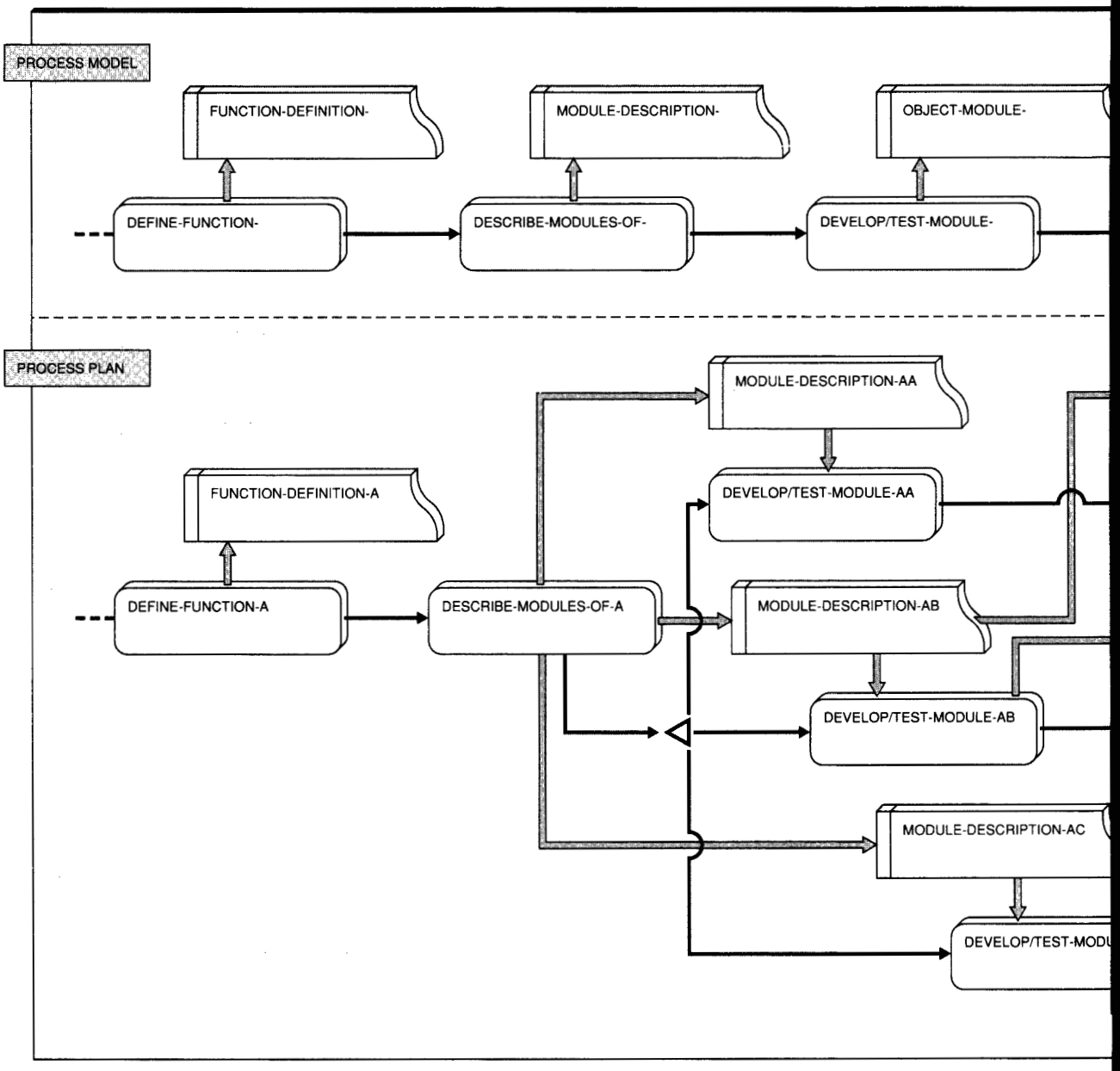
The process from the developer's perspective. Up to now, we have only described the logical conditions under which the navigation of activity execution progresses. Activities in a process can be defined as manual, requiring a developer to execute the activity, or as automatic, requiring a processor to execute code defined for the activity.

In defining the process plan, it must be established who is to perform each activity before its execution—a person (acting in a specific role) or a processor (for automatic activities). In the process model, an activity type can be defined such that it is logically related to the role intended to execute that activity.

Role-oriented work guidance. During process execution, developers will be assigned to roles so that specific responsibilities, according to the role requirements of the activity and the roles assigned to developers, can be identified. As activities become eligible for execution, the responsible developers will be notified by an automated navigator. In a similar way, automatic activities will be placed on work queues for the appropriate processors.

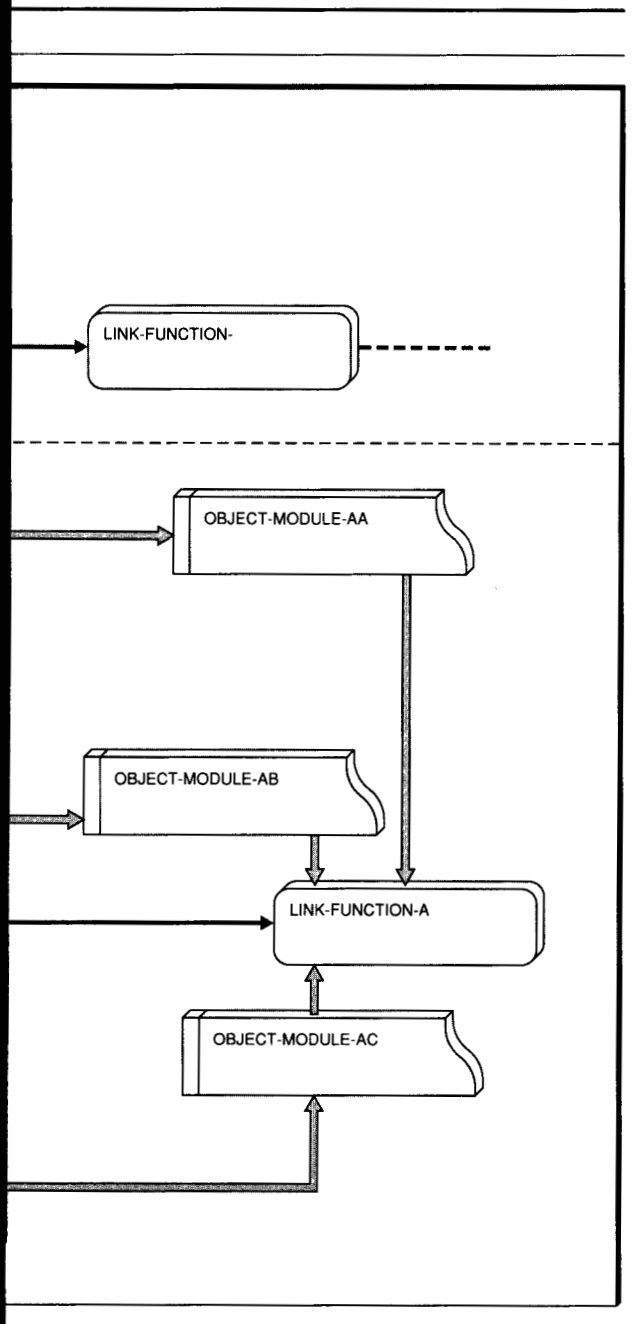
Earlier, we introduced the concept of units of work that can be performed in a project by one developer in an uninterrupted way. Such units of work can be defined as part of the process plan,

Figure 5 Process model and process plan



identifying which sequence of activities can be assigned to one developer. Such a sequence may not be dependent on deliverables that have to be provided from activities outside this unit of work and that are not yet available at the start of the unit of work. Many of the planning, management, and reporting attributes throughout a project are associated with units of work.

Role-oriented viewing of a project. The process model and the evolving process plan can provide a view of the progress throughout a project. The



total view can be subsetted to those portions assigned to specific roles. The model itself is static (unless the process model is changed during a project, which should be the exception), but as it progresses, it becomes populated by the instances of activities created for instances of deliverables. By graphical means, it may be shown how far activities have been executed and what status deliverables have reached. This concept can provide developers with an overview of the complete project process, as well as for their specific role.

The process from the management perspective. The management perspective of a process is concerned with the constraints of resources and time under which a logical process can be executed. The number of persons in the development team and in specific roles will be limited, time constraints and cost targets will have to be met, and various ways of optimizing the use of resources to meet the business goals for the project will have to be taken into account. Progress will have to be tracked and evaluated in order to produce auditable progress reports and to obtain early warning signals for plan deviations.

Reporting and documentation. There are many aspects of identifying and maintaining documents. These include: embedding information from various places in a process into one document, administering sets of documents, organizing the printing, electronic distribution, preparation for on-line viewing, and storing for audit and backup purposes. These will require a versatile and integrated documentation management and administration system for development projects. An example of such a tool designed to support the documentation needs of software development projects is Professional Documentation Manager (PrDM*).[20]

The ISO 9000 standards mandate that comprehensive documentation must be provided for projects following an ISO 9000 certified process. The certified process must be well documented and the project must define and explain deviations from the certified process. Documentation must also be provided about the progress of projects over time and about measurements taken and evaluations made throughout the project. Reports and documentation must be kept in order to prove standards adherence, to provide project auditability during the project and after its completion, and to provide data to analyze the process for further improvements. As part of this documentation, it may be necessary to maintain many of the deliverables of a project (such as graphical design objects, user interface design, source

code, and help information) simultaneously in both printable and on-line viewable format.

Measurements. The concepts of process maturity and of Malcolm Baldrige assessments, as well as the ISO 9000 standards, require an organization to accumulate and evaluate data throughout a process and to draw conclusions about process and deliverable quality and required corrective actions.[2] Planning data, assumptions, estimates, and projections have to be aggregated and recorded at certain steps of the process.

For many data to be observed, the elementary source are the units of work. Data may have to be collected about the deliverables involved and about the activities performed.

Information to be recorded for deliverables may be:

- Size
- Effort to produce
- Cost to produce
- Defect history
- Customer satisfaction attributes

Similarly, data may have to be recorded for each executing unit of work in a process. Such data may be:

- Effort
- Elapsed time
- Time to complete
- Defect history

For these data, the system should provide a means for recording initial estimates, revised projections, current values, projections to completion, and final completion values. These data will be needed in the quality system if we are to assess the quality of the deliverables and of the process and the improvements we have made in comparison to past projects.

Benefits of process automation

A highly automated quality system imposes constraints and puts additional requirements on many of the existing tools and requires additional enablers for process integration, process navigation, deliverables housekeeping, process and deliverables quality analysis, and the like. What are the benefits that justify such an effort and investment?

Enabling people. A large number of functions of the quality system that we are discussing represent routine work that has to be performed manually. Today, for example, the developer has to key in data to create a deliverable and then record additional data about the status of the deliverable, or its disposition. This information is redundant when we assume the existence of an interpretable project plan, which defines for each developer what tasks can be done next and what deliverables are involved. With full automation, deliverables will be identified automatically and passed to the tools supporting the tasks. Automatic tool invocation simplifies the process by further reducing the manual tasks of selecting and starting specific tools for various activities within the process.

The developer will be freed from the questions "what should I do next?" and "what deliverable should I be working on?" by automatic tool invocation and can fully concentrate on the creative work to be performed in each task. Being automatically guided, of course, has the additional benefit of preventing the developer from making the wrong decisions, selecting wrong deliverables, or selecting functions that are not yet executable due to the progress in the project.

Monitoring progress. A quality system implemented with an automated control flow navigation continuously "understands" the total state of progress in a system, at a relevant level of granularity. The progress can continuously be viewed and inspected, and early warning signals for out-of-line situations can immediately be communicated to the project manager.

Quality deliverables. A referencing system that at any instance automatically records each deliverable, its status, and its measurable attributes, represents an access and control system for the project. Continuous visibility of these attributes will also ensure that progress in producing the committed quality product is monitored and on target, or else intervention and replanning will be triggered. The potential for erroneous use of input that is not yet at the required state of completion is alleviated, as the navigation flow monitors both deliverable status and activity dependencies.

Process quality and productivity attributes. We have to accept an observation from physics that the function of measurement always influences the value measured about an object. The wealth of data needed for a quality system is only affordable if its collection and evaluation is supported with a high degree of automation and does not impact overall productivity. A computer-aided quality system will collect and analyze the required amount of data and develop corrective signals and feedback to the process before large deviations occur. Achieving higher levels of process maturity will only be feasible with automation for this critical aspect of a quality system.

Continuous improvement. An automated quality system can provide a substantial amount of analysis of collected process data. Only when an enterprise has collected sufficient knowledge over time about the way it does business in the area of software development will it possess sufficient information to analyze and compare all aspects of the applied process—methodology, tools, and personnel—against the accumulated experience. Trends can be detected and corrective actions taken for arriving at more accurately planned and higher-quality future projects. The volume of data to be collected and evaluated again suggests the use of automated process support.

Summary

Development organizations and the worldwide emphasis on quality (as embodied in ISO 9000, Baldrige, Deming, and Humphrey) require that we find a way to improve product quality, process quality, and productivity, both in our own development and in development solutions we offer to our customers. To effectively address these requirements, we need to begin with an environment in which we can depend upon predictable, repeatable processes, while integrating existing tools and technology. The environment must include an AD methodology, metric collection and analysis, project management techniques, and process management principles in a cohesive manner. Such a comprehensive solution could be implemented manually and yield incremental improvements in managing the various aspects of the AD process. An automated solution is required in order to achieve true process improvement, as this can only be realized by continuously monitoring, evaluating, and adjusting the overall process. At the same time, automating the overwhelming tasks of metric collection and analysis, deliverable tracking, and work status monitoring will increase productivity by alleviating this overhead work of a development team. In a total quality management environment, automation becomes the only realistic way to handle the large amounts of data and data analysis required to effectively perform continuous process improvement. A high degree of automation will help us to provide developers with workplaces optimally tailored to their specific roles in a project and enable them to be more creative in producing quality products that meet the needs of their customers.

Acknowledgments

The authors would like to acknowledge the intellectual stimulation they have received from the referenced sources in this paper, and from the Worldwide Application Development Consulting Practice Methodology Development team. For their patient reviews of the work in progress, we thank Kathy Freeman, Dick Antalek, Paul Saracelli, Wayne Stevens, Alistair Cockburn, Pat Gongla, and Gene Sakamoto, whose comments and suggestions helped to make this paper readable.

Cited references

1. W. S. Humphrey, *Programming Process Management*, Technical Report TR00.3320, IBM Corporation (December 12, 1984), p. 2.
2. C. Walrad and E. Moss, "Measurement: The Key to Application Development Quality," *IBM Systems Journal* **32**, No. 3, 445–460 (1993, this issue).
3. *SAA BookManager READ/2 General Information*, GB35-0800-1, IBM Corporation; available through IBM branch offices.
4. *Application Development Project Support/Application Development Model and Process Mechanism—General Information*, GH19-8109, IBM Corporation (April 1990); available through IBM branch offices.
5. *Application Development Productivity Strategy White Paper*, International User Group Council (1989), p. 23.
6. *Application Development Management Project Presentation*, International User Group Council, *GUIDE Fall '92 Conference*, Atlanta (November 1992).
7. *International Standard ISO 9000-3, Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software*, International Organization for Standardization, Geneva (1991).
8. D. A. Garvin, "How the Baldrige Award Really Works," *Harvard Business Review* **69**, No. 6, 80–93 (November-December, 1991).

9. M. G. Brown, "The Baldrige Criteria—Better, Tougher and Clearer for 1992," *Journal for Quality and Participation* **15**, No. 2, 70–75 (March 1992).
10. W. E. Deming, *Out of the Crisis*, Massachusetts Institute of Technology Center for Advanced Engineering Study, Cambridge, MA (1982, 1986), pp. 18–96.
11. F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Quarterly* **13**, No. 3, 319–340 (September 1989).
12. W. S. Humphrey, *Managing the Software Process*, SEI Series in Software Engineering, Addison-Wesley Publishing Co., Reading, MA (1989/1990).
13. *Handbuch fuer DV Projekte—Methoden fuer Planung, Steuerung, Entwicklung und Betrieb von EDV Verfahren*, GE12-1473, IBM Corporation (1978), out of print.
14. *VIDOC Einfuehrungs Broschuere*, GE12-1632, IBM Corporation (1985); available through IBM branch offices.
15. G. F. Hoffnagle and W. E. Beregi, "Automating the Software Development Process," *IBM Systems Journal* **24**, No. 2, 102–120 (1985).
16. G. Chroust, H. Goldmann, and O. Gschwandtner, "The Role of Work Management in Application Development," *IBM Systems Journal* **29**, No. 2, 189–208 (1990).
17. R. W. Phillips, "State Change Architecture: A Protocol for Executable Process Models," C. Tully, Editor, *Representation and Enacting the Software Process, Proceedings 4th International Software Process Workshop* (May 1988); *ACM Software Engineering Notes* **14**, No. 4, 129–132 (1989).
18. *SAA Common User Access, Advanced Interface Design Reference*, SC34-4290, IBM Corporation; available through IBM branch offices.
19. T. DeMarco, *Controlling Software Projects, Management, Measurement and Estimation*, Yourdon Press Computing Series, Prentice-Hall, Inc., Englewood Cliffs, NJ (1982).
20. *IBM SAA AD/Cycle Professional Documentation Manager/MVS & VM, General Information Manual*, GH12-5901-00, IBM Corporation; available through IBM branch offices.

Accepted for publication March 8, 1993.

Kristine D. Saracelli *IBM Consulting Group, 5505 Six Forks Road, Raleigh, North Carolina 27609 (electronic mail: kriss @rhqvm21.vnet.ibm.com).* Ms. Saracelli is currently a member of the Methodology Development group within the Consulting Practice, where her primary focus is on AD Process Management. Since joining IBM in 1979, she has held various positions within Information Systems, including Operations Management, End-User Compute Advocacy, and Information Center technical support. In 1986, she became involved with the competitive marketing team in IBM's National Distribution Division, where her focus was competitive workstation communications products and operating systems. In 1988, she began working to help groups become more effective through the use of groupware in areas such as application development requirements gathering sessions. She joined the consulting group in 1991. Ms. Saracelli received her Bachelor of Science degree in management information systems from Ramapo College in 1982.

Kurt F. Bandat *IBM Vienna Software Development Laboratory, Lassallestrasse 1, A1020 Vienna, Austria (electronic mail: bandat@vabvm1.vnet.ibm.com).* Dr. Bandat received a Ph.D. in communication engineering from the Technical University in Vienna in 1961 and joined the IBM Vienna Laboratory in the same year. He worked on various advanced technology projects in the areas of digital speech signal processing, synthesized speech, and formal definition of programming languages. During a foreign assignment to the IBM Poughkeepsie laboratory in 1968-1970 he worked on programming architecture and on the design and formal definition of command and control languages. Subsequently he was involved in the development of architecture concepts for parallel processing systems. From 1976 on, Dr. Bandat was responsible for the development of application development products (SDF, ADPS, Graphics Interface Kit/2). He is currently working on the technology for solutions involving the use of work management concepts.

Reprint Order No. G321-5517.