A programming process architecture

by R. A. Radice
N. K. Roth
A. C. O'Hara, Jr.
W. A. Ciarfella

The Programming Process Architecture is a framework describing required activities for an operational process that can be used to develop system or application software. The architecture includes process management tasks, mechanisms for analysis and development of the process, and product quality reviews during the various stages of the development cycle. It requires explicit entry criteria, validation, and exit criteria for each task in the process, which combined form the "essence" of the architecture. The architecture describes requirements for a process needing no new invention, but rather using the best proven methodologies, techniques, and tools available today. This paper describes the Programming Process Architecture and its use, emphasizing the reasons for its development.

In IBM's large-system programming laboratories there are hundreds of diverse software products being developed. The Programming Process Architecture provides one common process management view across all of the products while allowing for specific product differences and improvements.

Although there have been many efforts to describe a software engineering environment (SEE), none has used an operational process as its focus and driving force to the extent done in the Programming Process Architecture. Rather, the SEEs have primarily focused on tools and methodologies as the drivers. It is only recently that the solutions for software engineering tools and environments have been understood to require "a solid and formal theoretical base, a unifying conceptual framework, and a coherent programming process."[1]

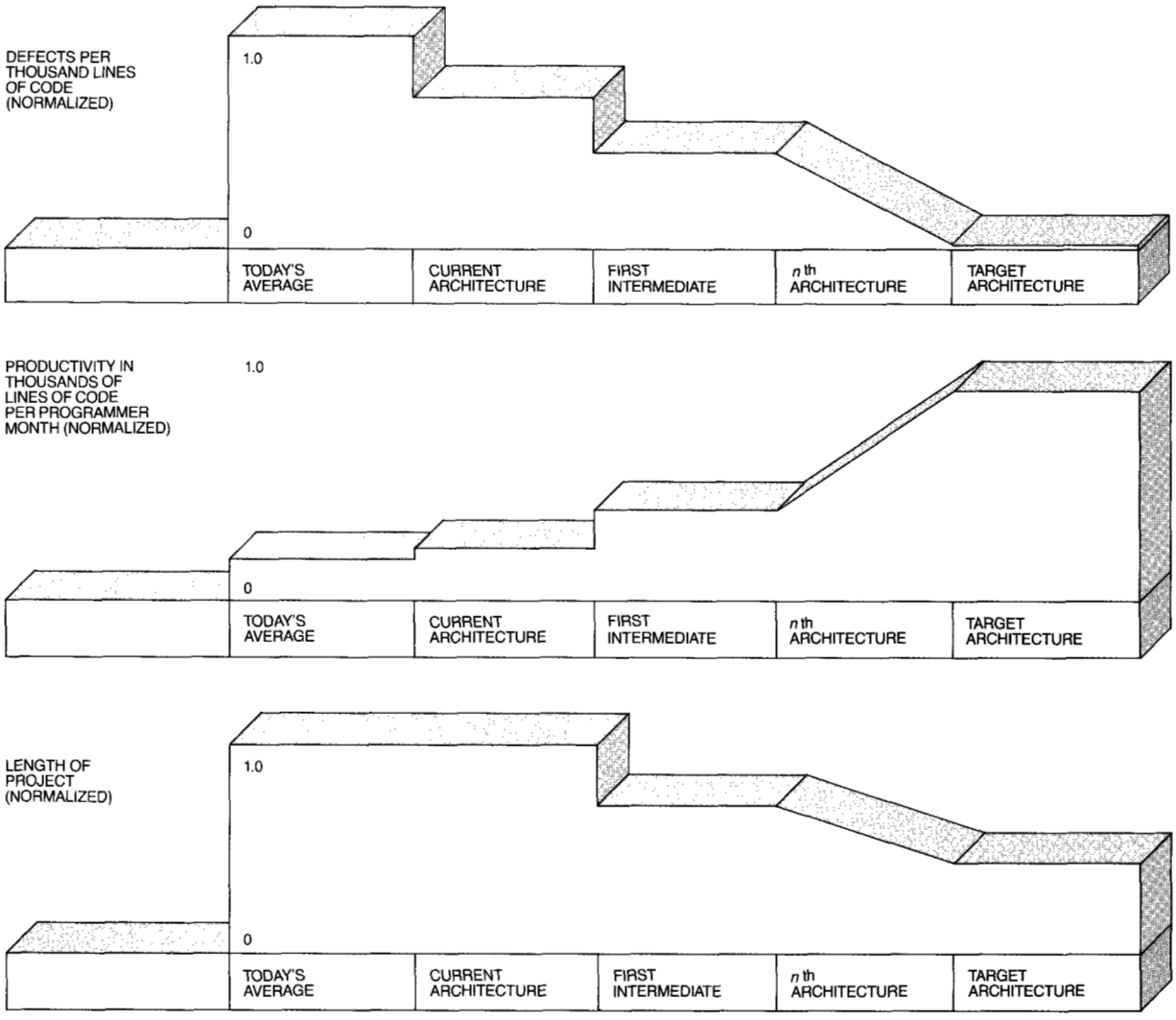
This focus on defining an architecture for the process first, then bringing in the tools and methodologies, marks a key difference in the Programming Process Architecture and in the definition of a future programming environment. Without a clearly defined and accepted architecture for the process, the tools and methodologies can only come together in a loosely coupled manner, with reduced effectiveness. The primary focus on the process allows for a tightly coupled process-tools-methodologies-practices structure on which an SEE can be built and its evolution toward a future programming environment can take place.

The Programming Process Architecture is the basis for this future programming environment, called the Target Architecture, which is the ultimate goal and which will describe a process directing and utilizing future tools and methodologies. The Programming Process Architecture defined in this paper, hereafter referred to as the Process Architecture, is a necessary and orderly step in progressing toward the goal of the Target Architecture.

The first version of this architecture, called the Current Architecture, provides a well-defined homogeneous process for use across many IBM large-system programming sites to

1. Accommodate easier transfer of product development across sites

Figure 1 Improvement progression



2. Assist in the process management of the product
3. Aid in a set of development practices
4. Most important, increase both quality and productivity in our products

It was decided, therefore, that within the area of large-system programs a well-defined homogeneous view of the process would be stated. Once this was accomplished and the sites and projects agreed to use the Process Architecture as a common structure, a full-scale evolution could be pursued to culminate in a final Target Architecture by progressing through a series of intermediate improvements to the Process Architecture. The goal of the Target Architecture for quality is zero defects, with a defect defined as any deviation from the specification. For productivity, the Target Architecture addresses magnitudes of 10 to 20 times today's average productivity rates (see Figure 1).

Thus, the Process Architecture is defining a position from which an orderly evolution of the business of developing software can begin. In order to achieve this, the architecture (1) ensures a repeatable and simple paradigm at all levels of the software process, (2) contains the essence of self-improvement by basing itself on the need for statistical quality control, (3) requires a validation mechanism for any work

elements produced during the development cycle, (4) is based on what is already existent in the software industry, but draws only from the best proven alternatives, (5) addresses the complete life cycle of software production, and (6) is independent of tools in its first iteration.

Need for a process architecture

The product set for which this Process Architecture is primarily intended is the Operating System/370 (OS/370) software, which is exemplified by products such as IBM's Multiple Virtual Storage (MVS), Virtual Machine/System Product (VM/SP), Customer Information Control System/Virtual Storage (CICS/VS), and Virtual Telecommunications Access Method (VTAM). Most of the products produced within this set are functional updates to previous levels. In many instances, multiple releases of a product are being developed concurrently. The code is written either in an in-house high-level language called PLS or in Basic Assembler Language (BAL). A fair amount of the code base has been around since the beginnings of OS/360. In addition, there are new products being developed in the same software laboratories. Therefore, all forms and variations of programming methodologies exist in this product set, including top-down, structured, functional, and data abstractions.

Although there was evidence that there existed, at a high level, a homogeneous software development process across all of IBM's eight large-system product sites, there still were many differences at lower levels. To understand these differences, a Site Study program had been instituted to gather information.[2] While these studies were occurring, all available processes being used by different product groups at the various sites were factored into the initial process model used by the Site Study team. This model, with the actual process performance information from the programming sites integrated with other industry and academic process definitions, eventually became the Process Architecture. In fact, this input flow of actual process performance information continues as the architecture moves through its intermediate states toward its goal.

Acceptance of the architecture

The objectives for establishing a current version of the Process Architecture were to

1. Determine the best available generic process for the eight programming environments developing OS/370 software
2. Achieve approval across the eight sites for a common process model for process evolution
3. Provide a model against which the sites could map more detailed product-specific process definitions

Trying to get eight different sites to agree on these objectives was not an easy task, as each site had a different perspective of what the architecture should be. Among the sites, some product groups

- Wanted more specifics
- Thought it was too specific
- Thought it was too restrictive
- Thought it was too costly to implement
- Questioned its applicability to new products as well as to incrementally changed products

These differences could only be resolved by having the programmers who would have to live with the

Process management techniques apply to all systems software development.

Process Architecture take charge of, or "own," it. This was accomplished by taking a bottom-up acceptance route prior to asking for site approvals.

Thus, the Current Architecture was based on (1) existent and "proven" subprocesses, practices, and methodologies, (2) the experiences of the Site Studies, and (3) a bottom-up acceptance of versions of the Process Architecture. All eight programming sites have approved the architecture and use it as the basis for their process implementations.

What was learned

Much was learned during the development of the Process Architecture. First, many process elements and concepts were broadly applicable across a diverse product set such as exists in the eight programming sites. One example is the Entry-Task-Validation-Exit (ETVX) paradigm, or concept, which is described in detail later. (See also Figure 2, shown later.)

It was also demonstrated that process management techniques apply to all systems software development. Indeed, to achieve consistently improving quality, the management practices of goal setting, measurement, evaluation, and feedback are an absolutely essential part of the process. Defining the process and getting it accepted from the bottom up were the two essential parts of the solution. Establishing goals and the capture, analysis, and feedback of data against these goals were the necessary next steps in the evolutionary process development. As a result, this foundation was integrated into the architecture.

The architecture would not contain the detail necessary for daily operational process activities.

Additionally, it became clear that no one static process could satisfy the needs of all our programming development organizations because of product differences. For this reason, the architecture started with the principle of defining "what" must be done during software development. The "how" of the process was defined in the site or product process documents, which are implementations of the architecture.

It was further understood that the site or product processes should not be trapped into static one-of-a-kind definitions. They should be constantly improving and changing their definitions in the interest of quality and productivity. As a result, process evaluation meetings, known as postmortem meetings, were included in the architecture within the exit criteria to serve as a built-in self-improvement mechanism for the site or product-specific process definitions.

The Current Architecture now represents the best set of proven alternatives for developing systems software under conditions available in today's large-system programming environments in IBM. It is a structured framework of activities that can be adopted immediately by development groups to deliver high-quality systems software. It is a process that can be adopted without having to depend on invention or untested new technology.

Architectural considerations

As previously stated, it was decided at the outset that the Process Architecture should emphasize what was to be accomplished during each segment of the process, not how it was to be done. The applicability of the architecture was intended to be as broad as possible. The architecture was not to contain specific tools or methodologies, but was to remain valid and independent of tools, methodology, and technology changes in order to establish a workable and proven base.

The architecture also would not contain the detail necessary for daily operational process activities. The defined detail was to be provided through either a site process guideline or a product-specific operational process guide.

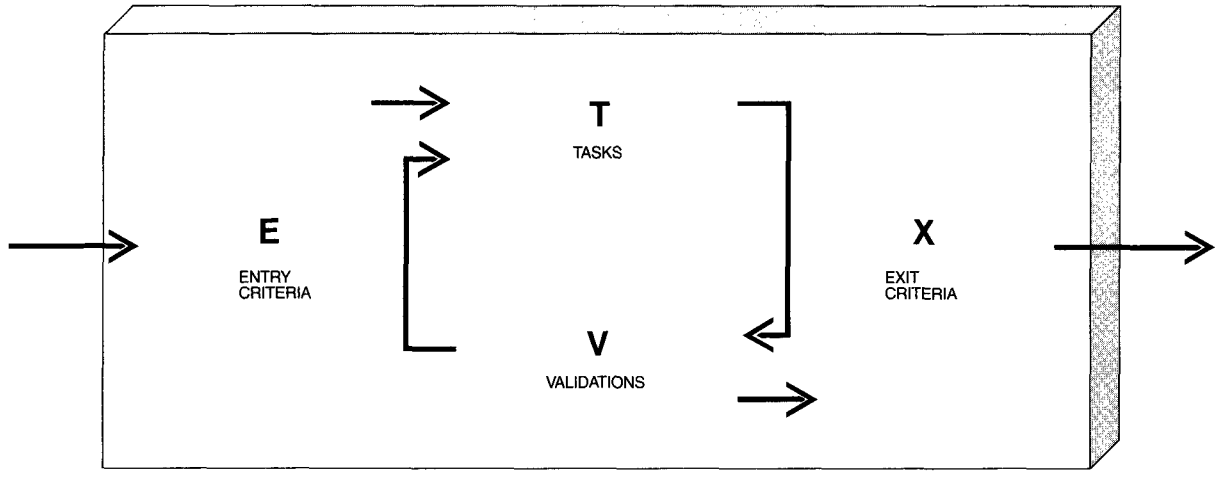
The structure of the architecture had to enable management to track and control the development process and the quality of the product.

Process management

The Process Architecture has been defined to support the following process management principles:

- The process must be actively, continually, and consistently managed to achieve consistently improving quality and increasing productivity.
- Consistent management requires that the process
  1. Be decomposed into parts (process stages)
  2. Have entry criteria, validation, and exit criteria defined for each task
  3. Have process data regularly reviewed, analyzed, and used for process improvement (statistical control)
- Each work item must be validated before being included in the product or its associated information.
- Problems with the product or process must be recorded and analyzed for cause, effect, and improvement.
- Changes to the product or process must be controlled. They should be recorded, tracked, and evaluated for effectiveness.

Figure 2 The ETVX paradigm



- Goal setting, data capture, analysis, and feedback are essential for improvement of both the product and the process.

Essence of the architecture—ETVX

The Process Architecture requires a structured method of process control for each activity within a viewpoint of a stage of the process. This structure, or subprocess, is the "essence of the architecture" and calls for a checklist of entry criteria, tasks, validations, and exit criteria for each activity or stage. The recurring subprocess of entry criteria, tasks, validations, and exit criteria (the ETVX paradigm) is formally defined as an *activity* in the architecture and is illustrated in Figure 2.

Although the ETVX model is used at a stage level, it does not imply that all activities or tasks in a later stage must wait for completion of predecessor stages. The later stages may be functioning in parallel with previous stages. However, in order for a stage to have all activities and tasks exit from it fully, all exit criteria must be satisfied at some point in the product life cycle.

For a given activity, the following are predefined: (1) a list of entry criteria that should be satisfied before beginning the tasks, (2) a set of task descriptions that indicate what is to be accomplished, (3) a validation procedure to verify the quality of the work items produced by the tasks, and (4) a checklist of exit criteria that should be satisfied before the activity is viewed as complete.

The ETVX paradigm indicates the relationships and flow among the four aspects of an activity. If a validation procedure indicates that change or rework is required within a task, the iterative loop of task and validation is followed until it is verified that the items produced by the task are satisfactorily completed.

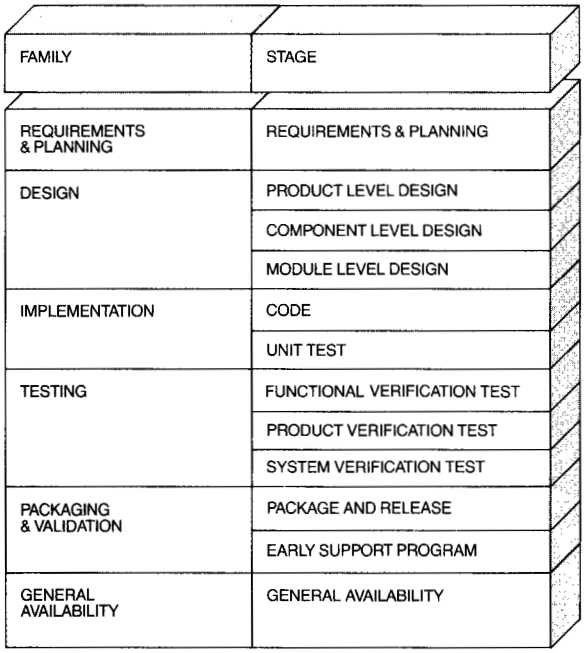
The paradigm displayed with ETVX can be applied to as fine a level of detail as is required to control a process, e.g., the lowest-level task. It can also be applied at the process stage level. Thus, it is a control structure to prevent problems or defects from moving forward from one stage of the process to another.

An underlying theme of the architecture essence is a focus on process control through process management activities. Each stage of the process includes explicit process management activities that emphasize product and process data capture, analysis, and feedback. Through a required quality plan and quality reviews, the product is monitored at every stage of the process. Through process evaluation meetings, the process is also monitored at the end of each stage. This monitoring allows a high degree of control, including corrective action as the product evolves, rather than waiting until testing is completed to determine the probable quality level.

Process stages

The Process Architecture requires that the process be partitioned into stages. Each of these stages may be viewed as a state of evolution of a product. Each

Figure 3 Process stages



stage is named for the major activity that occurs during that time frame. However, many other activities also occur in the same time frame. For example, although a stage is named "Code," it contains tasks for activities in testing, marketing, service, publications, process management, and other aspects of program development. The 12 process stages described in the architecture are shown in Figure 3, together with a grouping into families.

Implementation of the architecture at a development site might result in variations from the version shown in Figure 3. For example, the development of a particular product might show that two design stages are being used rather than three. Nonetheless, the essential tasks within the three stages would be performed under the definition of two stages. Therefore, the work specifics are not different, only the segmentation and the stage names.

The segmentation of the process into 12 stages in the architecture is primarily meant to demonstrate how partitioning of the process is accomplished and not to restrict site process guidelines or operational processes to exactly 12 stages. A brief description of each stage follows:

1. Requirements and Planning (RP): During this period two sets of activities occur: (a) Product and system level requirements are documented and entered into a tracking system, and (b) Project planning is begun with appropriate process and product activities, including creation of a product quality plan.

   In the first set, the architecture refers to a specific methodology wherein requirements are gathered and problem analysis is carried out, documented, and used to create solutions that describe new or enhanced function as a response to the problems. These solutions are then coordinated into high-level design. This procedure is carried out first at the system level and then for specific products.

   In the second, project planning consists of many activities including the completion of an Initial Business Proposal, which is a formal document required early in a development project, the picking and documentation of the process to be used, and the selection and implementation of a management support system for collecting process data, planning and tracking, and analysis, evaluation, and feedback of process and product data.

2. Product Level Design (PLD): A definition of the product functions that will satisfy the requirements is produced during this stage. This stage is the first or highest level of a design statement which is developed as part of the product solution. Performance and usability objectives are identified, and publications planning begins.

3. Component Level Design (CLD): Functions are partitioned and placed in substructures and hierarchical relationships representing the product. These substructures represent an intermediate decomposition of the Product Level Design statement but do not include the level of definition that exists in Module Level Design. The set of module names is defined. The principal data structures and control paths specified for each module in the hierarchy are also defined.

4. Module Level Design (MLD): This stage includes the detailing of each module into a specific logic solution. Each logic path is detailed to denote specific processing activities. Data structures are defined to the lowest level of detail. Installability and serviceability walkthroughs are held; that is,

sessions are held with various members of the organizations that will install and service the product to inspect the materials to be used by these groups. The Translation Plan, a document for the product programming community that defines standard ways in which to translate program messages from English into foreign languages, is completed, and the first drafts of product documentation are available.

5. Code (C): The transformation of the Module Level Design representation into a compilable language with resultant object code is completed. All test plans are completed, and test cases for the Function Verification Test and Product Verification Test are completed.

6. Unit Test (UT): The testing of the logic of each module occurs here to ensure that all logic paths are covered and operate according to the Module Level Design specifications. In some cases, groups of modules may be executed.

7. Functional Verification Test (FVT): The execution of all product functions in an integrated product with respect to the product specification is completed. At completion of this level of testing, the product functions have been proved to work in a simulated and constrained environment.

8. Product Verification Test (PVT): This stage includes execution of all product functions in a real and unconstrained environment. This test will be executed from a user's perspective, but it will not necessarily force the functions to execute in a stressed environment. Performance and usability capabilities are measured as a total product set. When this test is complete, the product can be announced to the market.

9. System Verification Test (SVT): This stage is the execution of all product functions from the user's perspective in an integrated hardware and software environment which will stress the system from performance, reliability, availability, usability, and capability viewpoints. When this test is complete, the product will successfully perform to the requirements from the user's perspective.

10. Package and Release (PR): During this stage, the various parts that define the full product set, including product tapes, tapes of the optional features available, product publications, and installation guides, are brought together as a unified product to perform as the users will see them. The results of this test will demonstrate that a user can install the product and have it operate successfully.

11. Early Support Program (ESP): This stage calls for execution of the product in a set of actual customer or user environments prior to release of the product for general availability. This stage is a verification of the marketing and field support interfaces for the product, and additionally demonstrates that the product functions to the users' expectations in their environments.

12. General Availability (GA): This stage starts with delivery of the product to the marketplace. All manuals, associated products, field service, market education materials, and support and distribution channels must be in place and working satisfactorily.

Stage overlap activity and rework. Even though the architecture segments the process into stages, which are described serially, the actual work occurs in parallel to various degrees across the product life cycle. For example, suppose a problem is found during a test within the Functional Verification Test stage, and it requires rework starting in the Component Level Design stage. The architecture calls for the various unaffected activities of the test stage to continue uninterrupted, while the required activities of the design stage resume to resolve the problem. This overlap of activities from different stages is natural and necessary. The design stage activities then lead to those required in the Module Level Design, Code, Unit Test, and Functional Verification Test stages. This parallelism allows for a smooth, managed, change-controlled process to resolve the problem while other process activities continue undisturbed by the flow of activities from the earlier stages.

Another example would be the parallelism of design, code, and testing at the product level. Although this level of parallelism is necessary for efficiency, the design, code, and test will, of course, be serial for any one module.

Viewpoints

There are many aspects in the development of a product that occur concurrently during each of the stages. These aspects, called *viewpoints*, include the following:

1. Program development
2. Testing
3. Publications and related material
4. Build and integration
5. Marketing and Service
6. Process management

See Figure 4 for a representation of the relationship of tasks with stages and viewpoints.

An example of viewpoints within a stage. An *example* of what may occur in these viewpoints is analyzed for the Functional Verification Test stage:

1. *Program development*—Problems found during formal testing are reported and documented with an appropriate data base. A programmer then analyzes each problem and works with others to determine a solution. The analysis and resolution determine at which stage in the development cycle rework should start. While other testing continues, the activities of the earlier stages required to resolve the problem are formally executed until the problem is closed out. If the solution involves design and code changes, appropriate design documents are updated, and code is written, inspected, and put through unit test prior to integrating it into the driver being used for the functional verification test. All of this is done using a defined automated change control process. Documentation is updated and kept current for all changes.
2. *Testing*—Running the functional verification test cases is the key test activity at this time. Concurrently, a usability test appropriate for this part of the development cycle takes place. Any problems or suggested changes are documented as the first step in a formal configuration management process. Preparation for Product Verification Test is completed. This preparation includes the writing, inspecting, and testing of test cases as well as the organization and training of test team and problem determination team personnel.
3. *Publications and related material*—Drafts of associated publications and any other such materials continue to be refined. Reviews of any language support materials for nondomestic markets are completed.
4. *Build and integration*—The Build Plan, a dynamic document detailing the management and control of all the product parts, is kept current, and test drivers are built and regression-tested to support the testing efforts.
5. *Marketing and service*—The final Early Support Program Plan is created and the Distribution and Support Plan is refined to reflect any changes. The Release for Announcement package is distributed for review. A check on serviceability aspects is completed.
6. *Process management*—Several process management activities occur during this and every stage. Refinement of the Quality Plan and continual monitoring and assessment of the Functional Verification Test process are, of course, key activities here. Others include the product quality review and process evaluation meeting mentioned earlier. They each have specific tasks that are directed by the data gathered during this stage of the process when they are compared to the product and process goals established earlier in the project cycle.

Streamlining

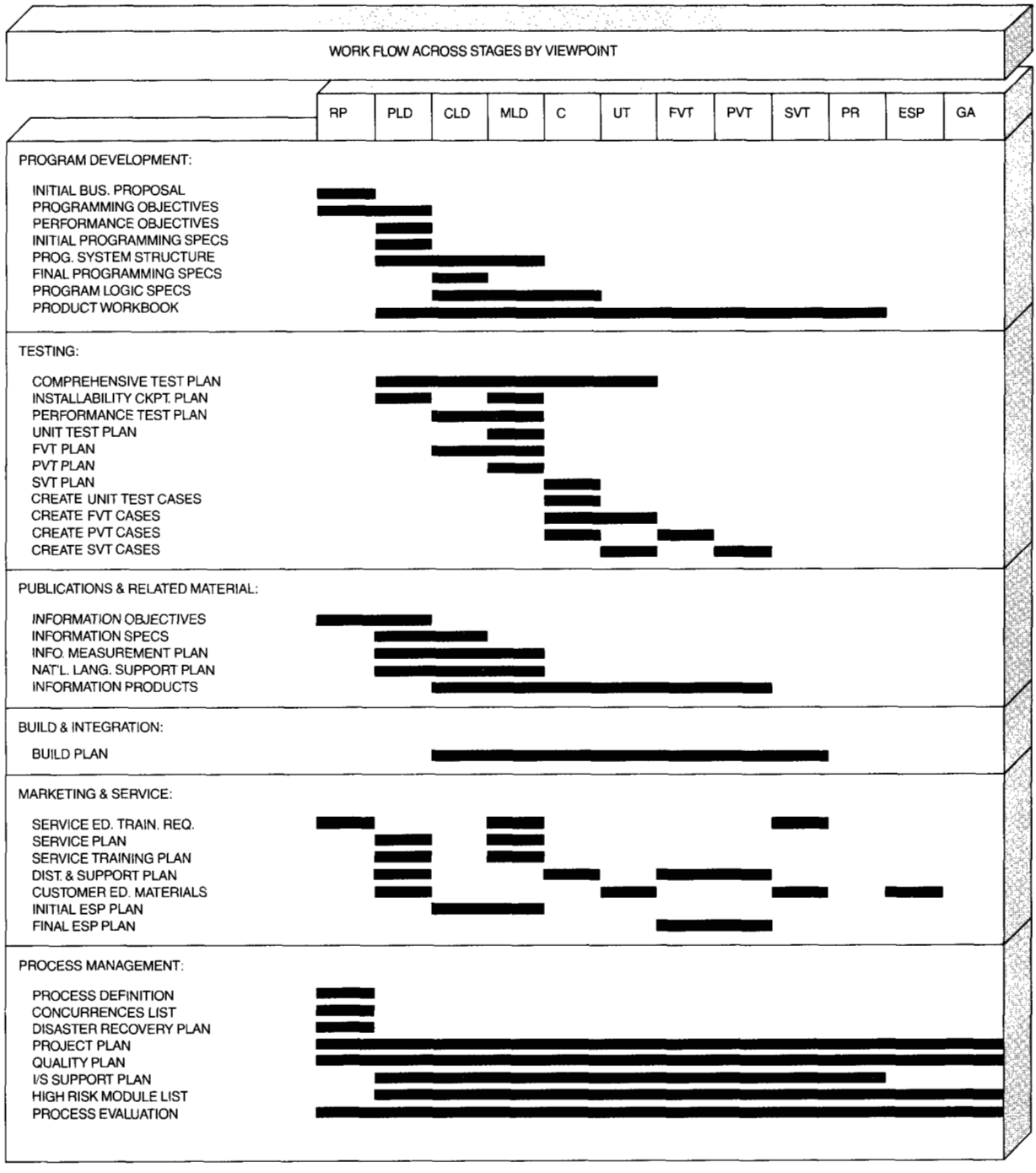
The architecture was developed with an orientation toward improving IBM's large-system software development. However, projects with a short development schedule or with fewer programmers may elect to use a streamlined process. For those projects that may qualify for a streamlined approach, there are key aspects of the architecture which apply to all software development projects and which are required, as follows:

- Segment the process into well-defined stages.
- Segment the different viewpoints necessary to produce a finished product set.
- Adhere to the essence of the architecture at the stage and activity level by defining task descriptions, entry criteria, validation procedures, and exit criteria.
- Establish appropriate process management activities, including a quality plan, periodic quality reviews, and process evaluation meetings.

Quality and product excellence

The Process Architecture calls for a Quality Plan to be established during the Requirements and Planning stage. This plan relates the targeted, deliverable quality of the product to the individual goals for each work item in each activity in every stage. The product manager determines which attributes of quality are to be addressed at each point in the process, and their relative priority. These individual quality subgoals, once set, are continually refined throughout the process as new information sharpens

Figure 4 Work flow across stages by viewpoint



the view. In this manner, quality is managed throughout the product development cycle, rather than only assessed at its end.

Statistical quality control of a programming process requires that measures taken of the quality of each work item be frequently compared with project

goals. Adjustments to work activities are made by management and staff to correct any deviations that become significant. Data collection is an essential activity in every part of the process from the Product Level Design stage to the General Availability stage.

Just as timely data are needed to manage the quality of the developing product, historical data are required to evaluate and correct weaknesses in the process over a succession of projects. Groups that collect data to manage the product are encouraged to enter those data into a data base for later study, analysis, and improvement of the process. Successful accomplishment of this is both a management focus and a technical responsibility.

The issue of quality is concerned with more than defects.

For the Current Architecture, the emphasis is primarily on defect detection and correction. The focus on entry criteria will additionally lead to the prevention of defects that are caused by inadequate and incomplete input to any stage, activity, or task.

The issue of quality is concerned with more than defects. To focus on the subject one needs priorities and measurable goals. These items are almost always product-specific and cannot be successfully dictated by any generalized process or the Process Architecture.

Productivity and process efficiency

When quality is managed in a statistically controlled and predictable manner, the resulting stability produces many additional benefits. For example, productivity gains result from decreased rework, which in turn is directly traceable to earlier defect removal. This productivity gain is caused both by less rework and by fewer iterations on each work item. In addition, when risk is better estimated and contained, improved use of key personnel results from the reduced level of "fire fighting." Finally, historical data for predicting new project milestones can be used more confidently when the relationship between measured quality and productivity is better understood. The Process Architecture emphasizes quality over productivity, with the understanding that as quality improves, productivity will follow.

Development schedules

Early quality goal setting and evaluations can lead to an earlier focus on areas of initial high difficulty. As a result, better initial allocation of key personnel and other resources can follow. Fewer iterations on a given work item can result in less overall calendar time. This reduction in time is achieved from the obvious savings in actual work time and from the reduction in the "queuing effect," in which rework cannot always be performed immediately, but rather must enter a queue in which it competes for attention with other required activities.

Managed entry criteria and exit criteria called for within the Process Architecture means fewer schedule holdups because of wrong or missing task prerequisites. Finally, successful attainment of these entry and exit criteria gives a very visible indication of development schedule progress measured against projected milestones. This implies that the formal sets of stage exit criteria define the major schedule checkpoints and are used in the initial establishment of these milestones.

Intermediate Architecture

While the Current Architecture is meant to serve primarily as the structure from which the common process evolution will begin, the Intermediate Architecture represents the stepping stone to the Target Architecture. This Intermediate Architecture is an evolution of the Current Architecture.

The Intermediate Architecture shows a convergence of existing and nearly developed tools and methodologies with the process. Subsequent architectures, which are additional Intermediate Architectures, may be necessary prior to the resolution of a Target Architecture, and as such they represent a technology roadmap linking the Current Architecture to the Target Architecture. This technology roadmap will show how the process, tools, methodologies, and practices fit together and will map the controlled evolution of a Process Architecture toward a Target Architecture over the next few years.

Summary

The Current Architecture provides for a homogeneous view of the software process and is based on the best available, proven alternatives. Its implementation allows for an immediate improvement in quality and productivity. The users do not have to wait for improvements in methodologies, subprocesses, tools, or practices before using the Current Architecture. The Current Architecture is the essential first step in the evolution toward a Target Architecture.

The Process Architecture leads to a consistent way of producing software while acknowledging that each product will employ the process definition differently, on the basis of individual business judgment. The differences are defined in the process definitions for each product, which are implementations of the approved architecture.

The architecture requires that data be gathered about the process itself, so that the process can be controlled to manage to predefined goals and can be improved from release to release of its use.

The Current Architecture and the Intermediate and Target Architectures are based on the Entry-Task-Validation-Exit (ETVX) paradigm. This model is the essence of the architecture, from the smallest task to the most encompassing stage in the development process.

The Process Architecture was developed from initial input from a number of sources inside and outside of IBM, but more importantly, it was completed using a bottom-up approach with the assistance of the programmers who would live with it. In this sense, the programmers not only helped define the architecture, they own it.

Finally, although this Process Architecture was defined for large-system programming environments, it should apply to any software project employing a number of programmers, whether for systems or application programs.

Acknowledgments

We acknowledge the efforts of Jim Anderson, Hobart Armstrong, Gussie Bailey, Neil Baitenger, Lou Cicilioni, Dick Conklin, Dave Desormeau, Bill Dubbeling, Kusum Gupta, John Henderson, Paul Hutchings, Marv Kaplan, Ken Knapp, Robert Mays, Dick Mikolajczak, Bill O'Neill, Len Orzech, Carol Schneier, Jim Tomsic, Jack Toulan, and Robert Young. Many others, too many to mention individually, especially those reviewers at the programming sites whose comments added considerably to the content, clarity, and brevity of style and ultimate usability and acceptance of the architecture documentation, are also gratefully acknowledged here. The list would not be complete without singular mention of the members of the Site Study team, who took time from very busy schedules to add considerably to our understanding of the process. Finally, we would like to thank Norman C. (Skip) Folden, the original technical team leader of the architecture development.

Cited references

1. M. M. Lehman, "A further model of coherent programming processes," *Proceedings Workshop*, Runnymede, England (February 1984).
2. R. A. Radice, J. T. Harding, P. E. Munnis, and R. W. Phillips, "A programming process study," *IBM Systems Journal* **24**, No. 2, 91–101 (1985, this issue).

General references

J. Aron, *The Program Development Process—The Programming Team, Part 2*, Addison-Wesley Publishing Co., Reading, MA (1983).

V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Transactions on Software Engineering* **SE-10**, No. 6, 728–738 (November 1984).

E. H. Bersoff, "Elements of software configuration management," *IEEE Transactions on Software Engineering* **SE-10**, No. 1, 79–87 (January 1984).

B. W. Boehm, "Software and its impact: A quantitative assessment," *Datamation* **19**, No. 5, 48–59 (May 1973).

B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1981).

F. P. Brooks, Jr., *The Mythical Man-Month*, Addison-Wesley Publishing Co., Reading, MA (1975).

P. B. Crosby, *Quality is Free*, McGraw-Hill Book Co., Inc., New York (1979).

W. E. Deming, *Quality, Productivity, and Competitive Position*, Center for Advanced Engineering Study, Massachusetts Institute of Technology, Cambridge, MA (1982).

M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal* **15**, No. 3, 182–211 (1976).

T. Gilb, *Software Metrics*, Winthrop Publishers, Cambridge, MA (1977).

Z. Jelinski and P. Moranda, "Software reliability research," *Statistical Computer Performance Evaluation*, W. Freiberger, Ed., Academic Press, Inc., New York (1972), pp. 465–484.

T. C. Jones, "Measuring programming quality and productivity," *IBM Systems Journal* **17**, No. 1, 39–63 (1978).

N. Kleid, "IBM's information quality measurement program," *Proceedings of the 31st International Technical Communication Conference*, Society for Technical Communication, Seattle, WA (April 29–May 2, 1984), pp. MPD.62–MPD.65.

R. C. Linger, H. D. Mills, and B. I. Witt, *Structured Programming, Theory and Practice*, Addison-Wesley Publishing Co., Reading, MA (1979).

W. A. Madden and K. Y. Rone, "Design, development, integration: space shuttle primary flight software system," *Communications of the ACM* **27**, No. 9, 914–925 (September 1984).

R. G. Mays, L. S. Orzech, W. A. Ciarfella, and R. W. Phillips, "PDM—A requirements methodology for software system enhancements," *IBM Systems Journal* **24**, No. 2, 134–149 (1985, this issue).

H. D. Mills, D. O'Neill, R. C. Linger, M. Dyer, and R. E. Quinnan, "The Management of Software Engineering," *IBM Systems Journal* **19**, No. 4, 414–477 (1980).

G. E. Murine, "The application of software quality metrics," *Proceedings of the IEEE 1983 Phoenix Conference on Computers and Communications*, IEEE Computer Society Press, Piscataway, NJ (1983), pp. 185–188.

M. Ohba, "Software reliability analysis models," *IBM Journal of Research and Development* **28**, No. 4, 428–443 (July 1984).

D. O'Neill, "The management of software engineering, Part II: Software engineering program," *IBM Systems Journal* **19**, No. 4, 421–431 (1980).

R. A. Radice, "Large Systems Software Implementation," *Eighteenth IBM Computer Science Symposium*, Shimoda, Japan (September 1984); available from author.

M. L. Shooman, *Software Engineering*, McGraw-Hill Book Co., Inc., New York (1983).

D. Teichrow, P. Macasovic, E. A. Hershey III, and Y. Yamamoto, "Application of the entity-relationship approach to information processing systems modelling," *Entity-Relationship Approach to Systems Analysis and Design*, P. P. Chen (ed.), North-Holland Publishing Company, Amsterdam (1980), pp. 15–30.

C. E. Walston and C. P. Felix, "A method of programming measurement and estimation," *IBM Systems Journal* **16**, No. 1, 54–73 (1977).

G. M. Weinberg, *The Psychology of Computer Programming*, Van Nostrand-Reinhold, New York (1971).

G. M. Weinberg, *Rethinking Systems Analysis and Design*, Little, Brown and Co., Boston (1982).

N. Wirth, "Program development by stepwise refinement," *Communications of the ACM* **14**, No. 4, 221–227 (1971).

Ronald A. Radice *IBM Information Systems and Storage Group, P.O. Box 390, Poughkeepsie, New York 12602.* Mr. Radice received his bachelor's degree from Upsala College in 1966 and joined IBM as a programmer immediately upon graduation. He held a variety of technical jobs in design, development, and testing of systems software products. In 1972, with M. E. Fagan, he began his work with software engineering which resulted in the inspection process for software design. Mr. Radice began his management career in 1974. Today he is the Information Systems and Storage Group Manager of Programming Process and is responsible for two groups that developed the Site Studies and the Process Architecture. He is a member of IEEE, ACM, and ASQC and is an Adjunct Associate Professor of software engineering at Rensselaer Polytechnic Institute, where he teaches software engineering courses in the graduate program.

Norman K. Roth *IBM Information Systems and Storage Group, P.O. Box 390, Poughkeepsie, New York 12602.* Dr. Roth received his doctorate in mathematics from the University of Massachusetts at Amherst in 1971. In 1968, he joined the Mathematics Department at the State University of New York College at New Paltz and was Chairman of the Mathematics and Computer Science Department there from 1978 to 1981. He received the State University of New York Chancellor's Award for Excellence in Teaching in 1975. Dr. Roth joined IBM in 1981 and worked in the VTAM Development group on the SNA Network Interconnect project. For that effort he received a Communications Product Division Award. In 1983 he joined the Quality and Process group of IBM's Information Systems and Storage Group to work on its Programming Process Architecture project. His interests are in groups, fields, graph theory, and software engineering as it applies to large-system software development.

Almerin C. O'Hara, Jr. *IBM Information Systems and Storage Group, P.O. Box 390, Poughkeepsie, New York 12602.* Mr. O'Hara obtained a B.E.E. from Union College in 1959. For the past twenty-six years he has worked in technical and management capacities in engineering, programming, and systems analysis and planning. He wrote one of the first interactive graphic programs for the analysis of electronic circuits. In addition, he has done extensive application development and user support, as well as information resource management, data analysis, and process modeling. Currently Mr. O'Hara works with other IBM locations to help define and understand programming process architecture, methodologies, and models of the programming process. His research interests are in the areas of abstraction and modeling of management aspects of development processes, modeling of man-machine processes, the quantification of relationships between measures of productivity and measures of quality, exploring those factors which limit the attainment of defect-free work products, and the prediction of systems performance.

William A. Ciarfella *IBM Information Systems and Storage Group, P.O. Box 100, Kingston, New York 12401.* Mr. Ciarfella joined IBM in June 1980 and is currently a member of the Software Engineering function at the Kingston Programming Center. Since joining IBM, he has worked on software requirements and design systems. Most recently, he was a member of the team that developed the IBM Information Systems and Storage Group Programming Process Architecture. He received a B.S. degree in mathematics and computer science from the State University of New York College at Brockport.

Reprint Order No. G321-5240.