Application Development Project Support (ADPS)

an Environment for Industrial Application Development

Gerhard Chroust
IBM Vienna Software Development Laboratory
Cobdeng. 2
A-1010 Wien
Austria

ABSTRACT

ADPS *(Application Development Project Support), developed in the IBM Vienna Software Development Laboratory, is an environment for the industrial development of application software. Crucial prerequisite for such an environment is the definition of a detailed process of how to proceed (a* **Process Model***) and an appropriate instrumentation via computer support (a* **Process Mechanism***) which not only helps the users to follow the established process but also provides the users with various support functions.*

*This paper puts the Process Model (***ADPS/M***) and the Process Mechanism (***ADPS/P***) into the broader context of current software engineering concepts. It explains principles and reasons for the architecture of ADPS.*

1.0 Basic Overview

1.1 Industrial Software Development

The observation that software development is an engineering process was made some 20 years ago at the famous conference at Garmisch [Naur_69] where actually the term '**software engineering**' was made widely known. The term was intended to emphasize the shift from the artistic world of genius-programmers to the mundane world of industrially produced software. At that time the hope prevailed that applying solid engineering principles would soon overcome the 'software crisis'. At the IFIP Congress 1986 F.Brooks [Brooks_86], however, shattered these hopes by claiming that no magic and no human toil will permanently eliminate the '*werewolf*' software. He cited four major reason for this state of affair:

- The **complexity** of software systems exceeds that of any other system devised by man so far.
- The requirement of **conforming** to and **interfacing** with already existing systems is much stronger than elsewhere.
- The **changeability** of software undermines the stability of existing systems causing interface problems.
- The **invisibility** of software products prevents an intuitive, 'common sense' handling and differentiation of the various components of a system.

Accepting Brook's analysis shifts our attention from the search for an ideal uniform solution to the much less glamorous path of trying to attack the software crisis on all possible levels hoping for a synergetic effect in our attempts [Jackson_82] [Lehman_85].

The following theses can be used as a starting point:

1. The quality of an industrial product cannot be established after the facts but only by establishing and adhering to an appropriate development process [Kraft_77].
2. In order to establish a meaningful control, the process must be specified to a sufficient level of detail and must encompass all necessary activities.
3. The defined process must be applicable to a wide variety of individual projects. The process must be describable, verifiable and subject for validation [Wileden_86]. This implies a certain formality in its description [Neuhold_85a].
4. Due to the fallibility of human nature a rigorous observation of the process can only be ensured by computer controlled stepwise execution of the process.
5. The process must accommodate change in order to function as a repository of a company's software development culture and experience.
6. The software development engineer is still by far the most expensive and scarce resource in application development [Kraft_77]. Improving his/her productivity promises the highest pay-off [Boehm_84b].
7. Productivity can only be achieved by software tools. A prerequisite is that the attachment and use of these tools does not require extensive effort from the user. This implies that the system offers a consistent tool interface [Akstaller_86] [Dolotta_76].
8. The complexity of the software process [Brooks_86] requires adequate guidance and instruction to be provided by the system. A basic requirement is that help information be available with undue effort at the point of need.

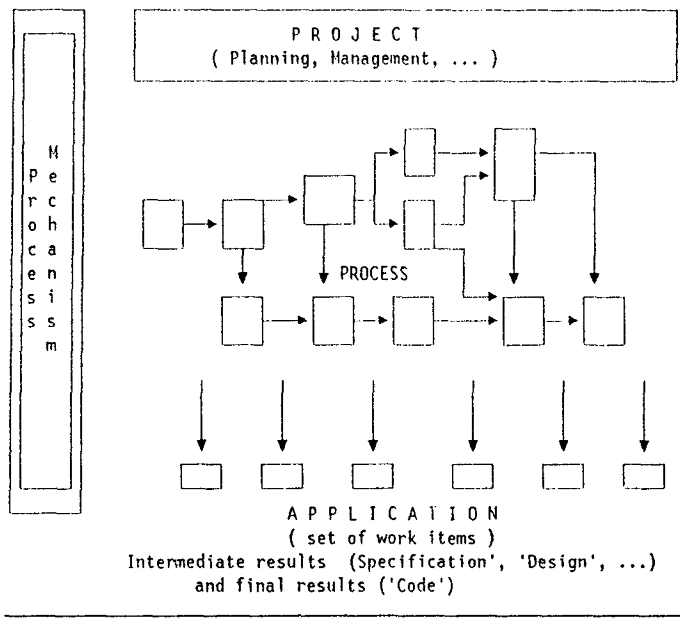
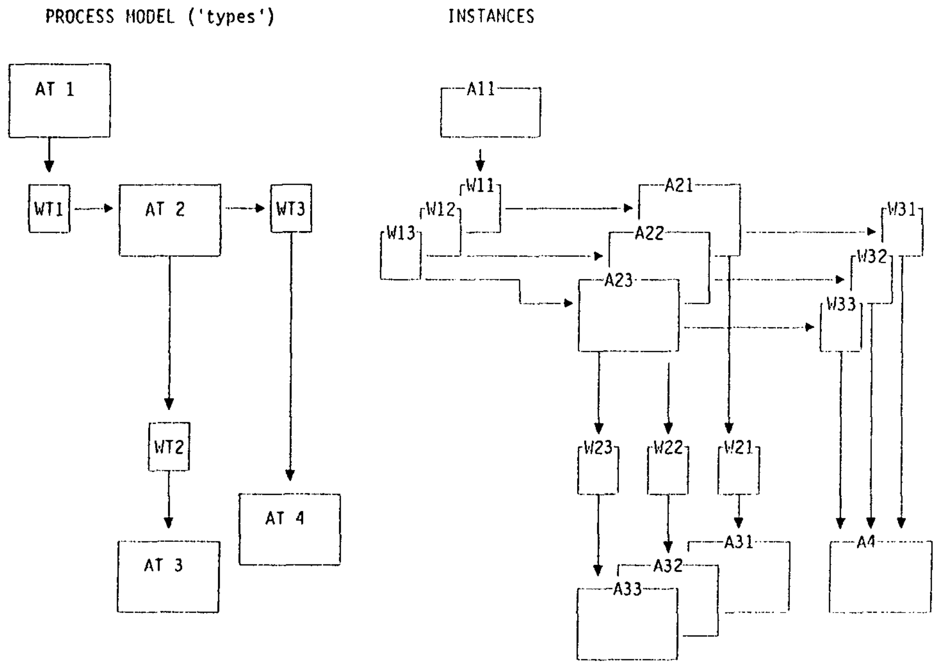

PROJECT
(Planning, Management, ...)
Process
Mechanism
PROCESS
APPLICATION
(set of work items)
Intermediate results (Specification', 'Design', ...)
and final results ('Code')

Figure 1. Product, Project, Process and Computer Support

Software, like any other industrial product, has to be produced by an industrial process. One of the yardsticks for the maturity of an industry [Crosby_80] [Zemanek_80] is the amount of abstraction in the description of the processes to be performed and subsequently the amount of automation resulting from the abstraction [Abbott_87]. Abstractions allow the separation of the common kernel in each project from the individualistic properties and peculiarities of any one project. The major abstraction is the separation of the properties of the **product** (the **'application'**) to be created from the **process** by which it is created (Figure 1). Once this separation is achieved, the **Software Development Process** can be described and discussed separately [Wileden_86]. An abstract description of a process usually lends itself to a formal description [Neuhold_85a] which in turn can be submitted to an appropriate interpretation mechanism (a **Process Mechanism**) which can control and partially automate the steps of the process. Once a computer supported Process Mechanism is in place further support can be given by the computer. Supplying these additional computer services creates an **Integrated Project Support Environment** (IPSE) [McDermid_85], also called **Software Engineering Environment** [Brereton_88] [Hausen_87] [Sommerville_86].

1.2 The Notion of a Software Development Process

One of the properties of industrial processes is that they are essentially repeatable with different people for different end-products. Essentially the process is an abstraction of many different processes containing all the relevant common actions and ignoring local differences [Lehman_85]. It is important to recognize that a Process Model just describes the *types* of activities to be performed and the *types* of results to be produced (Figure 2).



A ... Activity (Instance)
AT ... Activity Type
W ... Work Item (Instance)
WT ... Work Item Type

Figure 2. Activity types, work item types and their instances

The Software Development Process represents a large amount of far-reaching theoretical and practical considerations about the way software should be built and it is thus an expression of a company's *'software development culture'*. Existing process models [Hausen_87] [Hausen_87b] [Peters_78] show considerable differences both with respect to the suggested methods and to the means of description. Some of the more important considerations when designing a Software Development Process are:

Basic Components: According to common understanding we currently see a Software Process (Figure 2) as the description of a set of **activity types** to be performed, a set of **work item types** to be produced and a description of their relation (input/output).

Granularity: It is important to choose the appropriate granularity: a coarse granularity makes a process model applicable to most actual processes but includes the danger of not specifying the desirable process to a sufficient detail. Too fine a granularity might subdue the developer too much, forcing an undesirable straight-jacket on him.

Breadth of Model: The development process can be seen very narrow, just describing those activities which are essential for creating the application. It may, on the other hand, also describe supporting activities like quality control, project management, product marketing etc.

Type of target environment: The more diverse the anticipated target environments are the more complicated and/or non-committing a model has to be. Differences between batch and on-line systems, between conventional implementation languages and fourth generation languages, between imbedded systems [Chroust_88c] and user-oriented transaction systems etc. should reflect themselves in the model.

Prescription of Principles and Methods: The model designer has to decide to what extent principles and methods are pre-defined in the model, whether one or several methods are available at each step.

Sequentialization: The model has to specify to what extent the sequence of activities is prescribed and what freedom of choice the developer still has in choosing an order of execution.

Tool description: Rigorously defined methods can be translated into tools. A process model can also identify the tools to be used for the various activities, reflecting the use of this tool in the terminology and/or in the structure of the model.

Structuring and Numbering Scheme: Both activity types and result types, beyond a certain level of complexity, need a logical structure. Major (to some extent contradictory) criteria for structuring are: the logical connection of result types, the time sequence in which results are created, the sequential order of activity types, etc. An important structuring criterion can be established by an adequate numbering scheme. Usually it is desirable to reflect several ordering criteria in the numbering scheme, making it necessary to weight them. The restricted set of available digit positions and the restricted value range for digits make compromises necessary.

1.3 Instrumenting a Software Process

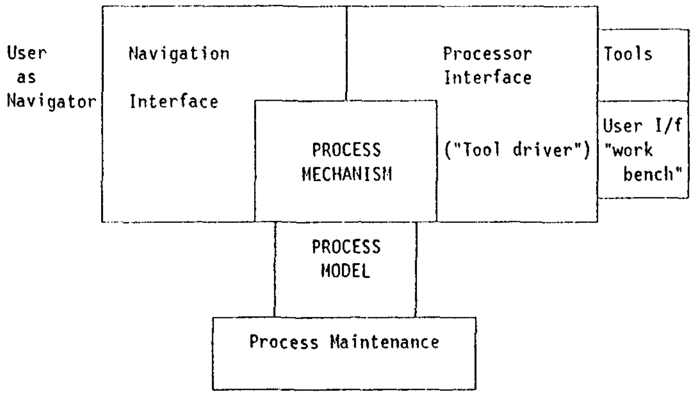


Figure 3. Interfaces of a Process Mechanism

Describing a Software Development Process is not sufficient to ensure adherence. Such descriptions have been prescribed in books and project manuals [Bender_83] [End_86] for many years but they are

- remote from the point of actual usage,
- cumbersome to be looked up,
- difficult to be maintained.

The obvious solution is to make them better accessible via on-line support, especially since the actual software development is also done via a computer. This leads to an arrangement where the description of the development process is accessible to the software engineer on-line and where the computer is able to provide guidance and advice (Figure 3). The Process Mechanism has two essential interfaces: one to the software engineer in the function of navigator *('manager of the process')*, deciding on the next actions to be performed, the other to the actors, be it tools or a software engineer, to create the intermediate or final results *('actor in the process')*.

1.4 Integrated Project Support Environments

The computer support *("instrumentation")* of the development process generates the desire to improve and increase the computer support. Some of these additional requirements are:

Support of the Development of the Application
(This includes programming in the narrower sense, i.e. the provision of compilers, editors etc.)

Support of Education and Professionalism
(The user may request help texts, information about standards, may be supplied with skeletons to be filled in)

Support of Product Quality
(Especially the mechanical side of verification can be handed over to the computer)

Support of Result Administration
(It is necessary to administer a tremendous amount of individual items, and this usually in several versions, incarnations etc.)

Support for Navigation
(The user needs information about what actions can be performed next and in what order)

Support of Project Analysis, Planning and Control
(Since all results are stored in the computer and all activities are performed in cooperation with the computer also the necessary management information is available and can be used for project management).

Providing these support functions to a reasonable extent establishes a so-called **Integrated Project Support Environment ('IPSE')** [Dolotta_76] [IBM_87]. The major components (Figure 4) are:

A formal description of the development process (the Software Process Model): The description must be sufficiently formal in order to be input to the Process Mechanisms and it must be sufficiently detailed to provide an appropriate guidance in following the process.

Navigation: It keeps track of activities ready for execution, helps in deciding on the next steps and allows triggering selected activities.

Tool Attachment: It allows the access and use of tools in a fairly standardized way, providing a standardized calling interface to individual tools and a uniform user interface.

Work Item Library: It takes care of the storage/retrieval of the work items created/handled during the project.

Relationship Store: It records the relation between the individual work items, i.e. how they depend on one another.

On-line Help and Explanations: These can be called upon any time explaining to the user possible actions etc.

User-Interface: It presents relevant views of the Process Model to the user of the IPSE allowing to interact with the system (Figure 3).

Process modification: The description of the development process reflects a considerable amount of individual user and company experience and is thus a valuable asset of a company. Since the Software Process is an abstraction of a large set of processes both small, local adaptations for an individual project and long-term changes to incorporate new methods and views are necessary.

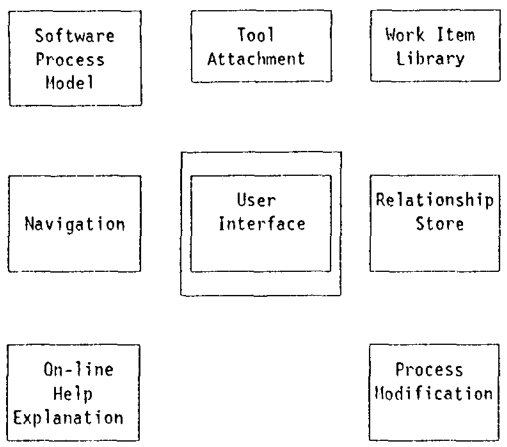


Figure 4. Constituents of an IPSE

2.0 The Structure of the ADPS Development Process

The reader should bear in mind that ADPS as delivered [Chroust_88g] [IBM_87] can only be the basis for an individual model adapted to the actual requirements of a specific enterprise (see also 3.5, "Adapting ADPS").

2.1 General considerations

Based upon experience with predecessor products like OPSS [Akstaller_86], VIDOC [IBM_86c], and COMMAND [IBM_86] the following major objectives and requirements for ADPS were identified:

Independence of specific methods and languages: ADPS should describe and model the development process in such a way that it is applicable to a large sector of the user population. This implies that the description should refrain from postulating specific methods or description languages. Typically project management should be general enough to fit many existing project management tools.

Adherence to established principles: Established principles of good software development should reflect themselves in the structure of the model. Some of these principles are

- separation of user view and technical solution,
- development by stepwise refinement,
- quality assurance in parallel with development,
- documentation as part of development,
- interleaving of project management and development.

The activities should be broken down to the smallest meaningful steps: The granularity was chosen (and this admittedly is a somewhat subjective decision) such that activity types and work item types represent relative detailed but still meaningful units.

Inclusion of documentation, project management and quality assurance: Based on the experience of its predecessor products it was found necessary to present an integrated model which not only describes the creation of the machine-executable code but also activities which concern themselves with producing the necessary documentation of the product, which model quality assurance and project management.

Numbering: Following the predecessor products one character and 4 essential digits were used to identify all work item types and activity types (When modifying the model the user may use 5 and 6 digits, respectively). The character was used to designate the path (see 2.2, "Basic structure - the paths"), the 4 digits were predominately used as a hierarchical classification scheme (Figure 5).

The numbering scheme for work item types was based on a *logical* order. Work item types of the same semantics received numbers close together even if their creation time was widely apart. Activity types were primarily numbered according to their *sequential* order of execution.

Comprehensiveness of the Model: The shipped model is intended to be a 'maximal' model in the sense that most relevant development activities are included in the model. The user rather has to delete certain parts of the model than to add new parts.

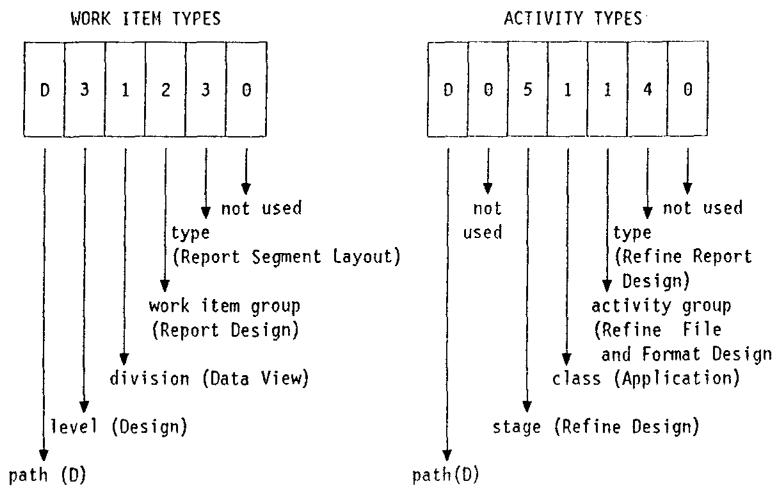


Figure 5. Numbering Scheme for Work item types and activity types

2.2 Basic structure - the paths

Modern software development is characterized by a cooperative team work of many specialists with different responsibilities. A major division, usually also reflected in the organizational structure of an enterprise, can be found between development proper, project management and quality assurance. These activities are usually handled by different people with different authorization and organizational position. ADPS reflects this in defining three different **paths**:

D(evelopment)-path: The set of those activities in the development model which are concerned with the actual development of the application and its associated documentation.

Q(uality Control)-path: The set of those activities which are concerned with the validation/verification of the application.

P(roject Planning and Control)-path: The set of those activities which are concerned with the management of the project.

Figure 7 shows the interplay between the different paths: based on planning in the P-path, certain activities of the D-path take place, the work items resulting from it being subjected to a Q-path activity. The information generated in the Q-path is feed back to the P-path whence further D-activities are planned etc. The statistical distribution of activity types and work item types over the paths is shown in Figure 6.

Path	Activity type	Work Item type
D	88	122
P	30	43
Q	63	15
SUM	181	180

Figure 6. Count of activity types and work item types

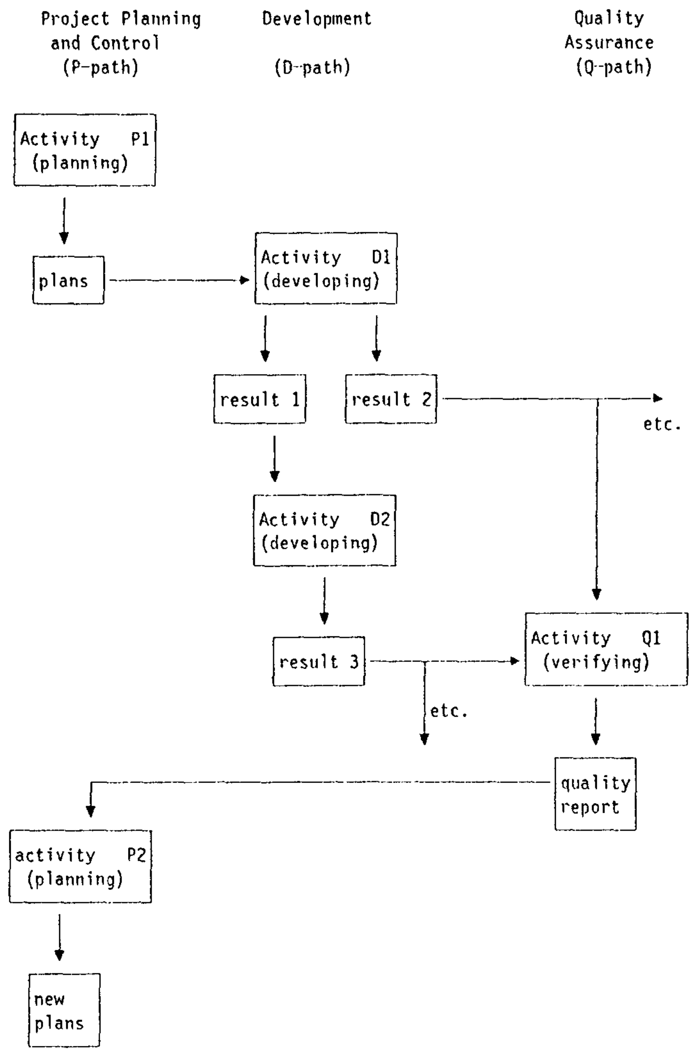


Figure 7. Different paths in SW-Development

2.3 The Development Path (D-path)

2.3.1 Work item types of the D-path

A software application cannot be conceived in one step as one solid piece of code because the gap between the problem to be solved and the final implementation is too wide [Lehman_80] [Lehman_85]. It is therefore necessary to describe the intended application at different levels, with different distance to the actual computer implementation. The concept of levels allows to develop the application in a stepwise fashion, concentrating on one set of issues at one time. Successive transformations finally yield the application in an executable form:

Requirements describe the overall expectations against the system, the boundaries of the anticipated application and the interfaces to the current system.
Model describes the business processes used by an enterprise as seen by the user.
Design defines how the application will automate selected business processes of the Model
Code is the transformation of the Design into a machine executable form.
Integrated System collects all parts of the application, ready to be used
Installed system represents the application put into work on a user machine.

division / level	PROCESS	DATA		DATA FLOW
		DATA VIEW	DATA ORGANISATION	
MODEL	Function Definition	Information Definition	Data Model	Data Flow/ Model
DESIGN	Program Design	Format Design	File Design	Data Flow/ Design
CODE	Programs	Formats	Files	

Figure 8. ADPS Kernel Work Items Classes and their Structure

Another way to classify the work items of an application is by identifying their different functions. In this respect we distinguish eight **divisions** of an application:

Processes describe the business transactions which are to be automated by the application. On the level of Model these are the transactions as seen or intended by the customer, e.g the handling of an order. On the level of Design the technical solution is described which has been chosen, e.g. the chosen algorithm. On the level of Code it represents the computer-executable representation of the transaction.

Data Views describe the form and the external appearance of the data which are processed by the *Processes*. On the level of Model it is the view of data (forms, panels, reports) as seen by the customer. On the level of Design the data are assigned to different media (screens, data bases, etc.). On the level of Code it comprises all necessary data definitions, control blocks etc.

Data Organization describes how permanent data are to be stored in the system. On the level of Model entities and their relations are defined, on the level of Design the form and type of data bases, file systems, etc. are described, while the Code level comprises all data base and file specifications and their associated access code.

Data Flow describes explicitly the association between *Processes* and *Data Views* or *Data Organization.*

Messages and Help defines all information offered by the machine executable application. It comprises error messages, help panels etc. Providing a separate division for it was to stress its growing importance for the man/machine interface.

Product Documentation is to be delivered together with the machine-executable application. It is an important, but independent part of the deliverables of a software project. The inclusion of this intends to cater for an integrated development of code and documentation, avoiding the post-shipment documentation-writing syndrome.

Application Environment collects all information about the interface between the application to be developed and its environment, recognizing the fact that practically all applications are part of a greater enterprise system. Both influences from the surrounding systems on the application and impacts of the application on its environment are recorded.

Development Support comprises mainly data and code for tests, but also generator code to generate the actual application etc.

Beyond structuring by level and division, work item types are collected into **groups** usually reflecting a logical alternative (e.g. flat file versus data base etc.). The **type** represents either

the amount of refinement (over-all, unit, detail)		or a logical affinity	
D22110	*Function*	**D32150**	*Access Module Specification*
D22120	*Function Group*	**D32160**	*Process Module Specification*
D22130	*Subfunction*	**D32170**	*Presentation Module Specification*
		etc.	

The names of the most important classes of work items (resulting form combining level with division) are shown in Figure 8. Figure 9 shows a small section of the list of all work item types as they appear on the screen.

Individually created work items carry with them also a set of attributes with auxiliary information (owner, applicable tools, associated library, creation date etc.).

2.3.2 Activity Types of the D-path

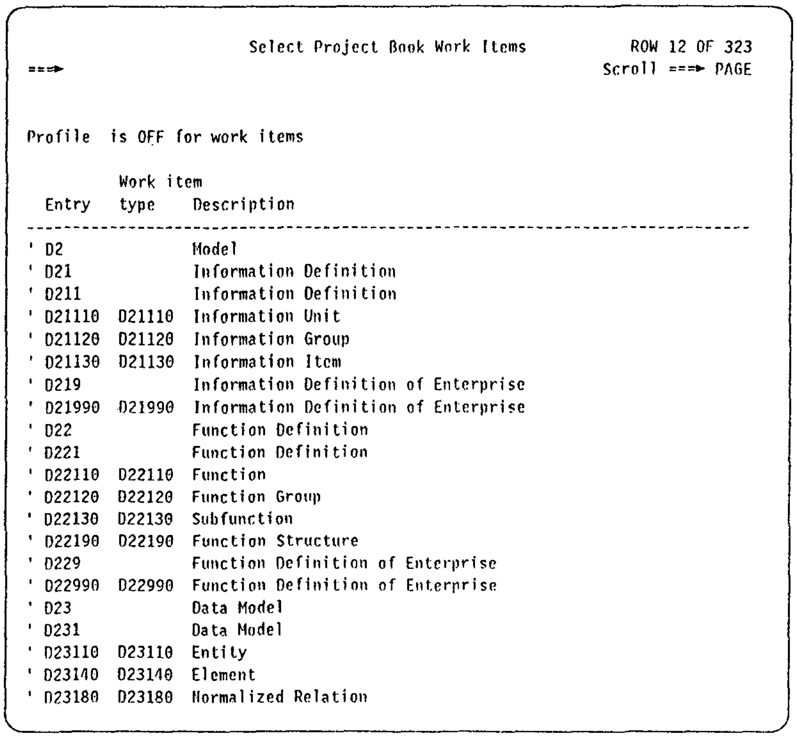
P H A S E	Management question	S T A G E
Define Model/Design	Is the application feasible and profitable?	Define Requirements
		Define Model
		Define Design
Refine Model/Design	Is the application technically sound?	Refine Model
		Refine Design
Produce Code	Is the application implemented?	Produce Code
Integrate and Install System	Does the application run?	Integrate Code
		Install System
Monitor System	Is the application successful?	Monitor System

Figure 10. The ADPS phase and stage structure

The development process' basic philosophy is reflected in the logical structure of the work items. The suggested order of their creation is a second major aspect of a development model. Here ADPS follows well established paradigms by introducing **phases**, **stages** and **groups**.

Phases provide the basis for management control of the development process. At the end of each phase a major question can be answered and associated decisions made (see Figure 10).

Stages are subdivisions of phases, accomplishing one important technical milestone. They correspond closely to the levels of work item types. It can be seen that *Model* and *Design* each are created in two steps: in stages 2 and 4, and in stages 3 and 5 respectively. The reason is that an evaluation of the feasibility and profitability can only be based on both *Model* and *Design*. On the other hand one should not invest too much effort into a *Model* before being certain about the feasibility of the project and about the stability of the requirements, and both may depend on the outcome of phase 1.



```
                         Select Project Book Work Items          ROW 12 OF 323
===>                                                             Scroll ===> PAGE


Profile  is OFF for work items

         Work item
  Entry   type    Description
-----------------------------------------------------------------------------
' D2              Model
' D21             Information Definition
' D211            Information Definition
' D21110  D21110  Information Unit
' D21120  D21120  Information Group
' D21130  D21130  Information Item
' D219            Information Definition of Enterprise
' D21990  D21990  Information Definition of Enterprise
' D22             Function Definition
' D221            Function Definition
' D22110  D22110  Function
' D22120  D22120  Function Group
' D22130  D22130  Subfunction
' D22190  D22190  Function Structure
' D229            Function Definition of Enterprise
' D22990  D22990  Function Definition of Enterprise
' D23             Data Model
' D231            Data Model
' D23110  D23110  Entity
' D23140  D23140  Element
' D23180  D23180  Normalized Relation
```

Figure 9. Work item types of Level 2 ("Model")

Groups are sets of activity types which produce a coherent set of associated work items (e.g. the group D512 *Refine Program Design* contains the activity types (Figure 14):

D051210 *Specify Access Module*
D051220 *Specify Presentation Module*
D051230 *Specify Process Module*
D051240 *Specify Module Control Flow).*

Activity types create one or a small number of closely related work item types.

In detail the following work is done in the individual stages:

Stage 1, Phase 1 (*Define Requirements*): The technical and organizational premises as established by the project contract are analyzed. Their useability, understandability and acceptability are investigated. Together with those pieces of information which already exist in the enterprise (e.g. the **Global Data Model**) the technical requirements for the project are derived (Figure 11).

Stage 2, Phase 1 (*Define Model*): The user view (the **Model**) is specified without working out all the details (this is reserved for stage 4). Starting with the the **Information Units** and **Information Groups** (i.e. the *data view*), the **Entities** in the **Data Model** (i.e. the *data organization*) are established. The **Function Definition** is specified by defining **Function Groups** and **Functions**. Thus they are viewed as transformations of data and are defined only after having defined the data. The **Data Flow** of the application (which is implicitly contained in the definition of the **Function Definition**) is explicitly shown. At the end of this stage the external view of the application (the **Model**) is established in its general form (Figure 11).

Stage 3, Phase 1 (*Define Design*): In this stage the basic decisions about the technical solution of the application are made. Based upon the **Model**, as far at it has been established in stage 2, **Information Definitions**, **Function Definition**, and the **Data Model** are mapped onto their computer realizations, i.e. on **Format Design**, **Program Design**, and **File Design**. The choice of a technical solution (the **Design**) includes choosing to perform certain activities manually or by direct interaction with the end user (e.g. spread sheets operations). If a designer creates temporarily more than one **Design** for the application, the decision between these alternatives is explicitly made at the end of this stage. At the end of stage 3, and thus of phase 1, both **Model** and **Design** are established down to a certain level of detail. This allows a rational management decision as to whether to continue with the project or not (Figure 11).

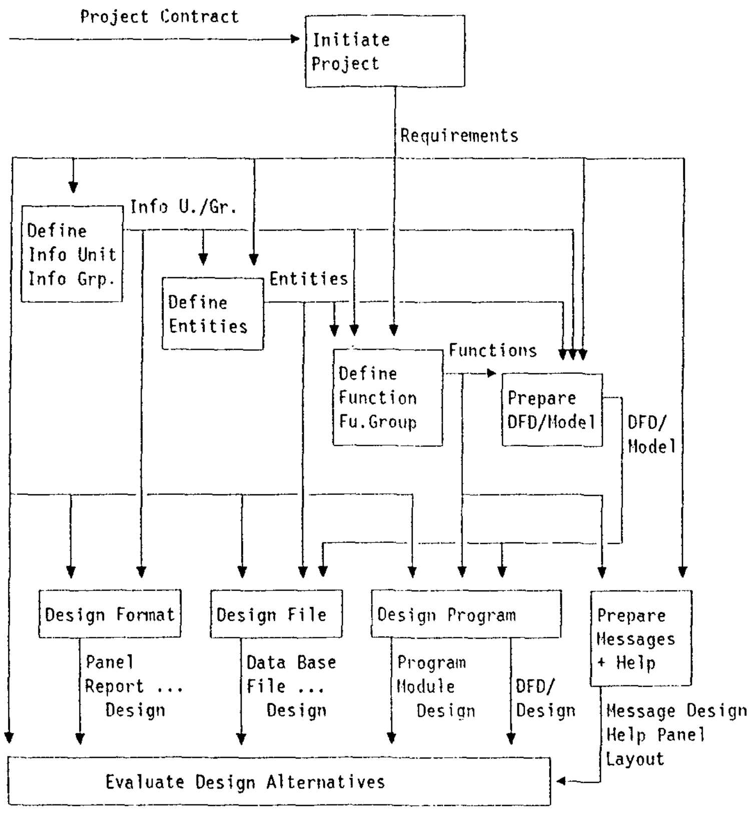


Figure 11. Overview of the ADPS development path, phase 1

Stage 4, Phase 2 (*Refine Model*): The **Model** is completed in all its details. The activities are analogue to stage 2 (Figure 12).

Stage 5, Phase 2 (*Refine Design*): The technical **Design** is finalized based on the **Design** existing so far and on the details of the **Model**. All technical details are established as far as they are not directly associated with **Code**. Appropriate modules are designed for accessing files and data bases and for presenting panels. The hierarchical structure of the modules is converted into a **Program Hierarchy** which defines the calling structure of the individual modules. At the end of this stage all technical decisions have been made. This allows verification whether the designed application is consistent and fulfills the **Requirements** (Figure 12 and (Figure 14).

Stage 6, Phase 3 (*Produce Code*): The detailed **Design** is converted to machine-executable **Code** (compilable or interpretable), including **Control Blocks** and **Formats** for file and database access. Sections of the application which are to be performed *manually* or via *end user processing* are separated: their 'implementation' corresponds to a (more or less) detailed descriptions of the necessary steps and the specification of the necessary files and archives (Figure 13).

Stage 7, Phase 4 (*Integrate Code*): The complexity of software applications requires considerable test effort. Therefore **Integration Tests** by independent test teams are performed on the integrated system.

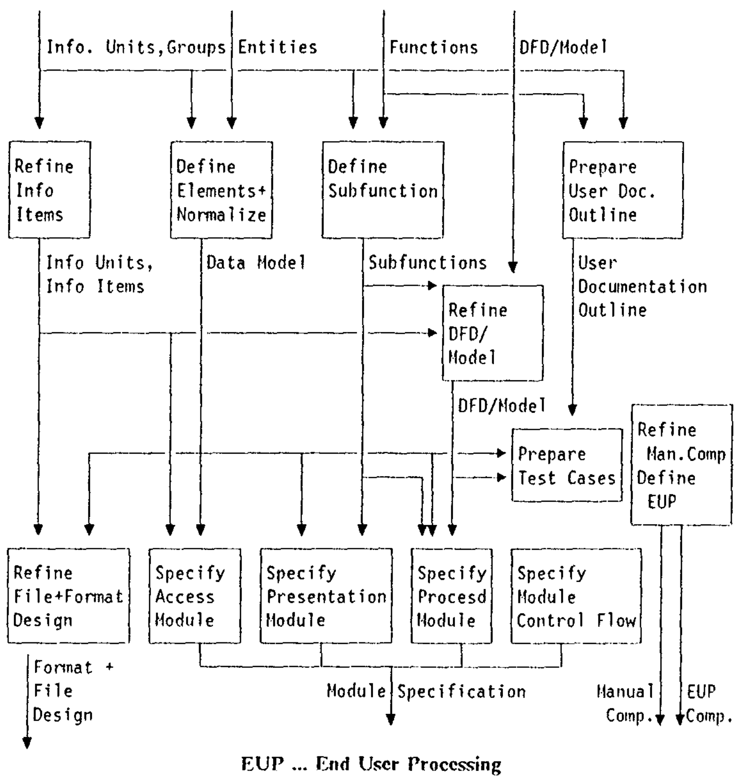


Figure 12. Overview of the ADPS development path, phase 2

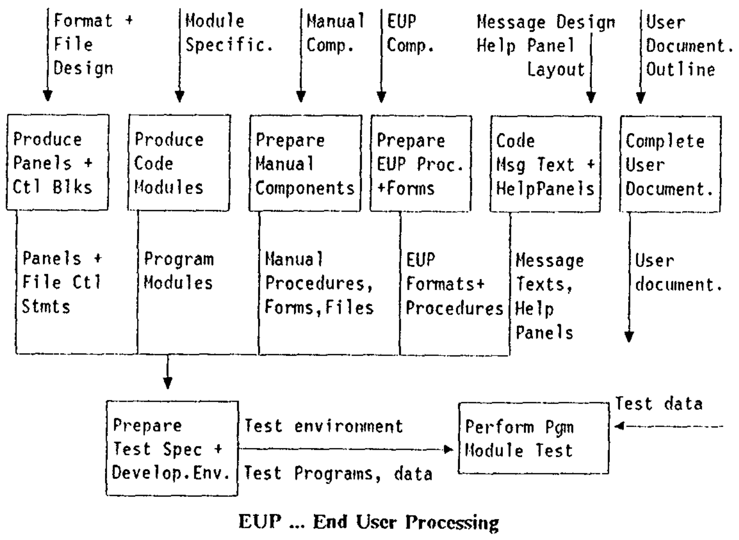
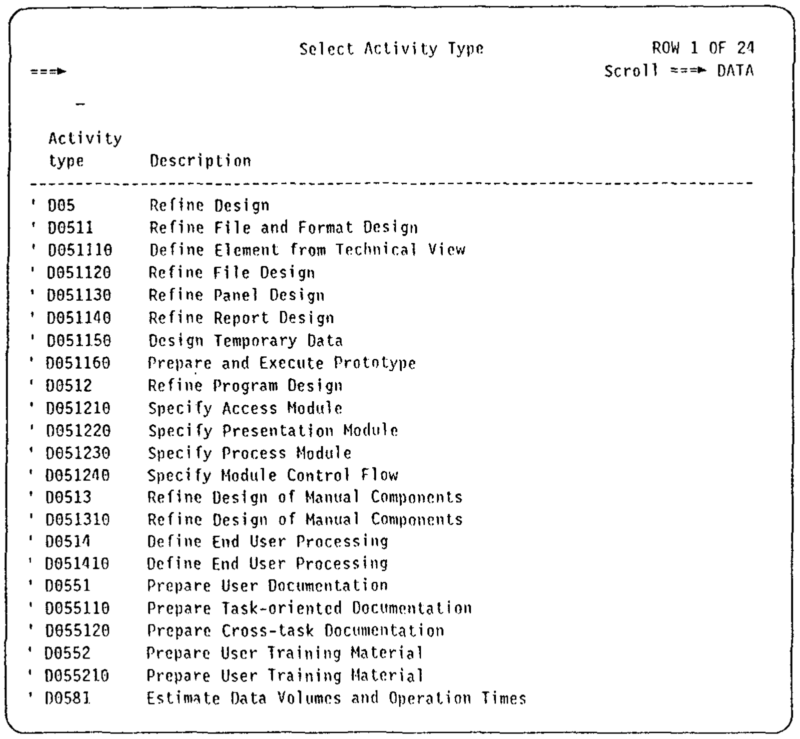


Figure 13. Overview of the ADPS development path, phase 3

Stage 8, Phase 4 (*Install System*): Beyond a certain complexity of an application its **System Installation** needs complex procedures and extra verification and tests. This includes conversion/migration of data sets and the like.

Stage 9, Phase 5 (*Monitor System*): After installation certain tuning and adaptations may be necessary, together with some counselling and support for the end-users. User requirements and complaints are collected in order to be routed to the creation of the next version of the application.



```
                        Select Activity Type            ROW 1 OF 24
===>                                                   Scroll ===> DATA
  -
 Activity
 type       Description
---------------------------------------------------------------------------
' D05       Refine Design
' D0511     Refine File and Format Design
' D051110   Define Element from Technical View
' D051120   Refine File Design
' D051130   Refine Panel Design
' D051140   Refine Report Design
' D051150   Design Temporary Data
' D051160   Prepare and Execute Prototype
' D0512     Refine Program Design
' D051210   Specify Access Module
' D051220   Specify Presentation Module
' D051230   Specify Process Module
' D051240   Specify Module Control Flow
' D0513     Refine Design of Manual Components
' D051310   Refine Design of Manual Components
' D0514     Define End User Processing
' D051410   Define End User Processing
' D0551     Prepare User Documentation
' D055110   Prepare Task-oriented Documentation
' D055120   Prepare Cross-task Documentation
' D0552     Prepare User Training Material
' D055210   Prepare User Training Material
' D0581     Estimate Data Volumes and Operation Times
```

Figure 14. Activities of Stage 5 ("Refine Design")

2.4 The Structure of the Q-path

The Q-path contains all activities and results which ensure an adequate quality of the application. The structure of the Q-path is systematically derived from the D-path. Quality assurance is performed in parallel with actual development (Figure 7). Therefore in ADPS no stage is explicitly devoted to testing (in contrast to many other models, e.g. [Boehm_80] [Boehm_84] [Peters_78]).

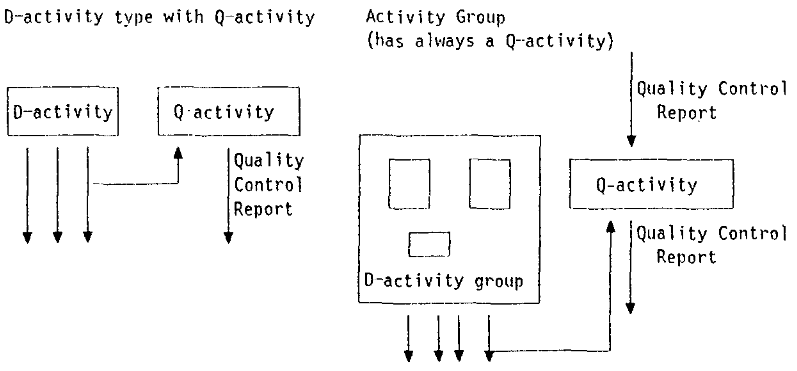


Figure 15. Systematics of Q-Activities

Analyzing the D-path in ADPS one finds that for some activity types it is good practice to subject their outcome to quality assurance while for others this seems too low level a verification. This implies that the Q-path looks like a selectively pruned D-path: For selected *D-path activity types* (35 of 88) there exists a Q-activity type which has all the output work items of the respective D-activity type as input. The output of the Q-activity is a Quality Control Report (Figure 15). In order to achieve a uniform quality assurance for the complete D-path it was decided to additionally include a Q-path activity types for each *D-path activity group* (Figure 15). This allows to collectively validate results created in the activity types of this group.

2.5 The Structure of the P-path

An integral ingredient for a successful software project is adequate planning and control [McClure_81] [Metzger_81]. Therefore the P-path was integrated into the development model of ADPS. The multitude of existing project management tools together with the wish to keep ADPS comparatively free from specific methods or tools induced a rather symmetric, combinatorial view of the P-path. Thus most project management tools would find appropriate work item types and activity types which are consistent with their view.

2.5.1 The work item types of the P-path

The data relevant for project management are classified according to 3 criteria (which reflect themselves also in the numbering scheme):

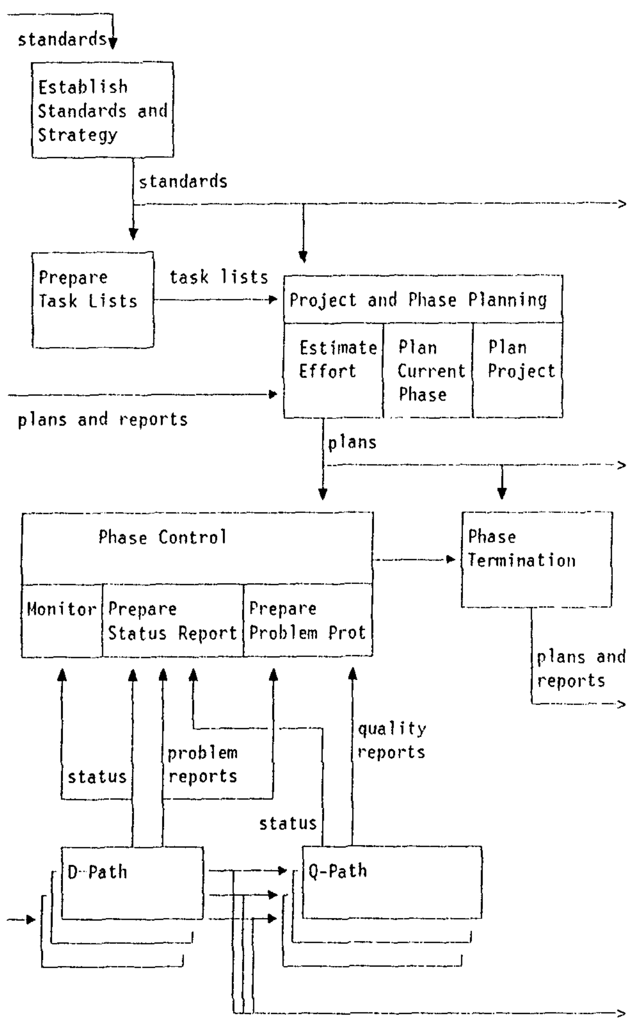


Figure 16. The Project Management Path of ADPS

Division characterizes how far or near from reality the respective data are:

Project Base is a collection of data necessary for project initiation, typically the Project Contract containing technical and project-guiding data.

Estimates specify quantitative statements about resource usage without concern of their distribution over time.

Plans and Schedules correlate Estimates with a time scale.

Status Information record actual values of resource utilization as observed in the project.

Deviations describe the difference between Plans/Schedules and actual resource consumption.

Evaluations and Recommendations are concerned with the impact and relevance of Deviations for the project progress. They contain proposed actions etc.

Decisions and Completion (reports), including Protocols record decisions made and milestones achieved. Certain activities end by creating a protocol.

Group The second criterion is the planning horizon to which the work item type belongs, spanning a range from inter-project background to individual activities:

- **Project Background** ('overall level')
- **Project level**
- **Phase level**
- **Activity level**

Resource Type The third classification dimension is the type of resource to be controlled:

- **Time and Tasks**
- **Personnel**
- **Materials (hardware and software)**
- **Budget**
- **Development Process Information**

2.5.2 The activity types of the P-path

Planning/control activities are essentially done in repetitive cycles. For ADPS the planning interval was chosen to be the **phase**. For each phase the same set of P-path activities is performed (Figure 16):

Establish Standards and Strategy:
Standard and rules, local to the project or the phase are established, e.g. naming conventions, special sequencing of actions. Note that *global* standards and procedures are actually provided by the Process Model.

Prepare Task List:
This is a key to project management. All tasks which should be included in the planning process are identified. Many of these tasks will be activities (or activity types) as described in the process model, but further tasks (e.g. education) have to be added and the proper granularity of planning decided upon.

Project and Phase Planning:
After estimating the effort needed for every task (time, manpower etc.) detailed plans for the current phase and - as far as the project schedule is affected - for the whole project are produced.

Phase Control:
The progress of the project is observed. Actual data about resource usage is collected, problem reports are analyzed and quality reports are scrutinized for potential problems which impact project progress.

Phase Termination:
The results of the phase are evaluated, the transition to the next phase is prepared. A decision about continuation is made.

3.0 The Process Mechanism

The purpose of the Process Mechanism is the instrumentation of the software development process, i.e. providing the services requested in 1.4, "Integrated Project Support Environments". The structure of ADPS reflects the basic components of an IPSE (Figure 17). One of the crucial components of an IPSE is the style of user interface. In ADPS a standard panel form, the so-called **work bench** (Figure 19) is the major interface for application developers. Different interfaces are provided for persons modifying and tuning the development process itself.

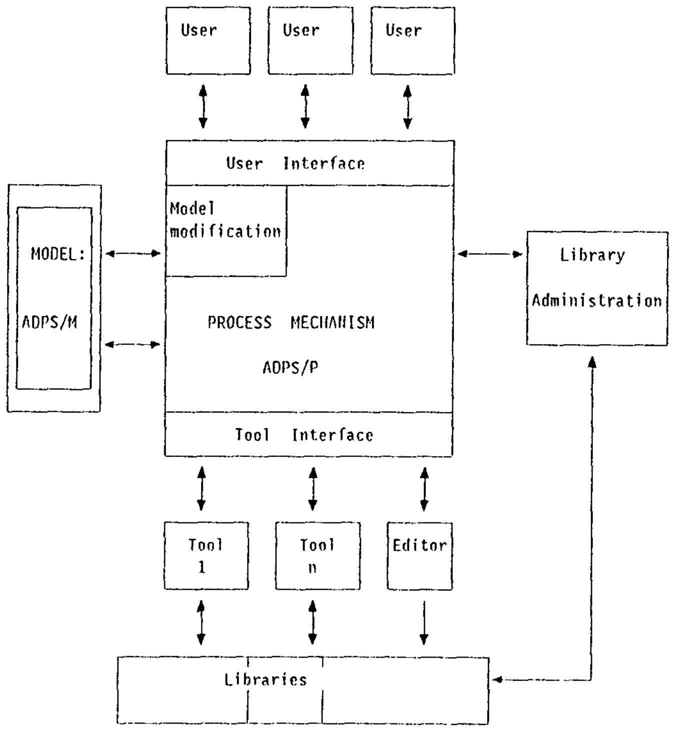


Figure 17. The structure of APDS

3.1 The Workbench, the prime user interface

The **workbench** (Figure 19) presents to the user one activity type for doing some project work, its abstraction is shown in Figure 18. It shows the activity type together with those work item types which are either needed to perform that work or which are to be created. Attached tools are indicated. For each activity type and work item type the user can request help information. For each work item type the user can, in a subsequent panel, see all existing work items. One can interrogate various attributes of the work item (e.g. owner, date of last update, associated tools etc.). From this panel the user will select specific work items to work on them by calling either an editor or a specialized tool.

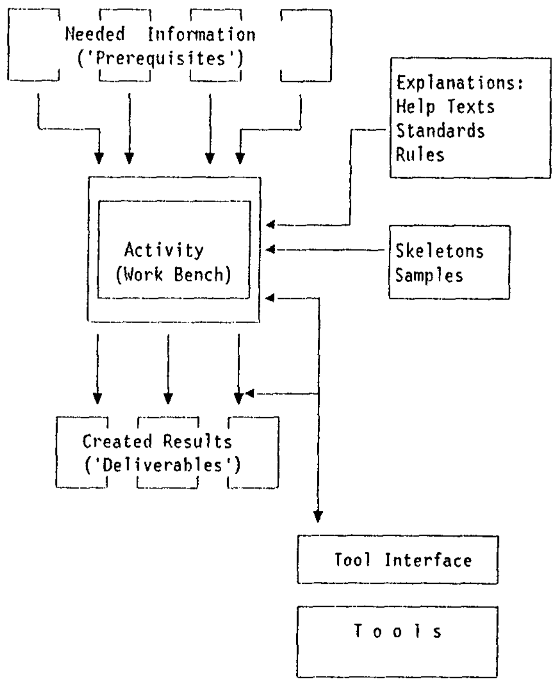
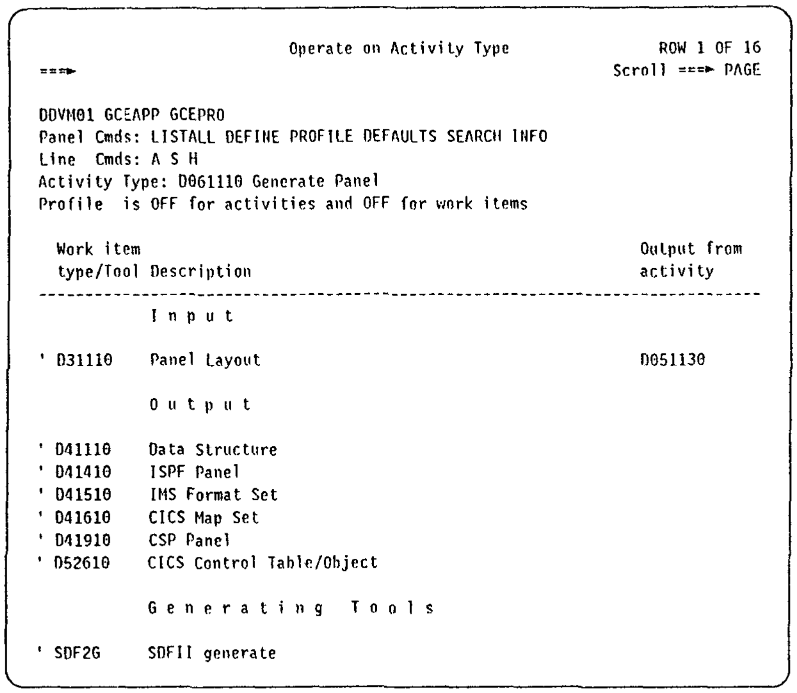

Needed Information ('Prerequisites')
Explanations: Help Texts Standards Rules
Activity (Work Bench)
Skeletons Samples
Created Results ('Deliverables')
Tool Interface
T o o l s

Figure 18. Abstract view of the ADPS workbench



```
                     Operate on Activity Type              ROW 1 OF 16
===►                                                     Scroll ===► PAGE

DDVM01 GCEAPP GCEPRO
Panel Cmds: LISTALL DEFINE PROFILE DEFAULTS SEARCH INFO
Line  Cmds: A S H
Activity Type: D061110 Generate Panel
Profile  is OFF for activities and OFF for work items

 Work item                                                Output from
 type/Tool Description                                    activity
-----------------------------------------------------------------------
           I n p u t

' D31110   Panel Layout                                   D051130

           O u t p u t

' D41110   Data Structure
' D41410   ISPF Panel
' D41510   IMS Format Set
' D41610   CICS Map Set
' D41910   CSP Panel
' D52610   CICS Control Table/Object

           G e n e r a t i n g   T o o l s

' SDF2G    SDFII generate
```

Figure 19. Panel form of ADPS' workbench .

3.2 Tool and Library Attachment

Generally one of the difficulties of software development is the use of heterogeneous tool interfaces and library interfaces. ADPS standardizes the access to all work items by allowing to state at project definition time in which libraries the various work items are to be stored. From then on it is sufficient to identify a work item by its ADPS-name - the link to the appropriate library is automatically established. In a similar vein access to tools is defined once, the actual call of the tool and the association with the appropriate work items can then be done by the system. Each tool attachment consists of two parts (Figure 20) the **Tool Driver** establishing the link to the ADPS environment and the **Tool Caller** providing the individual calling sequences for the specific tool.

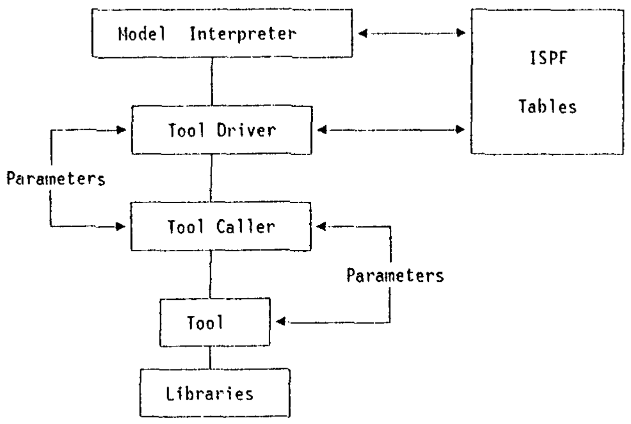


Figure 20. The ADPS tool attachment

3.3 Skeletons and Models

A low level but highly important flavor of re-use is copying standardized pieces of text. ADPS keeps for every work item type a skeleton which is presented to the user whenever a work item is initially created. Such skeletons not only ease the work of the software developer they also provide a important way to standardize work item appearance and contents. Similarly when writing code a programmer can call upon **models** of code constructs which act as templates for the code to be written.

3.4 Help Panels

Explanations, standards and rules are of little help, when they are stacked away in a shelf. Only easy immediate retrieval - via a 'HELP-Key' during work - induces a developer to frequently consult them. This allows to make standards, rules etc. available to the developers, and what is equally important, to keep the information up-to-date centrally. At the same time it increases the adherence to pre-established standards and procedures, common usage etc. Even to well-experiences professionals this provides valuable reminders and help.

3.5 Adapting ADPS

Only in few cases will the Process Model be 100% acceptable as delivered. For local, project specific changes ADPS provides two levels of *hiding* (i.e. suppression) of parts of the Base Model (Figure 21). For example a project which does not need manual operations can *hide* the respective activity types and work item types. Additionally several Base Models can be established (Figure 21) to accommodate major differing development strategies. It is expected that such variants of the Base Models gradually evolve as more and more know-how about with the development process in incorporated in them. Such *long-term* changes are done via the model modification component of ADPS. Thus in an actual implementation we see 3 levels of process models:

The **Base Model** contains the basic development model of the enterprise or division.
The **Application Model** is valid for a relevant area of business.
The **Project Model** finally controls development of a single project.

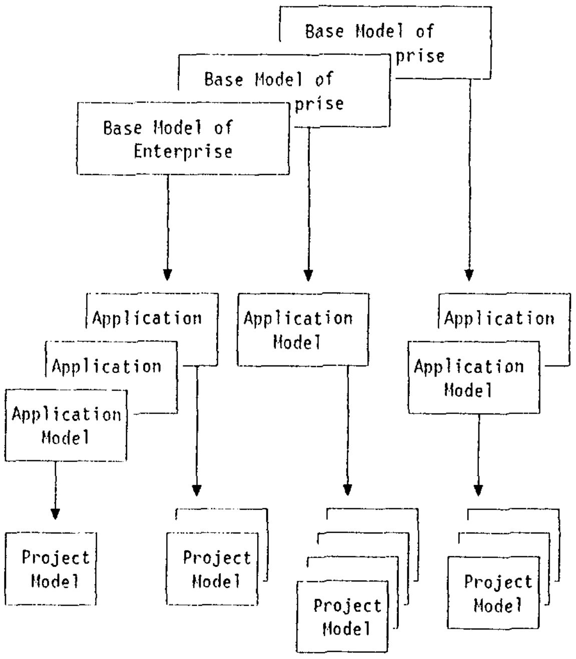


Figure 21. A Hierarchy of Process Models

4.0 Summary

ADPS is available as two separate products, each of which is available for VM/SP and MVS: (**Application Development Project Support/Application Development Model** and **Application Development Project Support/Process Mechanism**). ADPS is implemented as a set of application programs using ISPF as dialog manager, all data owned by ADPS being held in ISPF tables. ADPS is available in English, German and Japanese.

4.1 Developer's Views

For the developer the workbench (Figure 18) provides the necessary working aid in an appropriate, easy-to-understand, and structured way.

- The workbench shows the necessary input work item types of an activity type, the output work item types to be created, and the defined tools. It provides access to already existing work items for a work item type. Using associated tools, these work items can be processed, viewed and printed. New ones can be created.
- By using a suitable profile the developer can select only those activity types that are significant to him.
- Within one activity type, the application of a specific tool to specific work items can be defined explicitly by the developer. This is the basis for automating this activity by appropriate tools.
- Help information can be requested for activity types, work item types, and tools.
- Hidden from the user the system stores, keeps track and retrieves all necessary work items without undue effort by the developer. ADPS remembers and displays the status and many properties of any work items.

When one tries to install an IPSEs some problems may arise:

- Acceptance problems by developers may arise, because they
  - fear a loss of creative freedom,
  - fear excessive control ('Big Brother'),
  - shy away from novel approaches,
  - have to apply a higher level of formality and
  - have to exercise more discipline.
- A higher level of sophistication might be required by some lead programmers,
- Training is needed to understand the new system and method,
- Adaption/change of the existing organization, of existing methods and tools, of existing project planning methods will be come necessary.
- The cost of introduction of an IPSE should not be underestimated, because of the need for training and consulting, the cost of the initial learning curve cost etc.

4.2 Advantages of ADPS

An industrial process consists of many components and has to fulfill a wide range of sometimes contradicting requirements while maintaining appropriate priorities. ADPS emphasizes the framework-function of a process, i.e. it emphasizes the complete description of the Software Process without prescribing a specific method of software development. ADPS provides, among other things, an adequate administrative framework for an appropriate project management support.

The main advantages of ADPS are:

Methodical description of the Software Process: The Process Model covers the whole development process, including project management and quality control. The network character of the activities and results as described by the Process Model facilitates a systematic execution of the application development and contributes to higher reliability in planning, processing, and controlling of software development. The model has been kept free from specific methods. This methodological independence allows to satisfy a wide range of users.

Adaptability: The development process, methodically set up by the Process Model, enables an adaptation to specific needs of a company, and to temporary requirements of a single project. The ADPS model encourages its augmentation by specific methods and tools. Experience with the Software Development Process can be incorporated into the Process Model in order to aid later projects and to build up an *enterprise software development culture.*

Uniform work environment: The Process Model, interpreted by the Process Mechanism, provides a uniform work environment through the uniform representation of activity types. The uniform dialog environment provides a consistent access to activities and results as well as to help information. By connecting the appropriate software tools an ADPS installation can guide its users to the best solution of their tasks. Tools can be standard IBM products, as well as self-developed or acquired programs.

Wide applicability: The open, extendable tool interface permits a user to include tools and methods of his own into the model.

Project control: ADPS' structured development process and the unambiguous status information, facilitates project planning, project controlling, and project surveillance. The respective status of all activities and work items is accessible to all users.

Reduced education expenditure: The support of detailed help information and work item skeletons facilitates training of new employees, and thus contributes to the reduction of education effort, allowing newcomers to become productive earlier.

Higher productivity: ADPS guides the developer by its methodically designed process, which avoids unnecessary work or the omission of planned work. Through integration of the right tools at the right position of the process and through automatic access to work items via the library administration the basic effort for performing individual activities is reduced and the best possible support is provided. This suggests that with the same amount of human work force, considerably more development work can be achieved.

Higher quality: The verifications modelled by the Quality Control Path defines a timely examination of the output work items. By installing work item skeletons, the observance of project standards is facilitated. The developer is guided through all necessary process steps; this reduces the danger that necessary activities are circumvented. Through avoidance of incorrect operations, omitting of or deviating from company standards, the application can be completed with a significantly higher level of quality and transparency.

Financial gain: The introduction of an ADPS-supported development process presents, from the start, an efficient planning basis for each project. The higher productivity to be gained by using ADPS reduces the overall development expenditures. Due to the high quality standard, the expenditure for maintenance is reduced.

In summary, ADPS proposes a total system approach to application development by defining a detailed Software Development Process and making this process interpretatively accessible to the software developer, project manager and other personnel involved. The use of ADPS results in a more systematic, professional and transparent application development.

5.0 Literature

[Abbott_87] Abbott R.J.: Knowledge Abstraction.- Comm. ACM vol 30 (1987) no. 8, pp.664-671

[Akstaller_86] Akstaller U., Biendarra T., Graegl B., Noth T., Mertens P.: Automatisiertes OPSS-Praktikum.- IBM Nachrichten vol. 36 (1986) No. 284, pp. 31-35

[Bender_83] Bender H., Fuhrmann R., Kittel H.U., Menze B., Mueller J.E., Nadolny D.: Software Engineering in der Praxis (das Bertelsmann-Modell).- CW-Publikation, Muenchen 1983

[Boehm_80] Boehm B.W.: Characteristics of Software Quality.- TRW Series of Software Technology, North Holland 1980.

[Boehm_84] Boehm B.W.: Software Life Cycle Factors.- in Vick C.R., Ramamoorthy C.C.(eds.): Handbook of Software Engineering, Van Nostrand 1984 pp. 494-518

[Boehm_84b] Boehm B. et al.: A Software Development Environment for Improving Productivity.- Computer June 1984, pp. 30-44

[Brereton_88] Brereton P. (ed.): Software Engineering Environments.- Ellis Horwood Ltd., J.Wiley 1988

[Brooks_86] Brooks F.P.Jr.: No Silver Bullet - Essence and Accidents of Software Engineering.- Kugler H.J.(ed.): Information Processing 86, IFIP Congress 1986 pp.1069-1076

[Chroust_88c] Chroust G.: Softwareentwicklung fuer maschinennahe Systemarchitekturen.- Softwaretechnik-Trends Heft 8-1 (1988), pp. 9-12

[Chroust_88g] Chroust G., Gschwandtner O., Mutschmann-Sanchez D.: Das Entwicklungssystem ADPS der IBM.- Oesterle H. (ed.): Anleitung zu einer praxisorientierten Software-Entwicklungsumgebung.- Gutzwiller T., Oesterle H. (eds.): Anleitung zu einer praxisorientierten Software-Entwicklungsumgebung, Band 2.- AIT Verlag Schweiz 1988, pp. 123-148

[Crosby_80] Crobsy P.B.: Quality is Free.- Mentor Books / New American Library 1980

[Dolotta_76] Dolotta T.A., Mashey J.R.: An Introduction to the Programmer's Workbench.- 2nd Int. Conference on Software Engineering, 1976, p.182-186

[End_86] End W., Gotthardt H., Winkelmann R.: Softwareentwicklung - Leitfaden fuer Planung, Realisierung und Einfuehrung von DV-Verfahren.- Siemens AG., 5. ueberarb. u. erw. Auflage, 1986

[Hausen_87] Hausen H.L.: Zur Einschaetzung moderner Software Engineering Environments.- ADV (ed.) Quo Vadis EDV - Realitaet und Vision 1987 - ADV Tagung Maerz 1987, pp. 64-81

[Hausen_87b] Hausen H.L.: Effectively Instrumentable Life Cycle Model.- Schumny H., Molgaard J.(ed.): Proc.EUROMICRO 87, Microprocessing and Microprogramming vol. 21 (1987) no. 1-5, pp. 361-370.

[IBM_86] IBM Corp.: COMMAND - Vorgehensmodell fuer professionelle Entwicklung und Wartung von Anwendungen.- IBM Corp.,Form GT 12-3255-1, Feb. 1986

[IBM_86e] IBM Corp.: VIDOC - Werkzeuganschluss Personal Computer (VIDOC Benutzerhandbuch).- IBM Corp., IBM Deutschland, Aug 1986

[IBM_87] IBM Corp.: Application Development Project Support/ Application Development Model and Process Mechanism - General Information.- IBM Corp.,Form No. GH19-8109-0, 1987

[Jackson_82] Jackson M.A.: Software Development as an Engineering Problem.- Angewandte Informatik No. 2 (1982), pp. 96-103

[Kraft_77] Kraft P.: Programmers and Managers.- Heidelberg Science Lib, Springer 1977.

[Lehman_80] Lehman M.M.: Programs, Life Cycles, and Laws of Software Evolution.- Proc. IEEE vol. 68 (1980) No. 9, pp. 1060-1076.

[Lehman_85] Lehman M.M., Belady L.A.: Program Evolution - Processes of Software Change.- APIC Studies in Data Proc. No. 27, Academic Press 1985

[McClure_81] McClure C.L.: Managing the Software Development and Maintenance.- Van Nostrand Reinhold Data Proc. Series, Van Nostrand 1981.

[McDermid_85] McDermid J.(ed.): Integrated project support environments.- Peter Peregrinus Ltd. 1985

[Metzger_81] Metzger P.W.: Managing a Programming Project.- - 2nd Edition. - Prentice-Hall, 1981

[Naur_69] Naur P., Randell B.(eds.): Software Engineering.- Proc. Nato Working Conference Oct. 1968

[Neuhold_85a] Neuhold E.J., Chroust G.: Formal Models in Programming.- North Holland Publ. Comp. 1985

[Peters_78] Peters L.J., Tripp L.L.: A Model of Software Engineering.- Proc. 3rd Int. Conf. on Software Engineering, May 1978, pp. 63-70.

[Sommerville_86] Sommerville I.(ed.): Software Engineering Environments.- P.Peregrinus Ltd., London 1986

[Wileden_86] Wileden J.C., Dowson M.(eds.): Internat. Workshop on the Software Process and Software Environments.- Software Eng. Notes vol. 11 (1986) No. 4, pp. 1-74

[Zemanek_80] Zemanek H.: Entwurf und Verantwortung.- IBM Nachrichten No. 241 (1978) pp.173-182