The role of work management in application development

by G. Chroust
H. Goldmann
O. Gschwandtner

Quality is probably one of the most serious concerns of today's software community. For software applications exhibiting a certain complexity, the quality of a product can only be guaranteed by a methodological approach, using appropriate administration and tools. The methodology and administration must be manifested in a well-defined and well-observed application development process. The process must integrate the human activity, the tools, and the intermediate and final work products into a coherent flow of actions. In this regard, the development of applications follows patterns that are well established in other industries where an application development (AD) process model is defined and then executed via an interpretation mechanism. The complexity of the development process makes it necessary to support and integrate all of its aspects by means of on-line interactive computer support. Computer-aided process support in the general sense we call work management. This paper explains the concepts of an application development process model and of work management for application development under AD/Cycle™ and its relation to project management.

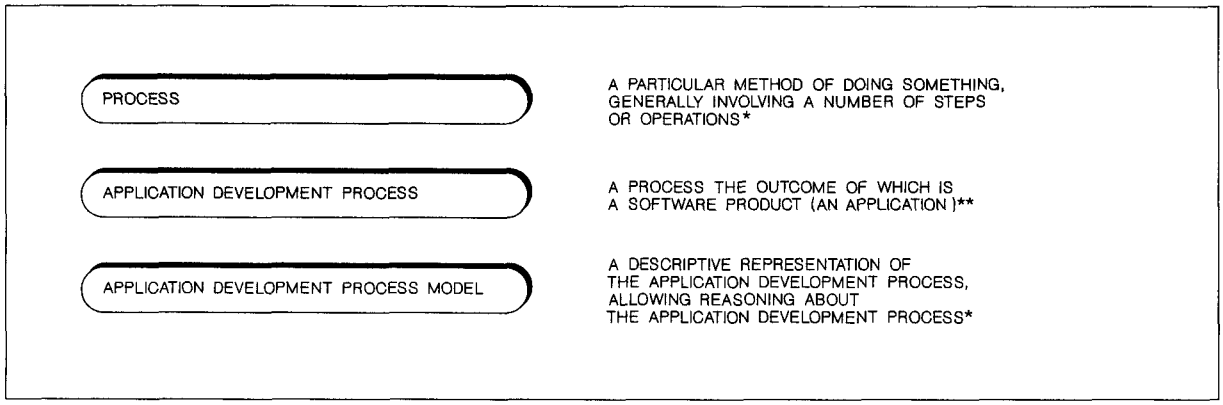
The concept of an application development process interrelates cost, timeliness, and quality—three closely interleaved problems[1,2] that haunt today's software industry. Quality affects the other two problems. Inadequate quality results in excessive testing, rework, and maintenance which in turn influence delivery dates and productivity, and therefore cost. Thus the issue of software quality seems central to the progress of the software industry. This paper discusses the concept of an application development process model and of work management for application development under AD/Cycle™.

Application development process

The concept. Most quality-related attributes[3,4] cannot be adequately measured and are difficult to predict. Even if a measurable value is found to be low, there is usually no easy way to improve it in a finished product, because one cannot inject quality into a product. This means that concerns about the quality of a software product must be addressed during its creation. In order to ensure a consistently high level of quality in a software product, one must ensure that quality be an integral part of the production process—a message that many other industries have understood for many years.[5]

The production process then becomes the object of scrutiny,[6] similar to that in other industries where quality products of any type are produced according to a stable, well-understood process which is somewhat independent of the product.

Figure 1 The application development process



* See Reference 7.

** See Reference 34.

An important step of abstraction is to separate concerns about the product (i.e., the application to be created) and the process by which it is created. Once separated, the application development process (see Figure 1) can be described and evaluated independently.[7]

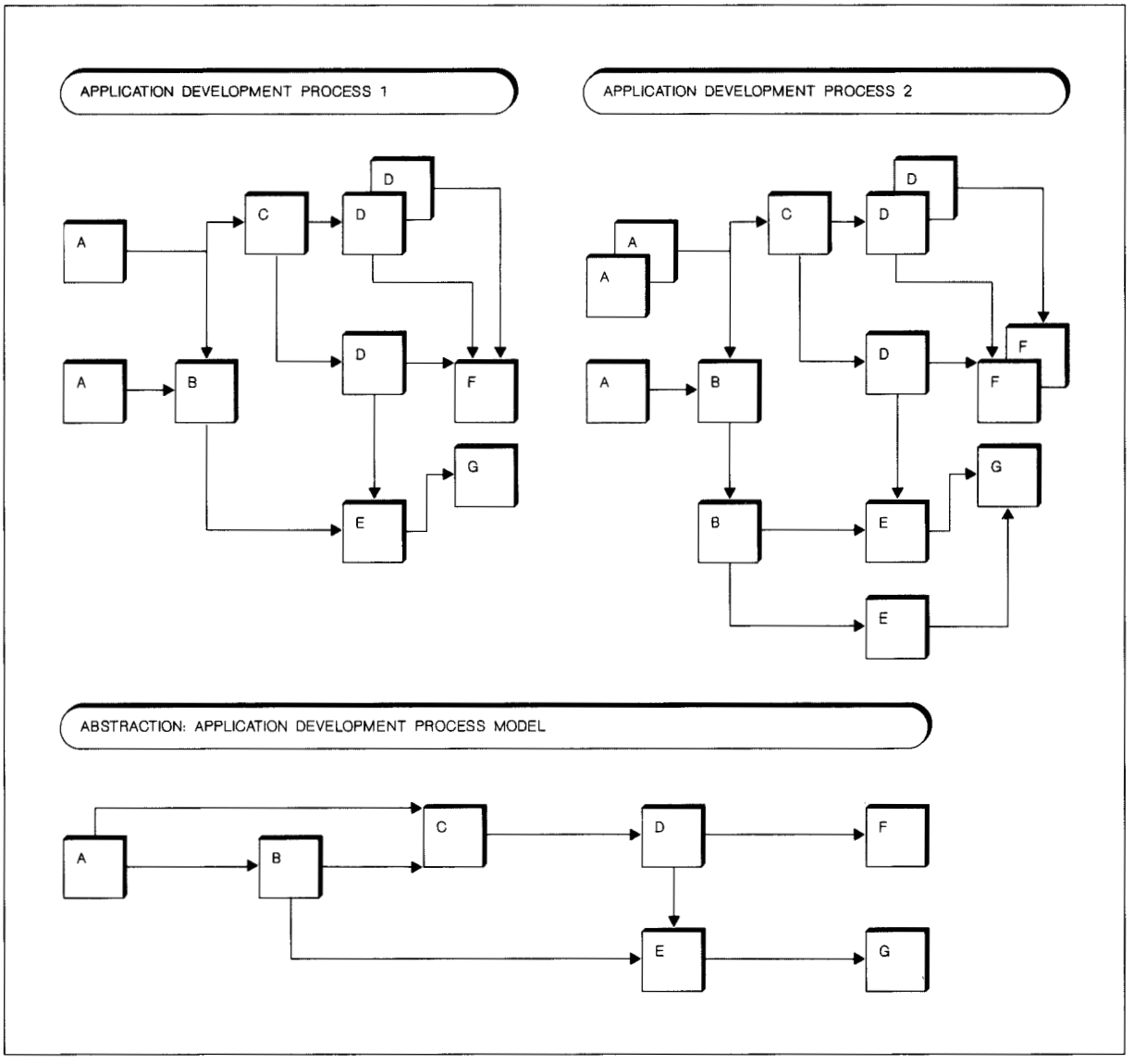
Conceptually an application development process model, sometimes called simply a process model, is an abstraction from actual application development processes (see Figure 2) and is intended as a template, or prescription, for future application development processes. The simplest structure of a process model describes the results—called the work objects—to be created and the activities which are necessary to produce them. The term *work object* is used to denote all the different results (final and intermediate) that are produced during application development. The creation of an application is achieved by a process based upon the application development process model. The actual process is then known as an *instance* of the model, with the understanding that all elements of the instantiated process (typically activities and work objects) are instances of the respective descriptions of activities and work objects in the model. An application development process model usually provides additional information which eases and standardizes the process.

The application development process model. Successful application development is the result of several factors. The ultimate aim of an application development project is the delivery of a set of work objects (e.g., code, documentation). It is necessary to derive them via numerous intermediate work objects, each of which shows a different aspect of the application. These work objects have certain relationships among them (see Figure 3). The totality of this information is described in the application development information model (AD information model) of AD/Cycle, which represents the work objects and their relationships. Further details are discussed later in the paper in the section on the AD information model. In addition, one has to establish the means by which these work objects are to be produced, that is, which steps should be taken and which methods applied. This can be stratified in two levels: (1) The individual steps to be taken can be defined locally and are further explained in the section on activities. Each activity defines one step, the inputs, the outputs, and what should be done in the step. (2) The methodology which performs the overall process is defined globally. The section on the work-flow structure shows how the individual activities are to be sequenced in order to form a complete process.

The AD information model and work-flow structure together form the core of the application development process model. A description of the process is shown in Figure 4.

The need for computer support. Application development process models have been around for many years. Some time ago, IBM Germany published and taught a textual description of a process model called "Verfahrenstechnik."[8] Further models include Boehm's waterfall model,[9] an IBM model,[10] and many others.[11–13] Such descriptions tend to be largely ignored because they offer information which is re-

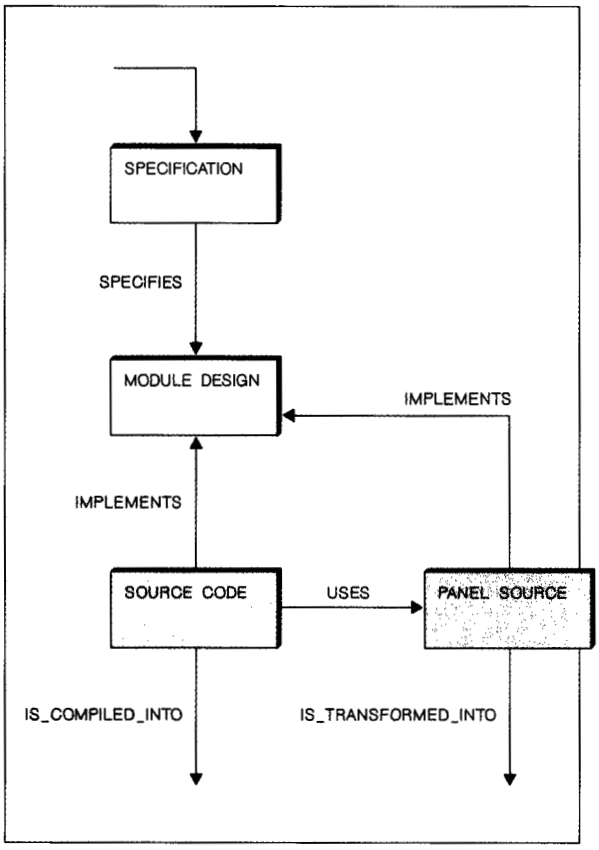
Figure 2 Abstracting an application development process model–the letters indicate activities that, when abstracted, result in one step in the application development process model



mote from the actual point of need, are often cumbersome to look up, and are difficult to maintain—especially if they exist in many copies.

In order to be fruitful, it is necessary to bring the developer into intimate interaction with the process definition. The complexity of the process, the multitude of intermediate and final work objects, the number of team members involved, and the need for high quality require computer support. A work manager is supported by a database system. An early account of these concepts, probably coining the term *process mechanism* for the first time, is found in Reference 14. Thus we develop applications not only for the computer, but also with the computer. In AD/Cycle we have defined Work Manager to be the work management component, discussed in a later section.

Figure 3 Relationships among work objects



Supplying the Work Manager with all necessary features needed by the members of the development team (e.g., tools, help texts), yields an environment invariably called an *Integrated Project Support Environment* (IPSE)[15] or *Software Engineering Environment* (SEE).[16–18]

The process model provides a description of the application development methodology. In this respect it can be compared to a street map, which indicates acceptable ways to reach a certain location. An on-line process model can also actively provide guidance to its users. The objective of process management is to aid in determining the order of activities and to communicate with the users.

It is also necessary to administer the produced (intermediate and final) work objects when performing actual application development. Storage and retrieval procedures have to be established, and tools have to be accessed.

Implementing a project is not just a matter of following a process model. People, schedules, and resources have to be considered and planned. This is the objective of project management. Project management will also (based on resource availability and on project priorities) influence the order in which activities may be executed. Extra tasks must be planned for education or vacation. The Work Manager will be controlled by the additional constraints imposed by project management. The Work Manager provides automated on-line process management by utilizing elementary project management functions and providing interfaces to project management tools.

Components of an application development process model

The components of an application development process model include the AD information model, the work-flow structure, and auxiliary information such as help texts, skeletons, and samples. It is an advantage to divide a process model into several model segments.

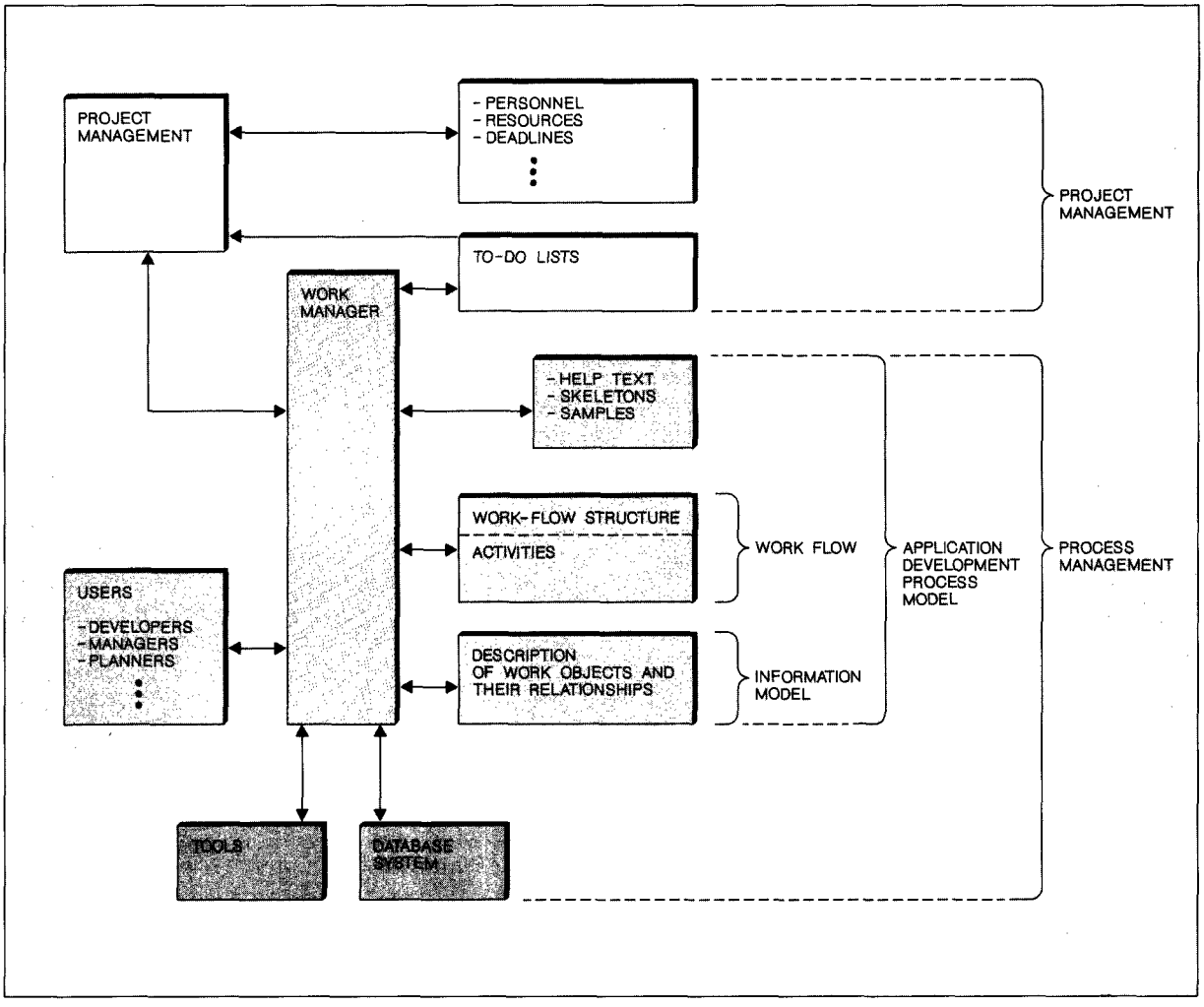
The AD information model. The AD information model represents the work objects, their relationships, and attributes.

Work objects and their relationships. The aim of a development project is the delivery of a set of work objects (the code, the documentation, etc.). The impossibility of deriving these end products in one step from the initial information makes it necessary to define several layered intermediate work objects.

The application is usually described by separate views (e.g., as the user sees it, how it is to be implemented, etc.) that are reflected in different documents (the various work objects). These work objects provide specialized views of the final application. This is similar to building a house, where different documents (floor plan, plan of plumbing installations, functional overview, etc.[19]) are created before a single brick is laid. At the end of a project an architect hands to the customer not only the keys of the house, but also a considerable amount of the intermediate documentation.

Similarly, an application may be considered as the set of all work objects that are created during the

Figure 4 Components of an application development project



application development process (e.g., the programs, the specifications, etc.). Each work object makes some contribution to the meaning and shape of the final application, providing some aspect (view) of the final application. These different work objects bear numerous relationships to one another, as indicated in Figure 3.

Most intermediate work objects should be preserved for auditing and maintenance at a later date. Thus the installed application is, so-to-speak, the only end-user visible part of a large set of work objects. All the created work objects may be of interest for a complete understanding of an application.

In AD/Cycle the description of the individual work objects and their relationships is stored in the AD information model. The AD information model is intended to be the common basis for tool integration and communication. The AD information model is extendable; this means that if a model segment (discussed in the section, "Model segments") needs additional work objects that are initially not in the AD information model, they can be added.[20,21] Actual instances of these descriptions, i.e., the work objects resulting from application development, are stored in the Repository Manager™. When performing actual application development, some of these relationships can be formally verified; with respect to others,

a more intuitive understanding is necessary. A correct (and complete) application will consist of a set of work objects that fulfill all the relationships.

The choice of which work objects are specified in the AD information model and their interrelations predetermines to some extent the methodology for producing them. The information model does not explicitly specify the order in which the individual work objects are to be created. It does express a goal—that is, what the structure of the final application should be. Naturally the structure has to be compatible with the development methods, i.e., it must provide for all intermediate work objects which are needed if following specific methods.

The AD information model describes the work objects and their relationships.

Attributes of work objects. In addition to its contents, one usually wants to record certain important facts (or attributes) about a work object. Several of these attributes will influence the direction in which the project proceeds. Because application development is a team effort, the attributes are also a basis for communication between team members.

Attributes of a work object may describe the following:

- Contents of the work object in a concise form, such as statistical data—especially when the computation of this attribute is difficult (e.g., the number of noncommentary source statements of a program[22])
- Historical information about a work object (e.g., creation date, last update)
- Administrative information, such as the name of the library where it is stored
- Project-oriented information, such as the owner, the authorizations to modify, etc.
- Statements about the work object, which are based on human evaluation, e.g., the completion state or level of quality

It should be noted that some of the interesting attributes, such as the completion state, cannot be assigned automatically, but only by human interaction. In general the computer cannot deduce whether a work object is finished, although there might be indications of disbelieving the developer's statement (if the compilation of a work object produces a severe error, it is obviously not finished).

The process model can restrict the values a work object attribute may have and the authorized transitions between the values. Changes to attributes may only follow established transition rules as illustrated in Figure 5.

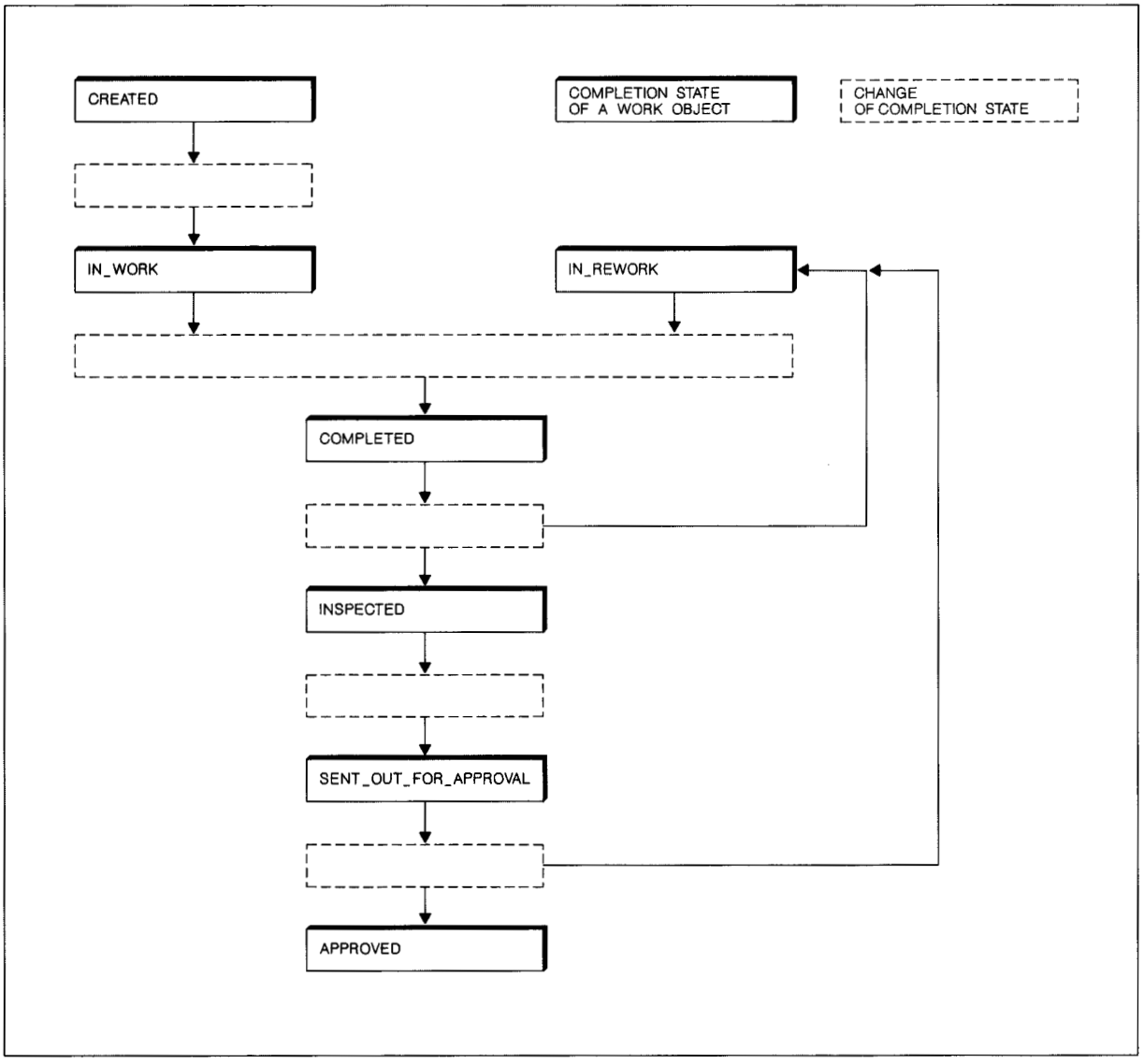
Work flow. The AD information model describes the work objects and their relationships. In an actual application development project, instances of the descriptions have to be created. This means that the steps to do the activities and their sequence (the work-flow structure) must be established in order to populate the Repository Manager with actual data.

Activities. An *activity* is considered to be the smallest unit of work identified on a certain level of the model. It specifies which work objects are to be worked on in one elementary step and which work objects are assumed to be prerequisites for this work. The *before*, *after*, and *together* activities for the work objects are defined.

An activity has to take into account the available or anticipated tools and also the relationships expressed in the AD information model. In simple cases an activity can be equated to the call of one or more tools; for many activities this may be just an editor. Based upon the current state of the work object, certain tools may not be eligible for invocation. Thus activities define the local use of tools and practices. In other words, identifying activities can be seen as interpreting certain of the relationships in the AD information model as transformations. For example, in Figure 3 the relationship of IMPLEMENTS can be taken to indicate that one should create SOURCE CODE (and also a PANEL SOURCE) based upon the MODULE DESIGN.

The choice of activities is not always straightforward: In a reverse engineering situation it can be deduced

Figure 5 Assignment of completion state to work objects



from Figure 3 that the module design should be recreated from the source code and the panel source.[23] Similarly the function-oriented and data-flow-oriented paradigms of application development[24] differ with respect to the chosen types of transformations. Thus different transformations (activities) with different prerequisites and results may be defined, despite the fact that the underlying AD information model is the same. Such differences will be reflected as different process models over the same AD information model.

From the relevant literature[15–18] it can be observed that the different paradigms of software development that are discussed are essentially concerned with the order in which activities are to be executed (the work flow), and with certain notational conveniences of describing the contents of the various work objects, while the contents of an information model are of little concern.

The definition of activities can also mean clustering several transformations into one step. This clustering

defines the "together" aspect of the activities. (In Figure 3 both the SOURCE CODE and the PANEL SOURCE might be produced in one step for good reasons.)

The choice of what constitutes an activity should also be consistent with the granularity of the AD information model. In other words, in each activity one or only a small set of work objects should be created.

Defining activities together with their prerequisites and results obviously imposes some ordering on the transformations. It defines sequencing on a local scale. Having defined the activities, only certain ways of traversing the model are still valid. This is similar to a street map where all streets are marked as one-way streets.

Work-flow structure. While the definition of activities can be considered a local introduction of work flow into the development process, the definition of how these activities should follow one another can be considered the global definition of the development method. Successor relationships can be established between the existing activities, resulting in a network-like arrangement of the activities, the work-flow structure.

Thus activities in the network are interrelated because the output work objects of some activities are input to other activities and may be restricted by the relationship holds. The definition of activities and of the work-flow structure are complementary.

For further control of sequencing (the navigation), we introduce entry predicates and exit assertions in the sense of the *state change architecture*[25] (see Figure 6).

Each activity may have an *entry predicate* that determines whether the activity may be performed. In general, this entry predicate is a logical expression involving attributes of some of the input work objects. Which attributes are actually process-relevant is only determined by their occurrence in some entry predicate or exit assertion.

An activity may also have an *exit assertion* that indicates whether this activity has produced all its expected results. It is in general formulated as a logical expression of the attributes of some work objects.

In the Figure 6 example, the activity CODE creates both the SOURCE CODE and the PANEL SOURCE. The work-flow structure may add an additional successor relationship between PRODUCE SPECIFICATION and EXTERNAL REVIEW, and CREATE DESIGN. It implies that despite the fact that CREATE DESIGN does not have an output or input dependency on EXTERNAL REVIEW, there is still a successor relationship.

Work object based work-flow structure. Experience has shown that in certain cases the definition of an activity can be understood by default using the relationships between the affected work object. In this case it is not necessary to explicitly define this activity.

As a consequence one can derive the order in which work objects should be created by defining a state change protocol for every work object. Based upon this state change protocol, the connection between work objects is mirrored by a connection between certain states of the work objects.

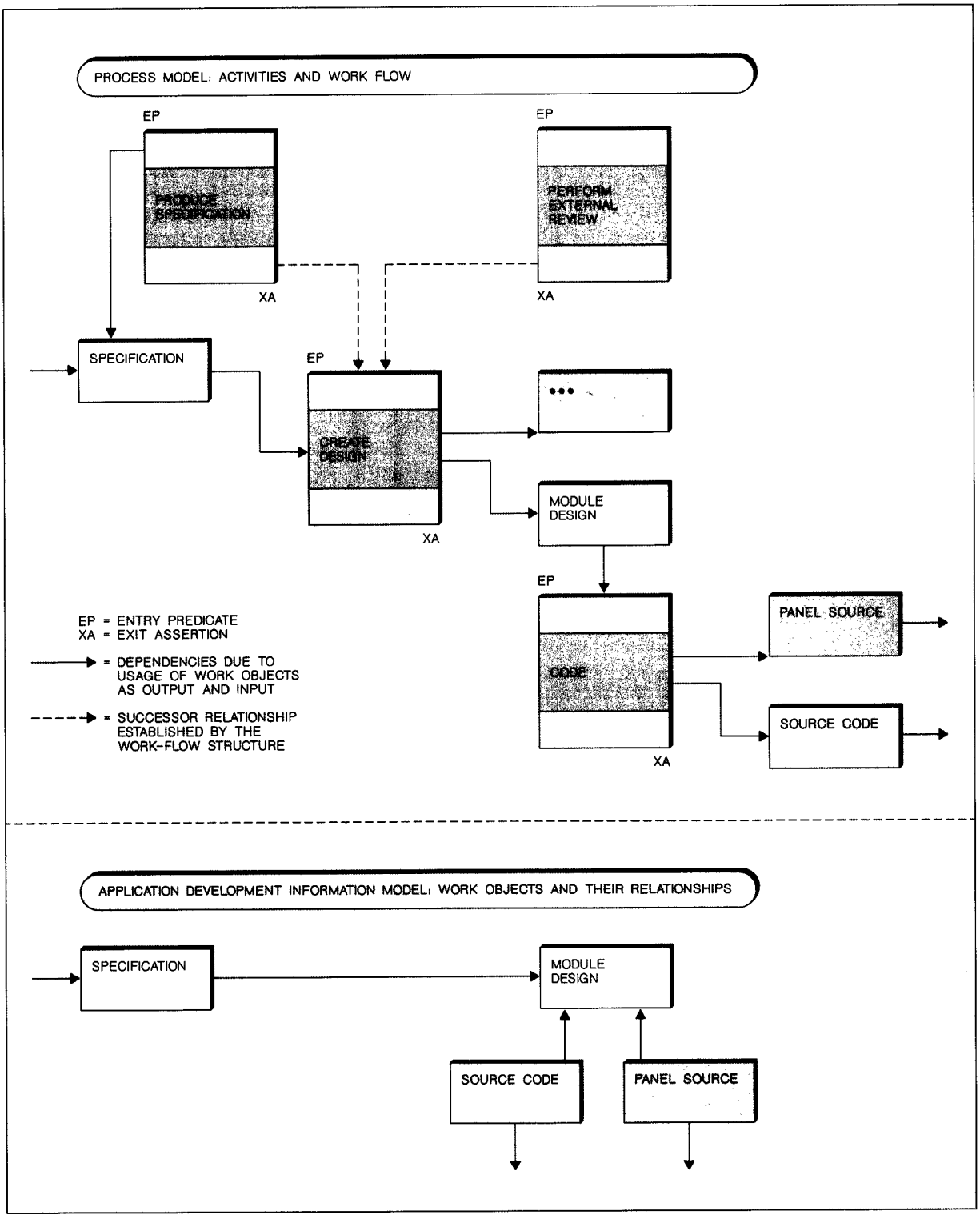
Help texts, skeletons, samples. The availability of computer support allows some additional productivity aids to be provided to the user. By providing on-line help texts, information about standards, and standardized skeletons for the results to be created, a more uniform and professional application results. The most important productivity aids are the following:

- On-line *help texts* are an essential component for user-friendly, usable systems.
- Reuse and standardization are enhanced by providing a specific *skeleton* for editing parts of a work object.
- *Samples* are read-only examples of completed work objects used as guidelines for work.

The above components are an integral part of any process model. The Work Manager has to take care that they are presented to the user in an appropriate form.

Model segments. A complete application development process model can be considered to consist of several separate model segments, which describe separate areas of responsibility or separate subprocesses. One can, on the other hand, also consider application development process models to be built from several model segments. This allows alternate model segments to exist in order to handle some tasks by different means. Model segments that are to be com-

Figure 6 Information model and process model



bined must exhibit a strong compatibility with one another.

The major model segment of any application development process model is the run-time application model segment, which describes the creation of the actual application. This core is usually augmented by several other model segments that describe activities not directly involved in the creation of the application, but nevertheless are integrally needed in order to create a quality application. The run-time application model segment is obviously the backbone of the process.

Model segments exist for different reasons, such as adding a perspective, the description of a tool, or choosing alternate paradigms.

In a development project, several areas of responsibility are often only loosely connected to the run-time application model segment and are concerned with creating work objects of their own. They are often called *perspectives*. Since the run-time application model segment is the central core of the process, all other model segments are dependent on it and must mirror the structure of the application development and be compatible with it. The quality assurance model segment,[26] for example, must have a structure analogous to that of the run-time application model segment. Major perspectives are documentation, quality assurance, and product control.

Another important example for a model segment is the *description of a complex tool*. Such a tool essentially performs a certain set of activities in the development process and thus can be described by a model segment. This allows the user to compare and match tools with an existing model.

For certain areas of application development, different methodologies or *paradigms* can be used. Typically the technical design of an application can be function oriented, data oriented, object oriented, or data-flow oriented. Having different model segments available allows the user to isolate the decision on the methodology and allows the model builder to choose the appropriate paradigm.

In order to build an application development process model, it is necessary to combine several such model segments as shown in Figure 7. Due to the interdependencies of the various model segments, some tailoring may be necessary before combination. The combination of model segments can also be considered reuse of partial models.

Combination works somewhat like set union, that is, identical work objects and activities in the model segments are collapsed into one, while differing work objects and activities are collected in the resulting model. Obviously the tailoring and combination function can be driven to unlimited complexity. For practical purposes, however, relatively simple functions like union, simple substitution, etc. are sufficient.

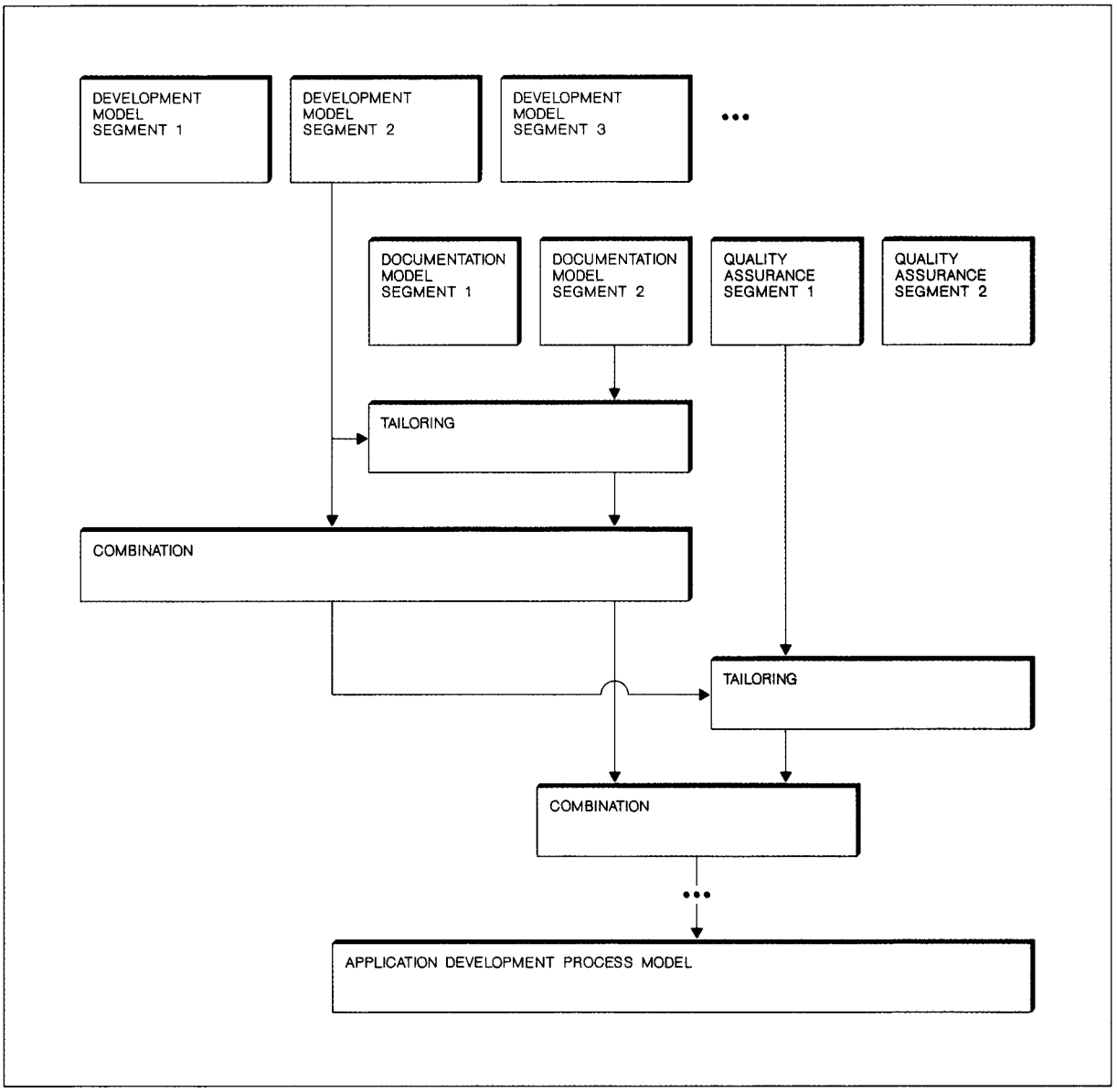
Examples of model segments. Important model segments (similar to ADPS[26]) are: the run-time application segment, the product documentation segment, the test environment segment, and the quality assurance segment. These are discussed in the following paragraphs.

The *run-time application model segment*, based upon the AD information model of AD/Cycle,[20,27] contains the definition of those work objects that contribute directly to executable user application, as discussed in a previous section, "The AD information model." It remains to define the activities and to specify the work-flow structure associated with it. The work-flow structure will follow the phase model of AD/Cycle[28] (requirements, analysis and design, produce, build and test, production and maintenance). Beneath that there will be groups of activities and single activities defined, closely resembling the process model of Application Development Project Support (ADPS).[26,29]

The second model segment describes product documentation. Product documentation (comprising error messages, help panels, and manuals) must be developed as an integral part of software development, in parallel with the actual application. This *product documentation model segment* uses as input all relevant work objects from the run-time application model segment (this means it must be tailored after the run-time application model segment) and delivers the necessary documentation work objects to the appropriate integration activities of the run-time application model segment.

The third model segment is used mostly for testing, but it can also be the basis for experimenting with code fragments. The developer needs an environment that mimics reasonably the target environment. This environment will be built on several occasions, e.g., for module test, integration test, and system test, etc. This *test environment model segment* is fairly independent from the development model segment, with the restriction that it must be com-

Figure 7 The concept of model segments

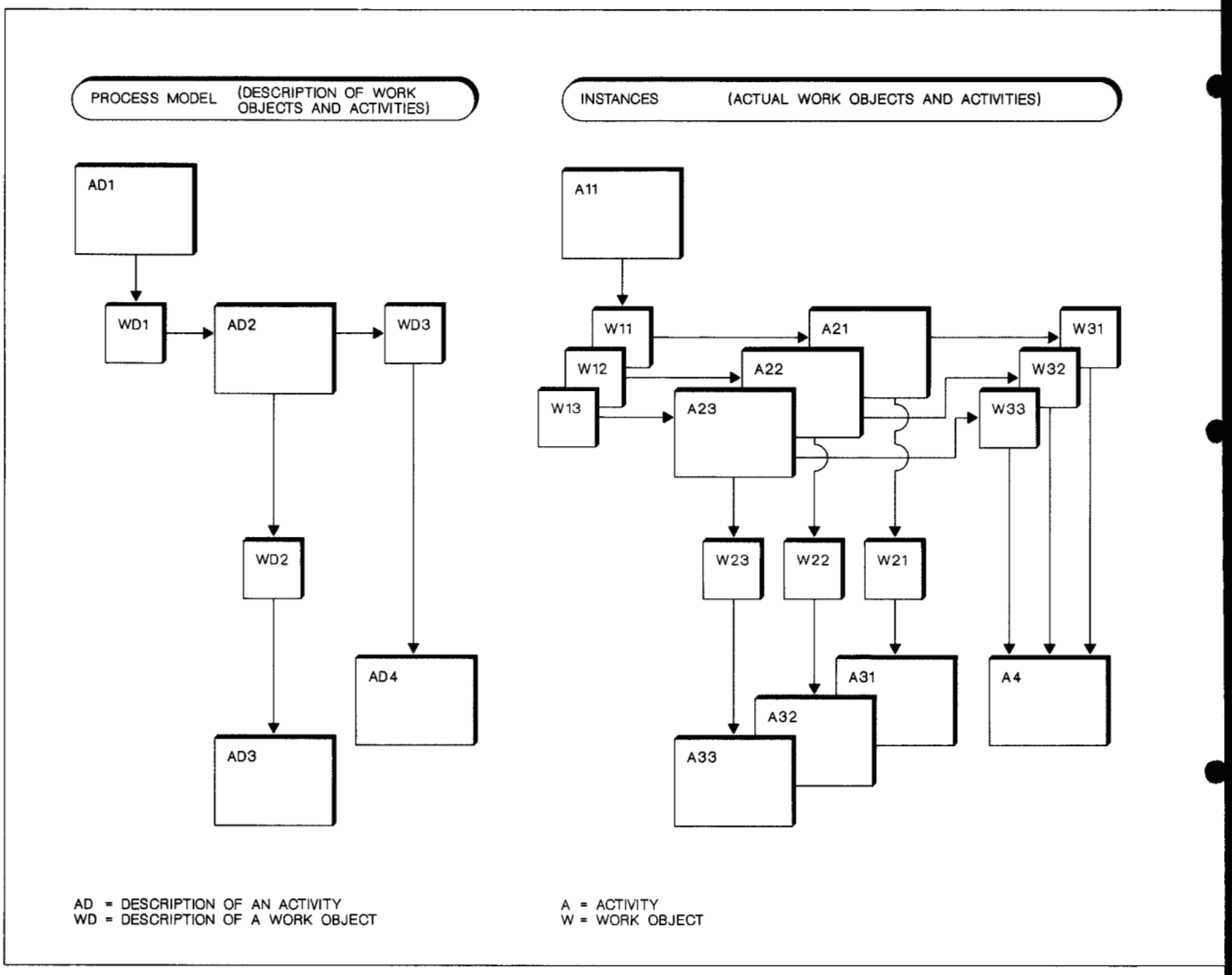


patible with the type of environment for which development is done.

Quality assurance is supported by the fourth model segment. It has to be interleaved with the respective development activities. Therefore the *quality assurance model segment* looks like a slightly pruned copy of the corresponding run-time application model segment. The quality assurance model segment can be tailored to resemble the run-time application model segment by applying the following two rules:[26]

- For every group of activities of the run-time application model segment, a quality assurance activity is defined to parallel it.
- For selected important activities of the run-time application model segment, a specific is additionally defined.

Figure 8 Activities, work objects, and their descriptions



Dynamically expanded model segments. In many instances a set of activities has to be repeated in several places. An example is creating an operational test environment. This obviously has to be done for unit test, and later almost the same set of activities has to be performed to create a system test environment. Therefore a process model has to provide means to expand a given activity into a chosen predefined model segment. This behavior is similar to a subroutine call in programming languages.

Work management

Models and their instances. An established application development process model is a template for future projects. Such a process model contains only descriptions of activities and work objects, not individual instances. Certain properties of the desired development process are defined in the model. Other properties, considered individual, are not described in the model and will vary from project to project.

Typically the model does not (but could) say how many instances of an activity or work object will be created in a specific development project (see Figure 8).

Project management. Even given an extensive process model there are many aspects outside of process management. Project management is concerned with embedding an application development process into a real-world environment, taking into account constraints and requirements posed by budget, time, manpower, risk,[30] etc. It is concerned with planning, tracking, and controlling any one individual project.

Project management has to rely on the contents of the underlying process model. Its findings influence the process inasmuch as additional constraints and considerations with respect to navigation arise (e.g., "Phase 3 should not be started before June first."), and additional tasks outside the project model (be it education, vacation, or interrelation to other projects) are taken into account.

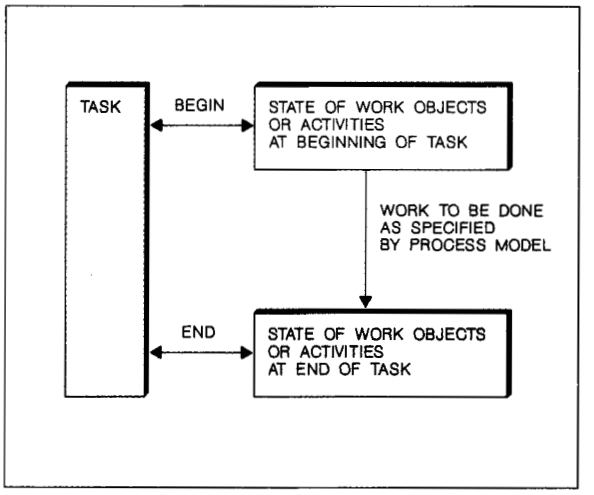
The unit of project planning and project control is called a *task*. A task identifies a certain amount of work to be done. It is associated with resources such as people, hardware, software, or time. Typically one records the planned and actual beginning of the task, the planned and actual end of the task, the responsible person(s), the resources to be utilized, and other pertinent information.

In general the beginning and ending of a task can be associated with states of selected work objects (see Figure 9). That is, the definition of a task includes the specification of an initial condition and a goal to be achieved. The process model describes how this possibly complex work step is to be done technically; project management defines it as a unit of planning for which certain resources are available and relates it to the other tasks within the project.

The granularity of a task (the size) depends on the characteristics of the project, the management style of the project manager, the type and experience of team members, the risk-level of the project, the enterprise culture, and many other factors. Project management literature[31,32] describes criteria for choosing the size of tasks. A task can be, for example, producing a complete design document or writing a source program until it is ready for inspection.

Figure 10 shows four work object types (SPECIFICATION, MODULE DESIGN, SOURCE CODE, PANEL SOURCE [see also Figure 3]) and their associated work objects (SPECS0, MDESIGN0, SOURCECODE1, SOURCECODE2, PANELSOURCE1, PANELSOURCE2). It also shows the associated tasks as chosen by the project manager. TASK1 and TASK2 are established for producing specifications and design, respectively. TASK4 and TASK5 are both related to producing source code and panels, respectively; each encompasses the production of two work objects. TASK3, representing education needed for TASK4, has no equivalence in the process model. It is introduced by the project manager. Project management may also introduce further dependencies between tasks (for example, education is only needed for TASK4).

Figure 9 Task and state of work objects



Work management for AD/Cycle. The Work Manager monitors application development based on the process model and on the current state of the various work objects. It identifies what is ready to be worked upon. It consults project management information as to whether further restrictions on the choice of actions exist. The Work Manager has to fulfill several functions and, as a consequence, several different interfaces can be identified (Figure 11).

Process definition and maintenance. A major task of the Work Manager is aiding to build models. This usually is a special component not accessible to a regular user. It involves the administration of the different model segments, their creation and modification, facilities tailoring model segments according to other model segments, and combination of model

Figure 10 Work object, activities, and their relation to project management

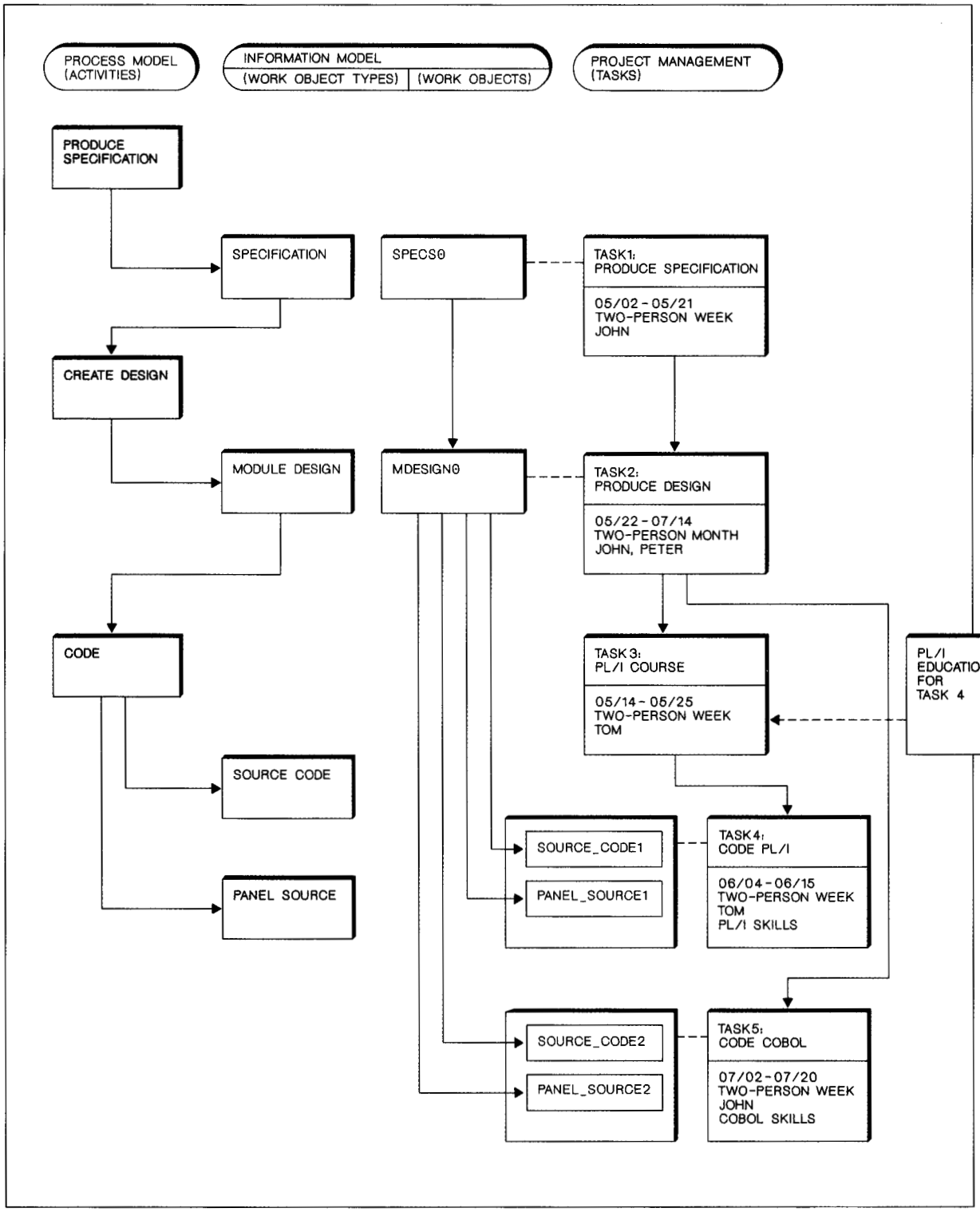
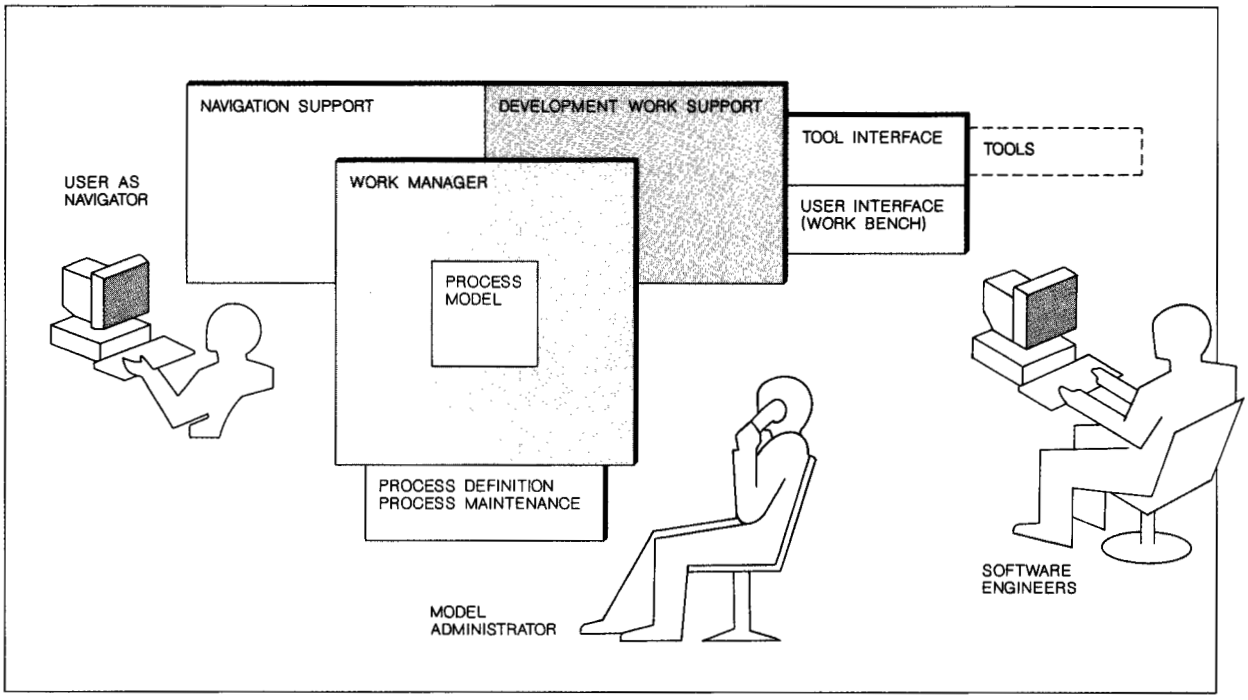


Figure 11 Components and interfaces of a Work Manager



segments (see Figure 7). To the regular user the application development process model should appear fixed. The Work Manager, however, can dynamically create the application development process model. Auxiliary functions would include comparing and copying model segments.

The Work Manager derives the process from the process model, interprets it, and presents it via a set of dialogs to the human user or invokes the tools integrated into the process (see Figure 12).

This user interface is used for the incorporation of both local changes, which are only relevant for one specific project, and permanent changes that reflect past experience. The latter allows the user to incorporate experience into the process model, which in turn makes the model a valuable asset of a company's development potential by establishing a certain software development culture.

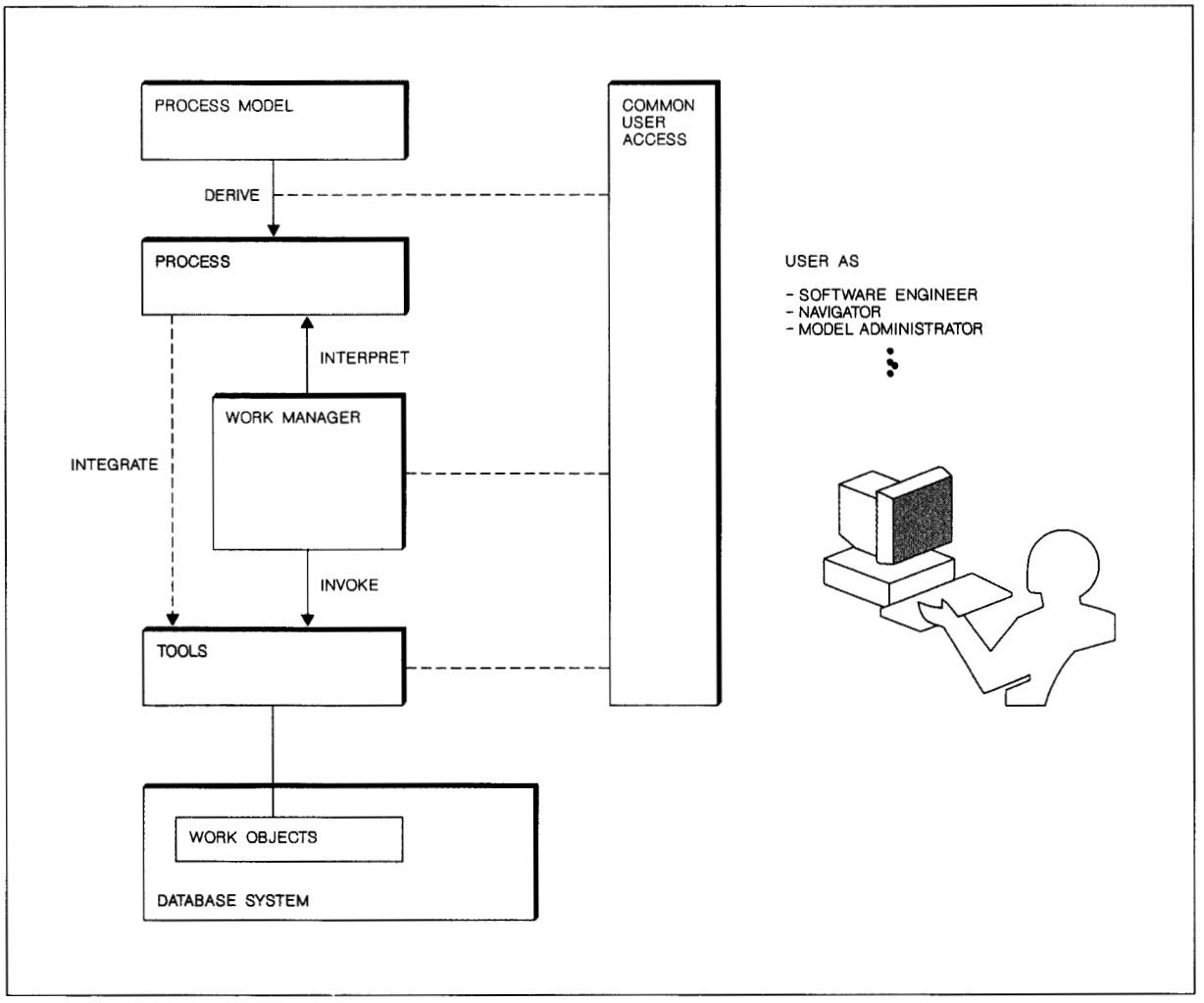
Performing development work. The Work Manager communicates with the agents of the application development process with software tools that perform some actions and with human beings who perform the creative tasks in development.

The use of tools is key to productivity. Tools have been employed from the very outset of software engineering, the earliest being assemblers and compilers. Incorporating the tool interface (see Figure 11) into the Work Manager relieves the user from considerable clerical detail. The calling mechanism is standardized and delegated to the Work Manager, and storage of work objects and their retrieval is performed under the control of the Work Manager and is thus mechanized.

The information contained in the process model is presented to the user in the form of a so-called "work bench," which allows access to the information, as shown in Figure 13. It provides on-line help texts and information about standards, for example.

Navigation. The application development process model is presented to the user through the workbench interface in order to let the user choose the next step(s). This navigation is subject to the constraints expressed in the process model, the data dependencies between the defined activities, the successor relations expressed in the process model, and the constraints imposed by project management. Furthermore the entry predicates and the exit asser-

Figure 12 Components of an application development project



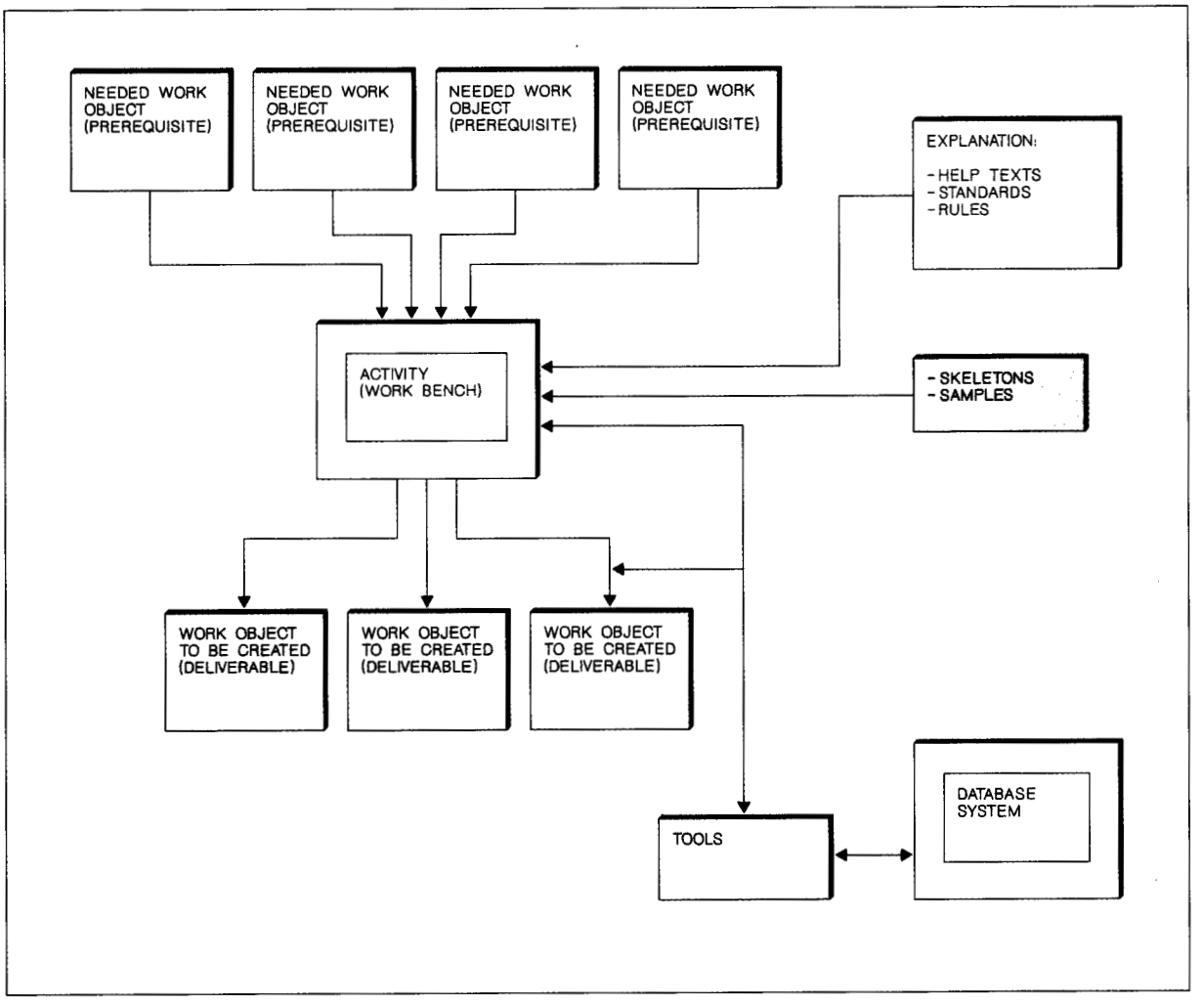
tions must be taken into account. Depending on the philosophy, the Work Manager will take more or less initiative in suggesting what to do next. The amount of control that the work process exercises over the end user varies from very restrictive to very liberal; that is, the to-do lists may prescribe the sequence in which items have to be dealt with, or may allow freedom of choice.

AD/Cycle work management in a cooperative environment. Figure 14 shows how work management fits into the AD/Cycle environment and illustrates how the host and the programmable workstation (PWS) cooperate.

The integrity and consistency of the overall application development process is maintained at the host. User interaction, on the other hand, is primarily handled by the PWS, following Common User Interface (CUA) rules[33] and exploiting the graphics capabilities of the workstation. Thus the Work Manager consists of two closely cooperating components: one on the host and one on the PWS. This cooperation is implemented by means of the communication facility between the workstation and the host.

Tools are invoked by the Work Manager, directly or indirectly, on behalf of a user. The Work Manager

Figure 13 Schema of a work bench



is sensitive to the return information reported back by tools and translates it into status information.

Host and workstation tools can be defined to the process model; tool invocation on the PWS exploits workstation services. The PWS can invoke host tools by requesting them as services from the host Work Manager, and vice versa.

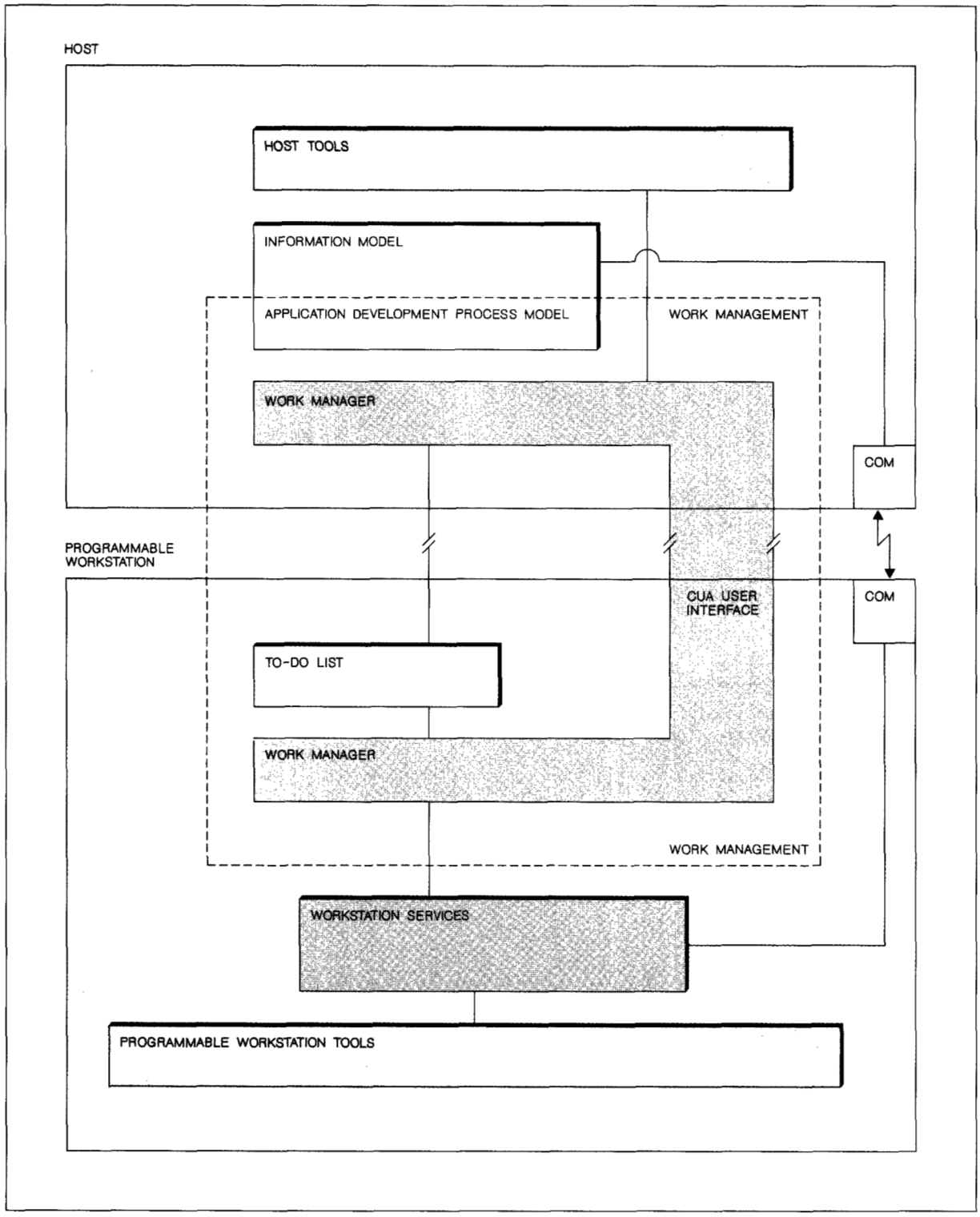
Summary

The pressing user requirements on the quality of software, and thus on the application development process, make an automated approach necessary. The clear definition of an application development process model, defining the steps to be followed and the work objects to be created, can be considered a necessary part of application development in addition to the separation of process and product function.

The requirement for computer-supported work management has arisen because of the complexity of the development process, the multitude of work objects, and the complexity and diversity of today's tools. A summary of the major requirements follows:

- Allow an application development process model to be defined, dynamically modified, and moni-

Figure 14 Cooperative application development work management



tored, measured, and improved. It has to be adaptable to specific enterprise needs.

- Allow individual processes to be derived from the application development process model and interpreted by a work manager. The system should allow for navigating through the process and for extracting to-do lists for individual users or user groups.
- Assist the user in performing the work by providing easy, automated, and standardized access to the work objects stored in the Repository Manager.
- Make it easy to access the right tools at the right time. Tools are used to implement the steps defined in the process model. This means that the role for each tool participating in the process has to be defined.
- The enforcement of the process should be adaptable to impose more or less discipline, as required for individual projects. Different styles of process management should be supported, from manual choice of the next step to the completely automatic navigation through the process.
- Provision of a project management interface for planning and controlling process resources is necessary.
- The system has to conform to the Systems Application Architecture™ (SAA™) requirements and follow the CUA standards. It must be integrated with AD/Cycle by relying on the AD information model. It should use the Repository Manager and be compatible with the AD/Cycle tool strategy.

Thus work management for AD/Cycle offers a total systems approach to application development, making the model of the process accessible on-line to the software developer, project manager, and other personnel involved. The use of work management results in a more systematic, professional, and transparent application development process, which in turn manifests itself in higher quality and productivity, and thus in reduced cost.

Acknowledgments

The authors acknowledge the contributions to the concepts in this paper by many researchers and developers in the IBM Corporation. Special thanks go to Kurt Bandat, Philip Joseph, Guenther Kalod, Roland Lutz, Dieter Mutschmann-Sanchez, Dick Phillips, Dieter Schoeberl, Franz Spickhoff, and Wolfgang Thumser.

Cited references

1. M. Gerisch and J. Schumann, *Software Entwurf*, Rudolf Mueller-Verlag, Koeln (1988).
2. G. Simons, *Introducing Software Engineering*, NCC Publications, Manchester, England (1987).
3. B. W. Boehm, *Characteristics of Software Quality*, TRW Series of Software Technology, North-Holland, Elsevier Science Publishers, Amsterdam (1980).
4. H. Sneed, *Software—Entwicklungsmethodik*, Rudolf Mueller-Verlag, Koeln (1986).
5. P. B. Crosby, *Quality Is Free*, Mentor Books/New American Library, New York (1980).
6. V. Merlyn and G. Boone, "Case Tools," *ComputerWorld* **23,** No. 13, 65–86 (March 27, 1989).
7. J. C. Wileden and M. Dowson, "International Workshop on the Software Process and Software Environments," *Software Engineering Notes* **11,** No. 4, 1–74 (1986).
8. *Handbuch fuer DV-Projekte-Methoden fuer Planung, Steuerung, Entwicklung und Betrieb von EDV Verfahren*, GE12-1473-1 (1978), IBM Corporation (out of print).
9. B. W. Boehm, "Software Life Cycle Factors," in *Handbook of Software Engineering*, C. R. Vick and C. C. Ramamoorthy, Editors, Van Nostrand Reinhold Company, Inc., New York (1984), pp. 494–518.
10. R. A. Radice, N. K. Roth, A. C. O'Hara, Jr., and W. A. Ciarfella, "A Programming Process Architecture," *IBM Systems Journal* **24,** No. 2, 79–90 (1985).
11. H. Bender, R. Fuhrmann, H. U. Kittel, B. Menze, J. E. Mueller, and D. Nadolny, *Software Engineering in der Praxis (das Bertelsmann-Modell)*, CW-Publikation, Muenchen (1983).
12. W. End, H. Gotthardt, and R. Winkelmann, *Softwareentwicklung—Leitfaden fuer Planung, Realisierung und Einfuehrung von DV-Verfahren*, Siemens AG., fifth revised and extended edition (1986).
13. L. J. Peters and L. L. Tripp, "A Model of Software Engineering," *Proceedings of the 3rd International Conference on Software Engineering* (May 1978), pp. 63–70.
14. G. F. Hoffnagle and W. E. Beregi, "Automating the Software Development Process," *IBM Systems Journal* **24,** No. 2, 102–120 (1985).
15. *Integrated Project Support Environments*, J. McDermid, Editor, Peter Peregrinus Ltd., London (1985).
16. H. Huenke, "Software Engineering Environments," *Proceedings*, Lahnstein, BRD, 1980, North-Holland Elsevier Science Publishers, Amsterdam (1981).
17. I. Sommerville, *Software Engineering Environments*, Peter Peregrinus Ltd., London (1986).
18. M. V. Zelkowitz, "Requirements for a Software Engineering Environment," *Proceedings of the University of Maryland Workshop*, May 1986, Ablex Publishing Company, NJ (1989).
19. J. A. Zachman, "A Framework for Information Systems Architecture," *IBM Systems Journal* **26,** No. 3, 276–292 (1987).
20. V. J. Mercurio, B. F. Meyers, A. M. Nisbet, and G. Radin, "AD/Cycle Strategy and Architecture," *IBM Systems Journal* **29,** No. 2, 170–188 (1990, this issue).
21. J. Sagawa, "Repository Manager Technology," *IBM Systems Journal* **29,** No. 2, 209–227 (1990, this issue).

22. M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal* **15,** No. 3, 183–211 (1976).
23. C. Chroust, "Software Development Paradigms—A Unifying Concept," R. Trappl, Editor, *Tenth European Meeting on Cybernetics and Systems Research*, Vienna (April 1990).
24. J. Blank and M. J. Krijger, "Software Engineering: Methods and Techniques," Wiley-Interscience Publishers, New York (1983).
25. 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).
26. G. Chroust, "Application Development Project Support (ADPS)—An Environment for Industrial Application Development," *ACM Software Engineering Notes* **14,** No. 5, 83–104 (1989).
27. R. W. Matthews and W. C. McGee, "Data Modeling for Software Development," *IBM Systems Journal* **29,** No. 2, 228–235 (1990, this issue).
28. *AD/Cycle Concepts*, GC26-4531 (September 1989), IBM Corporation; available through IBM branch offices.
29. *Application Development Project Support/Application Development Model and Process Mechanism—General Information*, GH19-8109 (April 1990), IBM Corporation; available through IBM branch offices.
30. B. Boehm, "A Spiral Model of Software Development and Enhancement," *ACM SIGSOFT—Software Engineering Notes* **11,** No. 4, 22–42 (1986).
31. P. W. Metzger, *Managing a Programming Project*, 2nd Edition, Prentice-Hall, Inc., Englewood Cliffs, NJ (1981).
32. D. J. Reifer, "Tutorial: Software Management," *IEEE Computer Society*, Catalog Number 81-85492 (1981).
33. J. M. Artim, J. M. Hary, and F. J. Spickhoff, "User Interface Services in AD/Cycle," *IBM Systems Journal* **29,** No. 2, 236–249 (1990, this issue).
34. G. Chroust, "Application Development with ADPS," *Softwaretechnik Trends* **9,** No. 3, 13–30 (1989).

Gerhard Chroust *IBM Programming Systems, Vienna Software Development Laboratory, Cobdeng. 2, A-1010 Vienna, Austria.* Dr. Chroust is currently a staff programmer, working on the definition of an application development process model for AD/Cycle. After joining IBM in 1966, he participated until 1970 in the creation of a formal definition of PL/I (Vienna Definition Language), becoming assistant to the laboratory director for the next five years. From 1977 to 1982 he was responsible for the mathematical/logical run-time library of the 8100 PL/I compiler. Since 1983 Dr. Chroust has had responsibility for the process model of Application Development Project Support (ADPS) and its successor products, coordinating the national language support for ADPS from 1987 to 1989. Dr. Chroust holds an M.S. from the University of Philadelphia and a Ph.D. from the Technical University of Vienna. Since 1980 he has served as Associate Professor at the University of Linz and as a lecturer on software engineering and microprogramming at several Austrian universities. He has published a book on microprogramming and computer architecture and has published numerous papers on firmware and software engineering. He is a member of the editorial board of the IBM Programming Series and a board member of both the Austrian Computer Society and the Austrian Society for Cybernetic Studies.

Helmut Goldmann *IBM Programming Systems, Vienna Software Development Laboratory, Cobdeng. 2, A-1010 Vienna, Austria.* Mr. Goldmann joined IBM in 1967 after graduating from the Technical University in Vienna with a Master's degree (Dipl.-Ing.) in physics. The areas in which he has worked include formal definition of PL/I (Vienna Definition Language) and of programming languages in general, programming system design, compiler construction (PL/I compiler for the 8100), and the implementation and architecture of software for application development support. He was responsible for the coordination of the model and driver components of the ADPS product. He is currently working on the modeling of the internal data of a process manager and related architectural and design problems.

Otto Gschwandtner *IBM Programming Systems, Vienna Software Development Laboratory, Cobdeng. 2, A-1010 Vienna, Austria.* Dr. Gschwandtner joined IBM in 1973. He was systems engineer in the large Multiple Virtual Storage (MVS) customer area. In 1977 he joined the Vienna Software Development Laboratory, where he worked on the Screen Definition Facility, various application development projects, and ADPS. He is currently a development manager, working on ADPS enhancements. Dr. Gschwandtner received a Ph.D. degree in mathematics from the University of Vienna. Before joining IBM, he spent one year as a visiting scholar at Stanford University and three years as teaching assistant and lecturer in the area of mathematical logic at the University of Vienna.

Reprint Order No. G321-5393.