Eindhoven University of Technology

MASTER

Building a tool for the first PIMS prototype

Wissing, R.F.

Award date:
1988

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Afstudeerverslag

Building a tool for the First PIMS prototype

R.F. Wissing

Keywords: artificial intelligence, expertsystem, Lisp, frames, message/objectmodel, modelling languages, object oriented programming, Project Integrated Management System, project management.

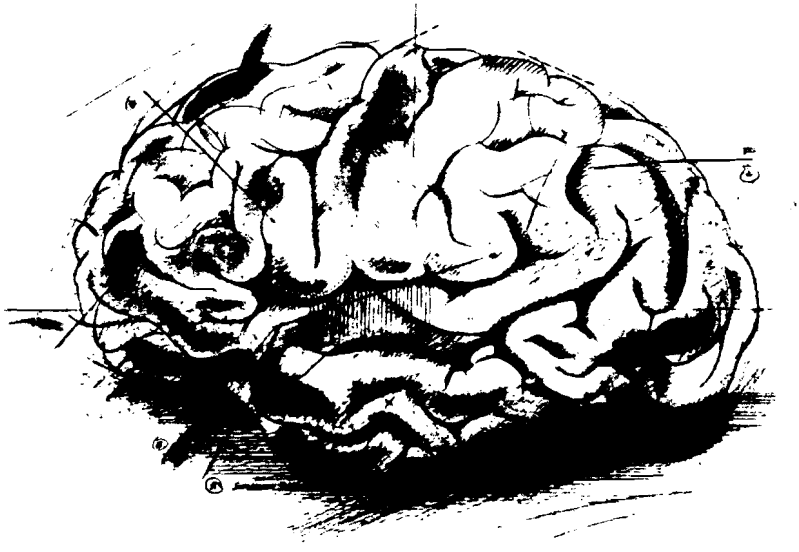
Hoogleraar: Prof. Ir. A. Heetman, TUE

Mentoren: Ir. A.G.M. Geurts, TUE

Ir. H. Frijters, BSO/ATE

Eindhoven, juni 1988

WHAT DOES PIMS HAVE THAT OTHER SYSTEMS DON'T?



PROJECT INTEGRATED MANAGEMENT SYSTEM,
ESPRIT PROJECT 814.

If we want to learn anything, we mustn't try to learn everything.
The Lump Law.

Summary

This report provides a brief overview of the ESPRIT project 814, entitled "Project Integrated Management System" (PIMS). The ultimate goal of this project is to realize a prototype system capable of assisting, in an intelligent way, the manager of a software development team. This means, techniques in artificial intelligence (AI) play an important role in PIMS.
The PIMS system consists of three main parts:

- the set of project management tools;
- the information manager;
- the front end (user interface).

Project management tasks are carried out by tools. In PIMS a tool is a set of procedures which perform the necessary processing related to a project management task. A tool does not perform any transput, and it has no detailed user interaction or communication with another tool.

All globally accessible data within the PIMS-system is maintained by an object base management system, called the information manager. The chosen structure is a semantic network of frame based objects.

The front end handles the user interaction and is responsible for selecting the tool functions to be called. This part has access to knowledge about the way the PIMS system can be used.

This report highlights the PIMS software development environment, including its impact on the work of a tool builder. The PIMS system will be implemented in Common Lisp with an object oriented shell: Flavors.

Each tool is a Flavors object with "methods" representing the tool functions. PIMS possesses modelling languages both for the data model and for the project management tools, they are called the Data Modelling Language (DML) and the Tool Modelling Language (TML). The TML description is illustrated with parts of the tool, which supports the project manager in tracking the works progress and forecasting the amount of work to do. It is clear, that the PIMS object oriented software development environment makes the design and implementation of the tool easier, because it transfers work from the tool builder to the system. That is why, it improves the productivity of the tool builder. The disadvantage of such a system is, that its performance is low.

Contents

Problems worthy of attack prove their worth by bitting back.

Piet Hein.

1. Introduction.

This report gives an impression of the activities, which the author has performed within the framework of his graduation. These activities took place from April 1987 till February 1988 at BSO/AT in Eindhoven. The supervisors were Ir. A.G.M. Geurts of the Digital Systems Group of the Eindhoven University of Technology and Ir. H. Frijters of BSO/AT.

The aim of the above mentioned activities was to specify, design and implement a tool, which supports the manager of a software project in tracking the work progress and forecasting the amount of work to do according to the data available.
This tool is part of the ESPRIT project 814, entitled Project Integrated Management System or PIMS. The ultimate goal of this project is to realize a prototype system capable of assisting the manager of a software development team in an intelligent way.

After this introduction Chapter 2 gives some general information about PIMS. In Chapter 3 a description is given of the project management domain. Since Artificial Intelligence plays an important role in PIMS, Chapter 4 gives a short introduction in this subject. Further the features of Lisp will be described in this chapter. Chapter 5 tells something about object oriented programming. Chapter 6 describes the architecture for the first PIMS prototype.

In PIMS, modelling languages both for the data model and for the project management tools are defined, these languages are called the Data Modelling Language (DML) and Tool Modelling Language (TML) respectively. These languages are described in Chapter 7 and 8. Finally, Chapter 9 describes how the PIMS system controls the execution of tools and how it makes use of the information contained in the Tool Modelling Language description of the tools.

2. PIMS

The European Community Commission launched the European Strategic Programme for Research and development in Information Technology (ESPRIT) in 1984. The main objectives of ESPRIT are:

- to make the European Information Technology competitive in the 1990's;
- to stimulate the European cooperative pre-competitive research and development in information technology.

ESPRIT project no 814, entitled "Project Integrated Management System" (PIMS) is a project in the field of project management.

It has almost become an accepted fact of life that software development projects over-run their budgets. Despite much effort being invested to support the design and construction of software systems, little attention has, as yet, been given to helping the project manager. The PIMS system will provide this support by:

- offering an integrated, compatible set of tools such as estimators, resource allocators and schedulers, progress tracking tools and report generators;
- linking this tool set together with an intelligent interface which will suggest possible courses of action;
- having an understanding of the manager's job. This understanding will be achieved by studying literature and applying knowledge engineering techniques to various aspects of the field of project management.

The four companies and the four universities on this 3.5 years project (effort: 56 man-years), which started in January 1986, are the following:

Prime contractor:	CAP Sogeti Innovation, France.
Partners:	BSO/Automation technology, The Netherlands.
	PA Computers and Telecommunication, Great Britain.
	The Turing Institute, Great Britain.
Sub Contractors:	University of Amsterdam, Department of Social Science Informatics; University College London, Computer Science Department; London School of Economics, Social Psychology Department; London Business School.

During the time the author was working at the project, the task of BSO/Automation Technology, with respect to the tools, was mainly to develop the scheduler, resource allocator, tracker and predictor tools.

The trouble with people is not that they don't know, but they know so much that ain't so.

Henry Wheeler Shaw.

3. Domain of PIMS.

The domain of PIMS is the management of software projects. Because of the vastness of the subject PIMS is restricted in some respects:

- Only technical activity management will be considered, for instance motivation team members and making project contracts will not be a topic.
- The main research is the daily management of the project.
- One person, the project manager, is in charge of the daily management.
- The project team has no sub-hierarchy.

In this limited domain the task of the project manager can be decomposed in the following (iterative) sub-tasks:

- initiation;
- workbreakdown;
- scheduling;
- resource allocating;
- tracking and forecasting;
- diagnosis.

Figure 1 shows these tasks in the waterfall model of the project management life cycle. Besides tools to support the tasks of figure 1, in PIMS #1 the following three tools have been implemented too: Resource Definition, Project Calendar and Risk Analysis Tool.

3.1. Initiation

The general function of initiation is to initiate a new project or to complete an already existing view on the project. This is achieved by setting the project name, the client name, the external milestone(s) and their dates, one or more (external) deliverables, and so on. Before starting a new project (accepting the contract) the project risk analysis tool in PIMS can be used to support the project manager in determining the risks of a certain project.This tool helps the project manager in characterizing the project's relations with its environment. Here, the project is viewed 'from the outside': its internal structure is not yet known, its functionality and costs (in the widest sense) are like those of a product which the organization is considering producing. The environment involves linkages with both the host organization of the project and with the client for its deliverables.

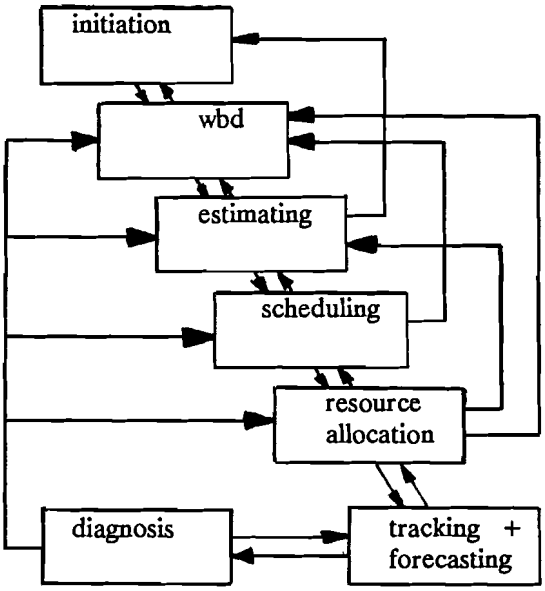


Figure 1: Waterfall model of the project management life cycle.

3.2. Workbreakdown

Because it is impossible to oversee the project as a whole, the project has to be decomposed functionally into small, uncomplicated components, each of which carries out a specific activity which has to perform a task. This means an initial workbreakdownstructure (wbs) has to be made. This structure is composed of two hierarchies: namely the product hierarchy (system, modules, submodules, subsubmodules and so on), and the activity hierarchy (analyzing, designing, coding, testing, integrating and so on). These hierarchies can be connected in which ever way best fits the project. There are three criteria which should be used as objectives for achieving a good decomposition:

-1. Maximize the cohesion within a low level (bottom) activity.
-2. Minimize the coupling between the low level activities.
-3. Aim at a balanced overall control structure.

Criteria in the first two areas, cohesion and coupling, have been proposed by Constantine and Yourdon (Yourdon et al., 79). Cohesion is a description or measure of the internal strength and consistency of an activity; this consistency should be as high as possible in order for an activity to be considered good. (Consistency here means, that the elements of an activity should be functionally related, as far as possible) Coupling, on the other hand, is a measure of dependence of one activity upon the others; it is kept as low as possible in a good decomposition. A further criterion is that of 'balance' in the hierarchical structure of the activities (figure 2): A balanced structure is preferable to an unbalanced one.

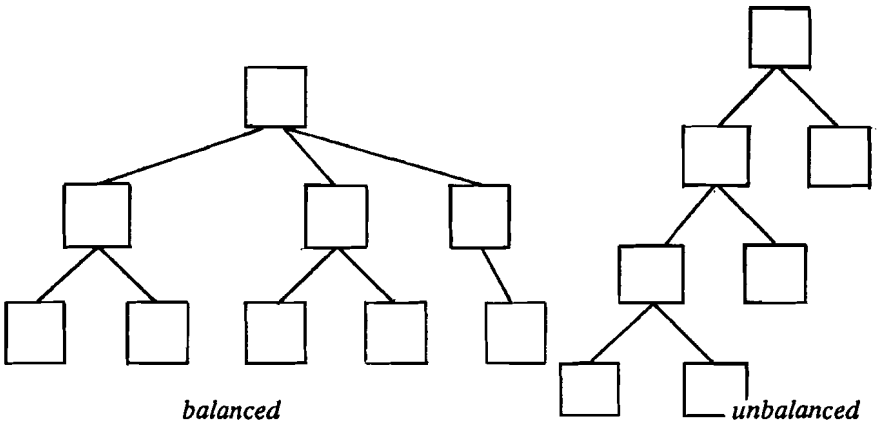


figure 2: A graphical representation of a balanced and unbalanced structure.

Coupling, cohesion and balancing are not the only metrics available for measuring the quality of a decomposition, although they are the most well documented and the most quantifiable, relatively speaking. Other, less exact metrics (better: characteristics) of functionally decomposed system structures are complexity and correctness. Complexity is a subjective characteristic at best, but less complexity is generally considered to be better than more complexity. A complex activity can usually be rendered less complex by dividing it into subactivities. Correctness of a decomposition would seem to be quantifiable, but in practice it is almost impossible to prove.

3.3. Estimating

The aim of each activity (workbreakdownelement) is to perform one or more tasks. The lowest level elements of the workbreakdown have a one to one relation to a task. (Note, for a good decomposition all the lowest level tasks should cover the entire project.) To each of these tasks an effort and a time (and thus indirectly the staffing level) have to be estimated with the aid of an estimation model.

In [Wissing, 87a] some known methods and practices in software estimating are listed, along with advice on practices to be avoided. Further, research in parametric estimation models is discussed. A view is given on the relationship between effort and time in software engineering.

3.4. Scheduling

Information about logical relations and time estimates of each task is used to construct a network diagram, which is a graphical model for analysis of the scheduling problem. In PIMS activity-on-node diagrams are used. In these diagrams each node represents an activity (or better: task) and each arc shows logical precedence.

This approach is used with the so-called Critical Path Method (CPM). This is the most favoured method in practice today. The longest path in such a network is called the critical path. It determines the project length in the sense that precedence relations permit no earlier completion of the project.

The activities along a critical path are called critical activities, because a delay in any one of them will cause a delay in the completion of the entire project. By contrast it may be possible to absorb some delay in noncritical activities without affecting the project completion date. The scheduler in PIMS takes the project calendar into account. This calendar should be defined with the aid of the so-called Calendar Definition Tool. With this tool it is possible to define the days which are special, like company holidays and public holidays.

3.5. Resource Allocating

In practice many tasks in a project require resources of various types. It is quite usual to have a resource limit that is not fixed at the same level throughout the project duration. The availability of manpower (and machinery) can be at different levels of availability during the working days.

Resource profiles can be defined with the aid of the Resource Definition Tool. With this tool it is possible to define the name of a resource, the characteristics of this resource (a list of properties), the daily costs to use the resource and the time intervals for not availability of the resource. Obviously, these resource profiles could be taken into account during allocating the resources to tasks. In PIMS #1 the resource allocator is a manual one.

So, it does not use a resource constrained project scheduling algorithm. These algorithms are based on heuristics. The idea behind heuristic algorithms is to rank the tasks by some rule and to schedule the tasks in that ranking order ensuring that the resource limits on the project are never exceeded. Thus tasks considered to be important in some sense are scheduled as soon as possible.

When resources are constrained, the scheduling algorithm cannot readily produce 'float' values for each of the tasks in the schedule, as were available in the unconstrained case. When task delays occur, the project manager must often reschedule the resource constrained project to determine new task start and finish dates. These may vary quite dramatically from those in the original schedule, thus causing the reorganization of future plans in the project, often at considerable cost. There is clearly a need therefore for a stable schedule, i.e., one that does not drastically alter task start and finish times, or at least alters them uniformly, when the project is rescheduled under the same resource constraints with a few changed tasks durations.

3.6. Tracking

The tracker provides the means to feed the user's actual knowledge of the running project into the PIMS system and it indicates to the user the status of the project. This has been described extensively in [Wissing, 87a]. The specification of the tracker tool in PIMS #1 can be found in [Wissing, 87b].

3.7. Diagnosis

The diagnosis tool tries to determine why a certain discrepancy between plan and reality occurred. There are several possible causes for this, like:

- the value of one or more project metric, that was attached during estimating the values of them, appears to be incorrect;
- the model, which is described by the metrics, is incorrect (for example the metrics are weighed wrong);
- The chosen metric set is incorrect.

In most cases the diagnosis will lead to reconsidering the plan. In PIMS #1 a diagnosis tool is not available.

3.8. Predicting

If there is a (serious) discrepancy between the plan and reality, generally the projectmanager wants to know what the influence of this delay will be on the scheduled project completion date. In this case a new prediction of the effort and timescale has to be made.
For that we see the next three possibilities:

- starting a re-planning process
- extrapolating the measured values on the basis of the project curve
- exponential smoothing (short-term forecasting).

The first possibility needs a diagnosis tool to determine the cause(s) of a certain discrepancy between plan and reality. If the cause(s) is (are) found and there is no remedy that does not affect the plan, a replanning process has to start by which one or more of the following things can be needed: completing or re-structuring the workbreakdown, re-estimating the effort and duration of task(s), re-scheduling the tasks, re-allocating the resources. The foregoing shows, if there is (or seems to appear) a serious deviation from plan, the planning tools can be used to make a new and hopefully more accurate, (long-term) prediction of the effort and timescale of the project. This possibility will be discarded in this report, because in PIMS #1 a diagnosis tool is not available and replanning is beyond the scope of the planning tools.

The second possibility to make a long term prediction extrapolates the measured values on the basis of the equation (curve), that is assumed to represent the project model (for project modelling techniques, see for instance [Wissing, 87a]). So, in this case we suppose that a project follows the curve that is described by this equation. Since in PIMS #1 the project curve is unknown, the predictor will also not cover this long-term prediction.

The third possibility makes only short-term prediction possible. To predict for this immediate future, forecasting techniques will be used to estimate the effort/cost to complete a task. The reason, that those techniques are only useful for short term prediction is, that they are based on the assumption that existing pattern will continue into the future and this assumption is more likely to be correct over the short term than it is over the long term.

The third possibility will be the one that will be supported by the predictor in PIMS #1. It will be based on the revised effort (the revised effort is equal to the effort spent plus the remaining effort). From this and the average utilization of the resource in the past, a prediction is made of the expected finished date. The prediction mechanism will never by itself modify any scheduling or allocation data.

In the fact a forecast can be considered as an extra variable belonging to the reports generated in the tracker [Wissing, 87b]. So, in order to minimize the number of tools in PIMS #1, both tools will be integrated. The specification of short-term predicting in PIMS #1 can be found in [Wissing, 88].

Today's expert systems are more akin to idiot savants than to real human experts.

R.J. Brachman at al.

4. Artificial Intelligence and Lisp

4.1. Artificial Intelligence

Since the stated aims of PIMS, it will be obvious techniques in artificial intelligence should play an important role in PIMS.

As early as 1947, British mathematician Alan Turing grappled with the meaning of machine intelligence [Turing, 50]. Turing concluded that if a user could sit at a computer terminal, type questions and not be able to tell whether a machine or hidden human operator was answering those questions, the machine could be said to have intelligence. This level of intelligence has not been achieved yet, but research into artificial intelligence has produced many examples of good computer science. The term artificial intelligence was originated in a 1956 conference at Dartmouth College. John McCarthy, who coined the term, attended a conference with Marvin Minsky, Nathaniel Rochester and Claude Shannon to discuss ways to simulate human intelligence using machines.

Early artificial intelligence (A.I.) research was based on theories of how the brain functions. Computer scientists tried to give machines intelligence by making them imitate the manner in which the brain processes information. But the working of the brain are still not fully understood. In addition, may aspects of human intelligence that are understood have yet tot be implemented on a machine. Consequently, artificial intelligence accomplishments were small. Over the last several years, artificial intelligence has delivered its first practical results, especially in the field of expert systems (an exploratory, rule-of-thumb approach to problem solving). Not only researchers are leaving the academic environment to start A.I.-firms, but also large corporations (in the Netherlands for instance Shell, Philips and Akzo) are conducting in-house A.I. research.

The knowledge in an expert system is stored in a so-called knowledge base. A knowledge base contains the domain knowledge; this is both factual and heuristic knowledge. Consequently, the knowledge base consists of two parts: a declarative knowledge part and a rule base with a set of production rules. In PIMS the declarative knowledge will be represented by a semantic net of frame based objects (see next chapter). A production rule is able to express specialized knowledge in the form of a relation between a premiss (condition, antecedent) and a conclusion, augmented with a degree of certainty.

In practice, production rules have demonstrated to be an adequate mean to represent heuristic knowledge. A rule states that certain conclusions (its consequents) will occur if specified premisses (its antecedents) are met. The rule's antecedent contains facts (eventually derived from previous conclusions) on which the rule's consequent can be concluded. If an antecedent is true, the consequent is accepted; that is, the rule fires.

An important part of a rule-based system is the inference engine. This mechanism uses the knowledge base (facts and rules) to infer conclusions. The inference engine acts on many rules. If one rule fires, its consequent might cause additions to the factual part in the knowledge base, that could to satisfy the other rules. When used with an inference engine rules are potentially more powerful than sequences of IF-THEN statements in conventional programs. To draw conclusions from conditional statements in a conventional program, a single consequent must describe each possible condition contributing to that conclusion.

Each antecedent-consequent pair must be known in advance and hard-coded into the program. In contrast, each rule in a rule-based system does not necessarily have to lead to a final conclusion. A rule can instead represent one in a series that must be fired to reach a conclusion. The inference engine contains the reasoning methods used in the system. The two methods often employed with production rules are forward chaining (data driven) and backward chaining (goal driven).

Forward chaining starts with facts about a problem makes all inferences possible and draws conclusions. Backward chaining, on the other hand, starts at the goal and works backward, as in diagnostic applications. When a combination of backward and forward chaining is used, the reasoning process will be quite efficient; this is the way how human reasoning commonly works.

In PIMS the knowledge acquisition is mainly done by the University of Amsterdam by interviewing the domain experts (experienced project managers). Knowledge acquisition is difficult, because domain experts often do not know how they solve problems or are unable or unwilling to verbalize their insight. As described in [Wissing, 87a] the management science literature gives us a lot of management rules too. Here, we will illustrate how an expert can classify a discrepancy between the initially estimated effort of the work performed and the actual effort for that work [Wissing, 87a]. In figure 3 we see a simplified decision tree of that problem. (The (abstract) factor quality, for instance, has been omitted although without this factor it is almost impossible to access the progress accurately: slow progress during the design stage may be because the program is larger as we thought or it may be that the designers are producing a particular good design that will be easy to code and test.) The decision tree of figure 4 is a rewrite of figure 3 and is easy to translate to rules.

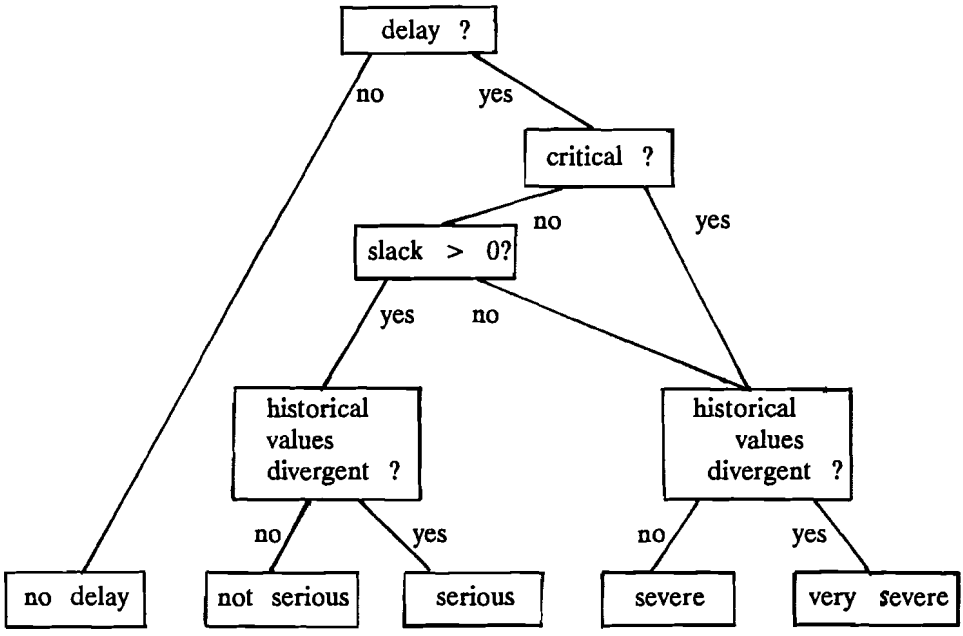


Figure 3: Simplified decision tree of classifying a discrepancy

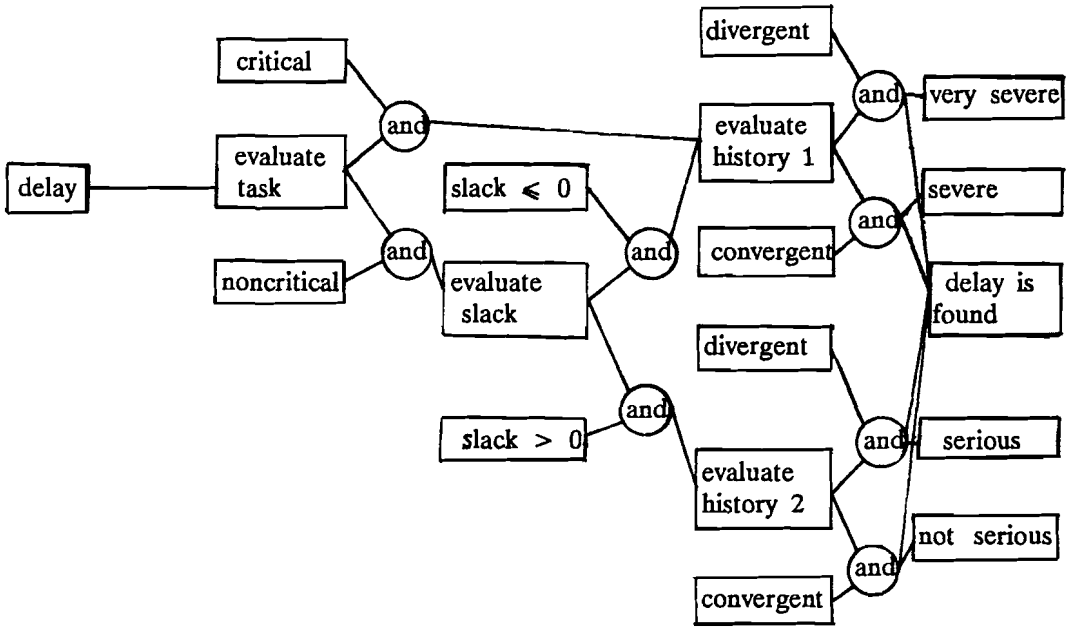


Figure 4: Another presentation form of figure 3

Logic Theorist, one of the first expert systems, performed mathematical calculations. It was developed by Alan Newell, Herbert Simon and J.C. Shaw, and announced at the 1956 Dartmouth College Conference on machine intelligence. Expert systems later emerged in medicine and chemistry. Dendral, developed at Stanford in the 1960's, deduces the chemical structure of molecules. It was one of the first expert systems to achieve widespread use. Mycin, also developed at Stanford, was used for diagnosing infectious diseases. The first commercially successful expert system did not appear until 1980. Developed by John McDermott at Carnegie-Mellon University [McDermott, 82],

R1 is a configuration program for all VAX systems and most of the PDP-11 systems. This expert system determines whether the components can be assembled into a usable system. Its number of rules has grown from 500 in 1980 to 4700 in 1988 and it contains a database of information covering about 9500 computer components. When Digital Equipment Corp. took over the program for daily use, it was renamed XCon (for eXpert Configurer). DEC's use of XCon has reportedly reduced the amount of time, and expense needed to configure new systems. What has emerged from these and later expert system projects is a basic modus operandi for expert system construction and a working definition of expert systems.

Compared with the processing algorithms found in conventional computing, expert systems are characterized by the combination of heuristic and factual knowledge. Unlike conventional programs which mingle data and instructions, expert systems separately maintain factual knowledge that is manipulated and the rules governing reasoning. Although conventional computing concerns itself mainly with numbers or at least with the ability of the computer to represent data using numbers, an expert system must concern itself with processing symbols.

4.2. Lisp

All the groups of the PIMS consortium have agreed to use Lisp as programming language. Lisp was the first popular language for performing experiments in artificial intelligence. E.W. Dijkstra has said once about Lisp:

> *Lisp has jokingly been described as "the most intelligent way to misuse a computer". I think that description is a great compliment, because it transmits the full flavour of liberation: it has assisted a number of our most gifted fellow humans in thinking previously impossible thoughts.*

One of the earliest third generation programming languages, Lisp was devised initially as a notation for solving logic problems. Lisp, shorthand for List Processing, features symbol structures as its primary variable. These symbol structures have atoms, either numbers or symbols, as their most basic element. Lisp's functions and data structures are composed of lists, which can contain atoms and other lists. These lists are represented by lists of pointers. In structure, these lists are binary trees of pointer pairs. In each

pointer pair, one pointer points to the data word for that element, the other to the next element (or node) in the list. Any node, that does not have another node following, it contains a pointer to the terminator NIL, which is the empty list.

In processing lists Lisp does not distinguish between functions (in general: code) and data. Stated in more conventional terms, Lisp is one of the few languages that permits passing functions as parameters. Functions are composed of lists just as complex symbols are. Since the only syntactic rule of Lisp is that parentheses must balance, all expressions containing balanced parentheses will be evaluated, no matter how nonsensical the operations performed may appear. That seems less nonsensical when one realizes that NIL is a valid response. Attempts at symbolic manipulations, that most language interpreters or compilers reject, are taken in stride by the Lisp interpreter. The object operated on could as easily be a program (note, a program is also a list), even the program currently running, as it could be numerical data or a symbol.

Since, as mentioned before, Lisp does not distinguish between data and code, it is possible that a program constructs another program dynamically. This ability is important in AI applications, in which a program generates and tests solutions and does not simply execute algorithms. Lisp's peculiarities may make it the AI software designer's life easy, but in some ways it occasions the hardware designer much brain-racking. Because list processing is the heart of Lisp, the normal limits placed by most language implementations on the size of data structures are totally unacceptable in Lisp. Lisp may construct enormous linked lists to achieve simple operations and the system's memory resources must make that possible or processing will grind to a halt. Thus, Lisp requires an enormous amount of memory.

At BSO we use a SUN/3 (based on a 24 Mhz 68020 microprocessor) containing 16 Mbytes of RAM, nevertheless the performance of the PIMS system was low. For this reason, Lisp is merely used for (rapid) prototyping in a research environment, while the ultimate commercial version will be (re)written in an conventional language, like Pascal and C.

Lisp also uses what is known as heap memory. Memory words in Lisp are not seen as portions of large, contiguous memory blocks, but as entities in themselves. They can be linked together to form lists and each list becomes a separate portion of memory in itself, totally unrelated to any other list. Each element of the heap memory consists of a data word and a link pointer. In Lisp, therefore, memory assignment is accomplished by removing a memory element from the heap or free list and linking it to an existing list or to other free memory elements from the heap to form a new list. When a portion of a list is no longer needed, it is discarded by returning it to the heap or list of free memory elements. This arrangement allows lists to grow, programmers don't need to allocate memory for the lists in advance. Considering how memory intensive it is for Lisp programs to assemble and process lists, advance allocation would be impossible.

Deallocation of memory, or garbage collection, also presents its own set of problems. Lisp deals with garbage collection by ignoring it until there is no more free memory. Then it takes time out to check the entire memory for the free elements that can be returned to the heap. Garbage collection (gc, get coffee) is a time consuming process, because the only way to detect free memory elements is to process all the atoms in memory to find what lists are pointed to by those atoms. Because all lists have names, which are atoms, all valid lists are pointed to by one or more atoms that name them. On the first pass, garbage collection marks each element it can reach by scanning every function and value pointed to by each atom. During the second pass, the entire memory is scanned, those elements that were not marked on the first pass are returned to the free list or heap. Obviously, avoiding garbage collection is one way of speeding execution in Lisp coded applications.

The major problem that arises in Lisp, aside from the hardware demands, stems from all the Lisp dialects. The reason for these dialects is the simplicity of extending Lisp; once defined, functions become part of the environment and can be used by the programmer to define other functions. Such extensibility provides an almost irresistible temptation to create a wholly personalized environment.

Among Lisp dialects, MacLisp, developed at MIT in 1966 is known for its efficiency and its programming facilities. InterLisp, developed a year later by Bolt, Beranek and Newman (Cambridge), emphasizes user interfaces. Xerox's Palo Alto Research Center currently supports InterLisp. Other popular dialects include Scheme from MIT, TLisp from Yale, ZetaLisp (originally derived from MacLisp) from the University of Utah and FranzLisp, originally developed by Digital Equipment Corp. (distributed by Franz Inc.).

Because there are so many dialects, portable Lisp software is virtually nonexistent. Fortunately, the Defense Advance Research Projects Agency (DARPA), which has a vested interest in AI research, stepped in to oppose this trend toward divergence in 1981. DARPA brought together a consortium of companies and universities working in Lisp, that included Symbolics, LMI, Dec, Bell Laboratories, Xerox, Hewlett-Packard, Lawrence Livermore Laboratories, Carnegie-Mellon University, Stanford University, Yale, MIT and UC Berkeley. This consortium defined Common Lisp, an attempt at a universal dialect, that is a subset of ZetaLisp, but most will continue to provide their own customized, extended version of the language. In PIMS we use an extended version Franz Lisp or Extended Common Lisp [Steele, 84]. Part of this version is a powerful feature, called Flavors, which supports object-oriented programming (see next section).

Some programs are elegant, some are exquisite, some are sparkling. My claim is that it is possible to write grand programs, noble programs, truly magnificent ones.

Donald Knuth

5. Object Oriented Programming

5.1. Introduction

Object oriented programming can improve software development. The object oriented approach increases programmer productivity by enhancing software maintainability, extensibility, and reusability. Object-oriented programming replaces conventional operator/operand concepts with new ones, namely messages/objects. Objects are private data, attributes, and operations, methods, supported on these attributes and messages are a request for an object to perform one of its methods. A message specifies what to do, but not how that method should be performed. Object-oriented programming can be considered either revolutionary or evolutionary, depending on the degree to which access to conventional programming techniques is retained.

Purely object-oriented languages such as Smalltalk [Goldberg, 83] represent the revolutionary approach and provide the advantage of conceptual simplicity. The programmer works in a computational environment that contains only objects, so the break with the past is clean and crisp. The evolutionary approach, by contrast, consists of adding object-oriented concepts on the top of conventional languages. A number of these hybrid languages exists today including Flavors (Lisp), Clascal (Pascal), and Objective C (C). These languages do not offer the conceptual consistency of Smalltalk, but they offer the possibility to switch between object-oriented and more traditional programming styles.

5.2. The operator/operand model

Programmers are taught very early that computers do operations on operands and this computational model is preserved through everything they learn subsequently. Generally, programmers think of the computer as two disjoint compartments, one containing operators and the other operands, and they express their intention by selecting what operators are to be applied and to what operands in what order. This conventional model of computation is called the operator/operand model. This model applies at every level. At the machine level we deal with instructions and data; at the language level, expressions and variables (and also statements and data); at the library level, subroutines and parameters; and at the command level, programs and data files. The implementation changes at each level, but the concepts remain the same:

- operators are active and make some predetermined change to whatever operand is supplied them;
- operands are passive and change only when affected by some operator;
- the environment determines what operators are to be applied to what operands in what order; the term environment stands for whatever determines what happens next.

The operator/operand model treats operators and operands as if they were independent. But in practice, operators place strong restrictions on the types of operands they handle correctly; for example, operators like text editors depend on their operand being text files and numerical functions (like square root) assume their operands being numbers.

The operator/operand model provides no way to record this dependence, so the environment must take on the task of determining the type requirements of the operators and the data type of the operands. Imagine what would happen if electricians used this model to design electrical systems, if they treated electrical operators (power sources) and operands (power sinks) independently, the user would be responsible for remembering the pair that carry high-voltage power and those carrying low-voltage signals (in other words buildings would be wired with standard interchangeable plugs and sockets).

Yet in computing, programmers have to handle this type of knowledge manually. Under the message/object approach, by contrast, objects record their type (class) explicitly. This record is used at runtime to determine the set of operations (methods) that can be performed on objects of this class. The traditional operator/operand concepts remain in tact, but they are submerged with a thin layer of new structure (in PIMS: Flavors) that records what operations are valid for what operands. (This can be compared with the incompatible plugs and sockets electricians use.)

5.3. The message/object model

The message/object model decouples the environment from the data types it contains by moving responsibility for operator/operand interdependencies out of implicit storage in the environment and into explicit storage in data structures called classes. The unit of modularity in a message/object system is the class (meta-object), so programmers speak of developing classes rather than programs or modules. Each class has a name, describes the type of objects its instances represent and defines the behaviour of its instances. A class includes a method (procedure) for each type of operation its instances can perform.

The environment manipulates objects by selecting an object and telling it what to do. The object decides how to carry out its assignment by selecting the message implementation in the table of messages its class supports. This is generally called sending the object (instance of a class) a message. So, the environment in the

message/object model is responsible only for deciding what should be done; the object decides how to do it.

Type dependencies cannot spread through the system, since they become permanently encapsulated within classes. A fundamental concept of object oriented programming is a technique that allows new classes to build on top of older less specified classes instead of being rewritten from scratch. In this way, classes can be organized in a hierarchy, where each new level introduces some specialization of the level immediately above. When a class is created, its position in the hierarchy is established by definition of its so-called superclasses, it inherits all the attributes and methods from its superclasses, and it may be augmented by adding increasingly specialized attributes and methods. In the PIMS Project Conceptual Schema, for instance, the three classes Human Resource, Durable Resource, and Consumable Resource have as superclass the class Resource (figure5).

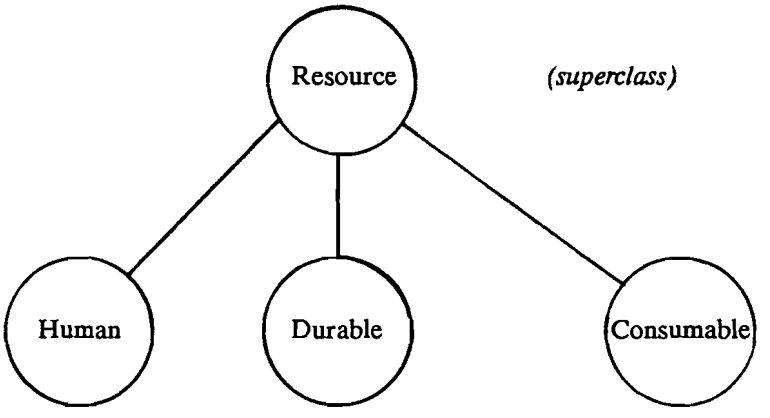


Figure 5: Simple object hierarchy: all the attributes and methods of Resource will be inherited.

In PIMS, all superclasses are partial types, that means, they are designed to have subclasses. When a subclass has multiple superclasses, the superclasses combine to form a type that includes all their individual behaviours. One has to proceed with caution by using superclasses. For instance, one could create type Allocation (the allocation of one particular resource to a particular task) out of the superclasses Task and Resource. But an allocation is composed of these parts, it is not the sum of the behaviour of these parts. In such a case object-oriented languages offer another aggregation mechanism: object attributes. The class Allocation should have attributes that contain the task and resource of the allocation. Achieving aggregation through attributes, instead of through inheritance, also allows more than one instance of a particular class, such attributes are called multivalued attributes; a superclass cannot be inherited more than once. Unlike, for example, relational databases, in an object base normalizing the data is not needed; m:n relations are allowed.

5.4. Object oriented design.

In the software life cycle, object oriented programming concentrates on the design and implementation stages of software development [Booch, 86]. A more comprehensive approach would combine object-oriented techniques with methods such as NIAM (Nijssen Information Analysis Method, see [Verheyen, 82] and [Nijssen, 82]). Because object oriented programming encompasses both design and implementation, it tends to blur the distinction between the two. As Meyer [Meyer, 87] points out, this is an advantage since design and implementation are essentially the same activity: construction software to satisfy a certain specification. The only difference is the level of abstraction: during the design certain details are left unspecified, but in an implementation everything is expressed fully.

In an object oriented design specifying the relevant objects in the application domain is the first task in designing the system. In PIMS some conceptual objecttypes are ProjectView, Task, Deliverable, Resources and Allocation. After specifying the major types of objects, the next task of the design is to develop an interface for each type by describing the methods that make up the type's behaviour. Last, an implementation for each type has to be developed by describing how the type's behaviour is achieved.

An important benefit of the object-oriented style is, that it lends itself to a particular simple and lucid kind of modularity. Modular programming constructs and techniques help and encourage the programmer to write programs that are easy to read and understand, and so are more reliable and maintainable. Object-oriented programming lets a programmer implement a useful facility that presents the caller with a set of external interfaces (methods), without requiring the caller to understand how the internal details of the implementation work. In other words, a program that calls this facility can treat the facility as a black box; the calling program has an implicit contract with the facility guaranteeing external interfaces, and that is all it knows.

A lot of object-oriented programmers think that object-oriented programming is grouping data and methods on that data into a class and they believe that modelling a process as an object goes against the grain of the object-oriented paradigm. By 'process', a procedural mechanism is meant rather than a concrete entity. For example, tracking the work progress is usually thought of as a process, but tasks and resources are considered concrete entities. Should tracking be modeled as an operation on tasks and resources? Or should it be modelled as an independent process defined as a separate type? In PIMS we have decided, in view of their complexity, that management tasks, like tracking, should be modeled as separate object-types (functional objects); this is, in our view, not contrary to the object-oriented style.

Objects should be used to model entities in the application domain. This does not mean, that the only types that should exist are ones that model the external view of the application. A program

can be viewed as a model of a piece of the real world. In moving from real world to a computer system, entities are introduced that do not necessarily correspond to visible 'real-world' entities. For example, in the PIMS system there is a functional object called the Front End (see next chapter). This object handles the user interaction. The concept of the Front End is artificial and is introduced only to help code the application.

People don't do serious work on UNIX systems, they send jokes around the world on UUCP-net and write adventure games and research papers.
Anonymous.

6. The architecture of the PIMS #1 prototype

All the groups of the PIMS consortium have agreed to standardize upon a SUN/3 workstation running under UNIX. This helps ensure that the results of PIMS could be widely available, particularly through eventual compatibility with the ESPRIT directed initiative into a Portable Common Tool Environment (PCTE). The architecture separates the PIMS #1 prototype into three main parts with different responsibilities [Fernstrom, 86]:

- Tools

Management tasks are carried by tools. In PIMS a tool is a set of procedures which perform all the necessary processing related to a project management task, like scheduling, resource allocation and tracking. A tool does not perform any input/output, user interaction or communication with another tool directly, but relies on other parts of PIMS for doing this. Treating tools and other parts of PIMS as objects means, that new tools or new versions of existing tools or other functions may be dynamically connected to PIMS without the need for modifying the existing code.

- Information Manager

All globally accessible data within the PIMS system is maintained by an Object Base Management System, called the Information Manager. The Information Manager provides a format for the internal representation of entities within PIMS. This format is mainly used to define the PIMS Project Conceptual Schema, it is capable of representing the range of concepts, that exist within PIMS and the relations between these concepts. The chosen structure is a semantic net of frame based objects. The Information manager possesses methods to access the objects which are manipulated by PIMS, and all the necessary functionalities for managing the PIMS tools. Obviously, any data that is produced by a tool is stored in the earlier mentioned PIMS Project Conceptual Schema.

- Front End

The Front End handles the user interaction and is responsible for selecting the chosen tool functions. The Front End has been split into three components:

* Dialogue Manager
The aim of this component is to interpret the user interaction in order to activate a tool or a tool command (if the user is allowed to activate this tool or command and the evaluation of the activation conditions is successful), to obtain help information, or to determine a PIMS session.

* Visualizer
This is an Information manager object that implements a display action and an associated user interaction. A visualizer is connected to a subwindow and handles display requests (messages) sent to a subwindow instance. The visualizer definitions for form, list and text windows are fixed (their display action and interactions are always the same), but a set of different visualizers are available for graphical subwindows; for instance, there are special visualizers for displaying Gantt Charts, networks and trees.

* Display manager
This component is concerned with the windowing and graphics system. In PIMS #1, the Programmable Computing Environment [Anjewierden, 86], PCE, is used as Display Manager. PCE is a completely object oriented interface to the standard windowing system on Sun workstations, the Sun Visual/Integrated Environment for Workstation, Sunview.
PCE is written in C and it runs as a separate process form the rest of PIMS. The communication with PCE is performed via UNIX-pipes.

The overall architecture of PIMS #1 is shown in figure 6.

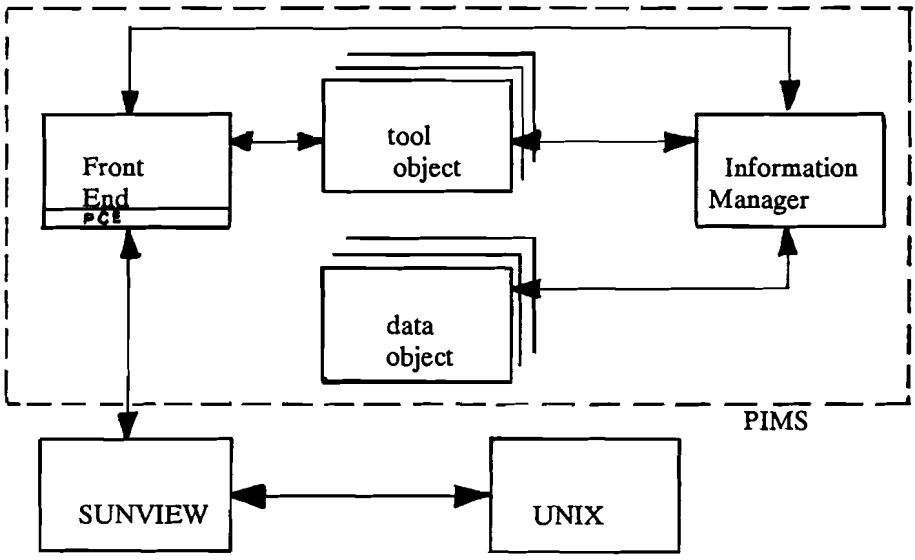


Figure 6: The overall architecture of PIMS #1 (arrows indicate messages to send)

7. Data Modelling Language

The Information manager provides a format for the internal representation of entities within the PIMS system. This format for formalizing the entities within PIMS is called the Data Modelling Language (DML) and is mainly used to define the PIMS Project Conceptual Schema. The DML must be capable to describe the objects that exist within the PIMS system and their relations. The chosen model is a semantic net of frame based objects, comparable with the structure available in commercial expert shells, like Kee and Goldworks.

A frame is a structure that contains a set of named attributes, called slots, which together describe a concept, object, in much the same way as fields in a record. Slots can be attached to simple facts, rules, frames and procedures. Unlike records in conventional languages, which can contain only declarative information (such as integers), slots can have procedural information (methods). Slots can contain subslots, called facets, which contain knowledge about the information in the slots. A frame in PIMS contains commonly used facets, such as Type, Value, IfNeeded, Default, and AfterModify.

The IfNeeded facet gives a procedure for calculating the value of the slot if no other value is attached to the slot. For example, when an attribute of the class Task is EffortSpent, an IfNeeded facet can be used to calculate the value of this attribute, because this value is equal to the sum of the effort spent of the resources allocated to the task. Default facets are available to provide a slot value if no other method is successful or available.

An AfterModify facet can contain procedural information. It specifies an action to be taken when the value in the slot is modified. Such procedural attachments, called demons, provide data driven infering (forward chaining) by causing procedures to execute, whenever the slot is modified. Demons are triggered automatically; the system checks for the existence of an AfterModify facet when it attaches a new value to a slot. Frames that describe slots and the types of values they have are called classes. Frames that represent actual objects are instances of a frame.

In the following we will describe the syntax used to define a physical class in the PIMS system. This description gives a clear impression of the PIMS frames and the possibilities of the Data Modelling Language [Doize, 87].

<physicalclass> ::= (DefClass <classname>
<superclasses>
<classdescription>
<attributes>
<instance-constraints>
<methods>
)

<classname> ::= symbol of the user package

<superclasses> ::= ({<classname>})
A class inherits the attributes, methods and constraints of its superclasses.

<instanceconstraints> ::= ({(<predicate>[<comment>])})
These predicates are Lisp forms describing relations between attributes of the same instance.

<classconstraints> ::= ({(<predicate>[<comment>])})
These predicates are Lisp forms describing relations between instances of the same class.

<methods> ::= ({((<methodname> <arguments> {<Lispforms>})
[<comment>])})
These methods are Lisp functions which are executed when an instance of the class receives the appropriate message. The forms are the function body; the value of the last form is returned, when the method is applied.

<attribute> ::= (<attributename> {<facet>})

<attributename> ::= symbol of the user package

<facet> ::=
(AttributeDescription <string>) |
This facet provides an informal description of the attribute.

(Type <type> | (<type> [<cardmin> [<cardmax>]])
[<typerestriction>]) |
This facet specifies the type of the attribute. When the attribute is multivalued, its value must be a list of elements of type <type>. The number of elements of this list must be between <cardmin> and <cardmax). If <cardmax> is not provided or nil, there is no maximum cardinality. If neither <cardmin> or <cardmax> are provided, there is no cardinality condition.

<typerestriction> ::= (Range{(<min> <max>)})|
(SetOfValues{<values>})

The first alternative means a value of the attribute must be contained in one of the intervals [<min>,<max>]. The second alternative means only the listed values are allowed for the attribute. These values must be of the same type of the attribute.

(AttributeConstraints {(<predicate>[<comment>])})|

Each predicate is a Lisp form representing a condition which must be verified by the attribute value.

(Value[<initvalue>])|

This facet indicates, that the attribute is settable. If there is no such facet associated to the attribute, this is not settable and its value can be obtained only through the IfNeeded or the Default facets. <initvalue> allows to define a default initialization value for the attribute.

(IfNeeded{(<Lispform>[<comment>])})|

When an attempt is made to read an attribute whose value is undefined, the functions given by the IfNeeded facet are used. They are applied in the order they have been given, until one of them succeeds.

(Default <defaultvalue>)|

This facet allows to define a default value for the attribute. It is used, if the attribute value is undefined and could not be calculated by an IfNeeded function.

(AfterModify{(({<event})<Lispform>[<comment>])})|

This facet allows to define data propagations. The given Lisp forms will be automatically applied after every modification of the attribute's value. According to the event which has resulted in the modification the corresponding functions are applied.

(ReverseAttribute <attributename>)

The attribute for which this facet is defined has a reverse attribute; this means they represent semantically the same relation: if a link is created, its reverse link has to be created too; on the other hand if a link is suppressed, its reverse link must be suppressed too.

Any programmer who fails to comply with the standard naming, formatting or commenting should be shot.

M. Spier

8. Tool Modelling Language (TML)

In order to formalize the PIMS system view of a tool, a Tool Modelling Language has been defined. In principle, a tool could be described as a class using the Data Modelling Language, but since many of its characteristics must be known to the Front End, a specified Tool Modelling Language has been defined [Fernstrom, 87]. A TML description of a tool contains all the information related to a tool that is needed for the Front End to build up a user interface.

The formal TML description of a tool is processed by the Tool Compiler that produces a tool class and other information used at run time. The TML supports the description of the following aspects of a tool:

- object description of the tool: attributes and methods
- textual information about the tool: used for short or long help
- activation condition of the tool: criteria that must be verified before activation
- specification of the commands (functions that may be activated by the user): the syntax of the commands and their relation with tool methods
- state transition table describing precedence rules between commands
- definition of the static tool window: definition of all the static subwindows used by the tool
- language dependency: how user messages have to be expressed in different (natural) languages

A tool definition contains two kinds of information:

- internal information used by the PIMS system, which is independent of the (natural) language used in communication with the user
- text that will be displayed on the screen; this text is dependent of the user language

For each language a number of user messages is defined in the form of strings or lists of strings. Each user message has a reference number. This number is used in the language independent part to associate the user message.

To give an impression of the possibilities of the TML we give a short description of this modelling language. On the highest level, the syntax of a TML description is as follows:

```
<ToolDescription> ::=
    (:Tool  <ToolIdentifier>
      (:ToolExternalName  <StringNumber>)
      [(:Languages  {<Language>})]
      [(:ToolIcon  <FileName>)]
      [(:SuperTools  {<ToolIdentifier>})]
      [(:ImportClass  {<ClassName>})]
      (:ToolShortInfo  <StringNumber>)
      [(:ToolAims  <PacketNumber>)]
      [(:ToolPreConditions  <PacketNumber>)]
      [(:ToolEffects  <PacketNumber>)]
      (:UserStatus  {<UserStatus>})
      [(:ToolActivationCondition  {<ActivationConditions>})]
      (:ToolCommandSpecification  {<CommandGroup>})
      [(:TransitionTable  {<StateTransition>})]
      [(:SubWindows  <WindowDescription>)]
      [(:Attributes  {<Attribute>})]
      (:Methods  {<Method>})
      (:DependentPart  {<LanguageDependency>})
    )
```

The slot Tool contains the internal name of the tool: this symbol is used as name of the Information Manager class of the tool.

The slot Tool ExternalName refers to the name that the PIMS user will see in a button in the PIMS main window (see next chapter).

The slot Languages contains the list of languages for which external strings are described.
The slot DependentPart describes the strings for the languages taken into account by the tool.

The slot ToolIcon contains the definition of a small graphical object used to represent the tool on the user's screen.

The object description of the tool has four slots: SuperTools, ImportClass, Attributes and Methods. After the keyword SuperTools appears a list of ToolIdentifier's; the current tool inherits all the attributes and methods of the tools defined in this list. After the keyword ImportClass appears a list of ClassName; these classes are used by the tool, but defined in other modules. All the classes defined in this list can be used as type for tool attributes.

After the keyword Attributes appears a list of slots Attribute. Except for the keyword Attribute an attribute has the same syntax as in the DML (see previous chapter). An attribute may be typed by using a predefined type of the Information Manager (e.g. string or integer) or a class defined in the PIMS Project Conceptual Schema.

After the keyword Methods appears a list of slots Method. A slot Method has the following syntax:

```
<Method> ::=
        (:Method  <MethodName>
          ([{<Parameter>}])
          <Lispcode>
        )
```

The methods are the entry points of the tool. In the TML description of the Tracker Tool, the <Lispcode> is a function call; in this way it is possible to compile the TML part and the functions (representing the methods) separately. A method is activated either by the Front End after the user has given the corresponding command or directly by a tool through a message.
The textual information about the tool (help) has four slots:

- ToolShortInfo, which refers to a string representing a short help text about the tool.
- ToolAims, which refers to a list of strings describing the aims of the tool.
- ToolPreConditions, which refers to a list of strings describing when the tool can be activated.
- ToolEffects, which refers to a list of strings describing the modifications of the Information Manager objects made by the tool.

The activation condition of the tool is defined by the slots UserStatus and ToolActivationCondition. The keyword UserStatus is followed by a list of user statuses; only users that have one of these statuses can activate the tool.

ToolActivationCondition is a list of ActivationCondition's:

```
<ActivationCondition> ::=
        (:ConditionCheck  <Predicate> <StringNumber>)
```

The predicate is a Lisp expression that will be evaluated. If it fails, the user message referenced by the StringNumber is displayed and the tool will not be activated, otherwise an instance of the tool is created.

The command specification of the tool is a two level structure. Each first level description describes a command group. The specification of the commands is defined by the slot ToolCommand-Specification. This slot contains a list of CommandGroup's, where each element defines a group of commands.

```
<CommandGroup> ::=
      (:CommandGroup  <GroupIdentifier>
        (:GroupExternalName  <StringNumber>)
        (:GroupShortInfo  <StringNumber>)
        [(:GroupAims  <PacketNumber>)]
        [(:GroupEffects  <PacketNumber>)]
        <CommandDescriptor>{<CommandDescriptor>}
      )
```

GroupIdentifier is the internal name of the group of commands.

The slot GroupExternalName refers to the external name, the PIMS user will see in a menu bar of the tool dialogue subwindow (see next chapter).

The slot GroupShortInfo refers to a short help string describing the group; it can be displayed using the help facilities.
The slots GroupAims and GroupEffects refer to the textual explanations with respect to the commands of the group.

A <CommandDescriptor> contains the complete description of a command, it describes how a method defined in the slot method can be activated.

```
<CommandDescriptor> ::=
    (:Command  <CommandIdentifier>
      (:MethodName  <Methodname>)
      (:CommandExternalName  <StringNumber>)
      (:CommandWriteAccess  <WriteAccess>)
      [(:UserStatus  {<UserStatus>})]
      (:CommandShortInfo  <StringNumber>)
      [(:CommandAims  <PacketNumber>)]
      [(:CommandPreconditions  <PacketNumber>)]
      [(:CommandEffects  <PacketNumber>)]
      [(:CommandActivationCondition  {<ActivationCondition>})]
      [(:ParameterDesc  {<Parameterdescriptor>})]
      [(:CommandRestriction  {<ConditionCheck>})]
    )
```

The <CommandIdentifier> contains the unique internal name of the command; it is used to reference the command in the state transition table and in popup menus defined in subwindows (see next chapter).

The slot CommandExternalName refers to the external name, which appears in a popup window when the PIMS user selects a group.

The slot CommandWriteAccess defines if the tool function (method) will modify data in the Information Manager.

The slots UserStatus, CommandShortInfo, CommandAims, CommandPreConditions, CommandEffects and CommandActivationCondition mean the same as the with these 'command' slots comparable 'tool' slots (see above).

If a command has parameters, they are described by the <ParameterDescriptor> that contains information used by the Front End to build a parameter acquisition window for the parameter input. The list of the <Parameterdescriptor> in the command specification, must be ordered as the parameters mentioned in the associated method.

<ParameterDescriptor> ::=
    <AtomParameter> | <ListParameter>

An AtomParameter has only one value, a ListParameter has a list of values.

<AtomParameter> ::=
    (:AtomParameter <ParameterIdentifier>
      (:ParameterPrompt <Stringnumber>)
      (:ParameterShortInfo <StringNumber>)
      [(:ParameterLongInfo <PacketNumber>)]
      (:ParameterType <ParameterType>)
      [(:SelectionPath <SelectionPath>)]
      [(:DefaultValue <LispCode>)]
      [(:Optional)]
      [(:ConditionCheck <Predicate> <StringNumber>)]
)

<ListParameter> ::=
    (:ListParameter <ParameterIdentifier>
      (:ParameterPrompt <StringNumber>)
      (:ParameterShortInfo <StringNumber>)
      [(:ParameterLongInfo <PacketNumber>)]
      (:ParameterType <ParameterType>)
      [(:SelectionPath <SelectionPath>)]
      [(:DefaultValue <LispCode>)]
      [(:EachParCheck <Predicate> <ErrorMessage>)]
      [(:AllParCheck <Predicate> <ErrorMessage>)]
)

A <ParameterIdentifier> is the local name of the parameter.

The slot ParameterPrompt refers to a short string used as prompt in the parameter acquisition window.

The slot ParameterType is the type of the Parameter.

The slot SelectionPath is the name of an attribute of the class defined in the slot ParameterType. The value, given by the PIMS user via the keyboard, is replaced by the Information Manager instance of the class (defined by the slot ParameterType) for which the attribute SelectionPath has the value given by the PIMS user. If the slot SelectionPath is not provided the Key of the class is used.

The slot DefaultValue is the value assigned to the parameter before the user gives a value.

For an AtomParameter, if the slot Optional is present, no value has to be given by the PIMS user for this parameter; in this case the DefaultValue is used.

For an AtomParameter, the slot ConditionCheck describes the condition that the parameter value must fulfil.

For a ListParameter the slots EachParCheck and AllParCheck describe conditions that each new value and the list of values must satisfy.

The slot CommandRestriction defines checks on the parameter values; the command can be activated only if these checks are successful.

In Appendix 1 the command EnterEndDate of the Tracker Tool is shown. This command, which is one of the commands of the group EnterResourceData, has three atom parameters:

- PersonKey (the name of a resource).
- TaskName.
- EndDate.

The Front End performs the following parameter condition checks:

- if the PersonKey belongs to a human resource which has been allocated to at least one task, that belongs to the current project status (check_task_allocated function in Appendix 1)
- if the TaskName is the name of a not acknowledged task, that belongs to the current project status (check_task_not_ack function in Appendix 1)
- if the EndDate is an existing date and is not later than the current date (valid_date_check_function in Appendix 1)

There is one CommandRestriction:

- if the PersonKey belongs to a resource that has been allocated to the task which belongs to the entered TaskName (check_ResAllocation function in Appendix 1)

The with the command associated method ENTER_END_DATE performs the following additional checks:

- if the EstimateToComplete value is zero
- if the entered EndDate is equal to or later than the StartDate of the resource on the task

If the parameter values are valid the method stores the EndDate and its display in a textwindow data related to the tasks allocated to the entered resource.

The state transition table, defined in the slot TransitionTable, is to express a logical dependency between command activation conditions.

The slot SubWindows contains a static description of the tool main window and the subwindows of the tool. It describes the tool's window as it will (initially) appear on the screen. It defines the size of the main window in pixels and the relative position and the size of the subwindows that appear inside the tool main window.

In addition to statically defined windows, a tool can deal with Popup windows. These windows can be dynamically created and destroyed by the tool. For their manipulation a set of functions has been defined. Most dynamic subwindows are completely defined in the code of the methods of the tool. The exception of this is the popup FormSubWindow, used for forms, which needs a <SubWindowDescriptor> in the TML description.

In PIMS, four types of subwindows have been defined:

- TextSubWindow, which can be used for displaying text (strings). It has a vertical scroll bar.
- GraphicalSubWindow, which can be used for displaying graphics, like Gantt charts, networks, and trees. The user may pick with the mouse displayed PIMS objects, which can be used as parameter values to tool commands. Such a window has vertical and horizontal scroll bars.
- ListSubWindow, which can be used to display lists of items. These are displayed as a list of strings, each referring to a PIMS object. A displayed item may be picked with the mouse and the corresponding object can be used as a parametervalue to a tool command. It has a vertical scroll bar.
- FormSubWindow, which can be used to display forms, which have to be filled in by the user. This is an alternative way for the tool builder to acquire information from the user (the other way is the command acquisition handled by the Dialogue Manager).

The syntax of the <WindowDescription> is:

```
<WindowDescription> ::=
        (:WindowSize   <HorizontalSize> <VerticalSize>)
        (:WindowType   {<SubwindowDescriptor>})
        (:WindowGrid   <SubWindowTopology>)
<SubWindowDescriptor> ::=
        (:SWDescriptor   <String>
           (:SWType   <SubWindowType>)
          [(:SWPopup   <SubWindowPopup>)]
          [(:GDisplaySize   <HorizontalSize> <VerticalSize>)]
        )
<SubWindowtype> ::=
        :TEXTSUBWINDOW|
        :GRAPHICALSUBWINDOW|
        :LISTSUBWINDOW|
        :FORMSUBWINDOW|
```

In Appendix 2 the main window description of the Tracker Tool is shown. This window will be described further in the next chapter.

To give an impression of the size of the Tracker Tool (including forecasting), the lines of code will be given:

- TML description: 1400 lines of code;
- Lisp functions: 3500 lines of delivered source instructions.

9. Using a tool in PIMS

When a PIMS environment (Information Manager, Dialogue Manager, Visualizers, tools) is invoked, a PIMS main window is created. For each tool present in the environment, this window contains a button in which the string defined by the ToolExternal-Name is displayed. This button is used to activate or to display information about the associated tool.

In the PIMS main window, two lines are reserved to display error messages and help text.

To activate a tool the PIMS user has to select the associated button in the PIMS main window. This activation is only possible if the preconditions have been evaluated, i.e. the status of the PIMS user appears in the slot UserStatus and the evaluation of the predicates defined in the slot ToolActivationCondition are successful. If the tool activation is possible, an instance of the tool (-class) is created. This means, among other things, the tool main window, which contains the tool dialogue subwindow ('automatically' built by the Front End) and the tool subwindows (defined in the TML description of the tool), is created. In figure 7 the tool main window of the Tracker Tool is shown.

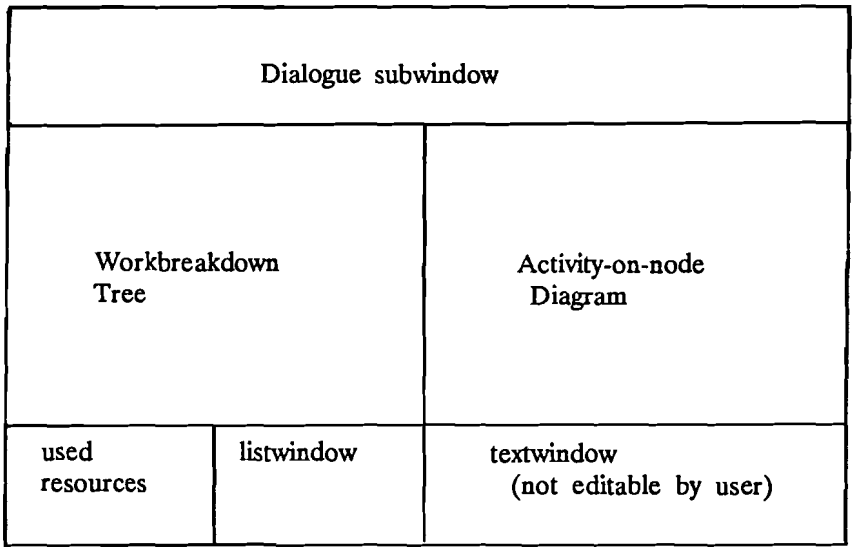


Figure 7: The tool main window of the Tracker Tool

The tool dialogue subwindow is used by the Dialogue Manager for command processing. It contains a menu bar with a button to QUIT the tool, and two lines to display error messages and help information. Further the menu bar contains the GroupExternalName of all the command groups. When the user selects, with the left button of the mouse, one group, a popup menu, that contains all the CommandExternalName of the commands belonging to the group, will appear. The selection of one of these commands is sent to the Dialogue Manager as a wish to activate the command. A command can only be activated if the status of the PIMS user is correct according to the content of the slot UserStatus and if all the ActivationCondition's are verified.

If the command has no parameters the with the command associated method is activated. Otherwise the so-called Parameter Acquisition Window is shown. This window has two main goals: display all the parameter values yet furnished (including default values) and allow the user to modify these values. The acquisition window consists of:

- two buttons for Validate and Cancel the command input and two lines to display error messages and help information
- a parameter prompt, containing the string defined in the slot ParameterPrompt; the help information is the string defined in the slot ParameterShortInfo. Initially, the editable field of the parameter is either empty or it contains the default value optionally specified by the slot DefaultValue
- for every ListParameter there is also a list subwindow displaying all the values given so far. One of these is the current value (highlighted) which is the one shown in the editable field of the parameter

The actions the PIMS user can perform are:

- select a parameter prompt using either the keyboard or the mouse; the parameter belonging to this prompt becomes the current parameter
- use the keyboard to enter a new or to modify an existing value in the editable field
- use the left button of the mouse to select an object in any other subwindow, containing selectable objects. The object selected will be assigned to the parameter associated with the current editable field, and a textual representation of the object will be displayed
- update the list of values of a ListParameter (add or delete values)
- select one of the buttons Validate or Cancel; a Cancel aborts the command processing and a validate triggers the final check of all the parameter values according to the condition check in the slot CommandRestriction. If this is successful the associated method is activated

Obviously, each new value for the current parameter is checked according:

- to the ParameterType
- to the ConditionCheck for an AtomParameter or to the EachParCheck and AllParCheck for a ListParameter.

If the new value is incorrect an error message is displayed and nothing is changed for the command currently handled. When the new value supersedes the previous value, for a ListParameter the new value is added at the end of the previous list of values.

10. References

[Anjewierden, 86]
Anjewierden, A., "PCE-Prolog Reference Manual", University of Amsterdam, 1986.

[Booch, 86]
Booch, G., "Object-Oriented Development", IEEE Trans. Software Engineering, Feb. 1987.

[Doize, 87]
Doize, M., Fernstrom, C., "User's manual for the PIMS #1 Information Manager", CSI, CSI-F-SP-004f, 1987.

[Fernstrom, 86]
Fernstrom, C., "System Architecture", CSI, CSI-E-SP-002, 1986.

[Fernstrom, 87]
Fernstrom, C., Konc, S., Paris, J., "Tool Builder's Handbook for PIMS #1", CSI, CSI-F-SP-003c, 1987.

[Goldberg, 83]
Goldberg, A., Robson, D., "Smalltalk 80: The language and its implementation", Addison-Wesley, 1983.

[McDermott, 82]
McDermott, J., "R1: A Rule-Based Configurer of Computer Systems", Artificial Intelligence, Vol 19, N° 1, 1982.

[Meyer, 87]
Meyer, B., "Reusability: The case for Object-Oriented Design", IEEE Software, March 1987.

[Nijssen, 82]
Nijssen, G.M., Vermeir, D., "A procedure to define the object type structure of a conceptual schema", Inform. Systems, Vol 7, N° 4, 1982.

[Steele, 84]
Steele, G.L.Jr., "Common Lisp: The Language", MA: Digital Press, 1984.

[Turing, 50]
Turing, A., "Computing machinery and intelligence", Mind, 59, October 1950. Reprinted in Feigenbaum, E., Feldman, J., "Computer and Thought", McGraw Hill, 1963.

[Verheyen, 82]
Verheyen, G.M.A., Van Bekkum, J., "NIAM: An Information Analysis Method", in Olle, T.W., Sol, H.G., Verrijn Stuart, A.A., "Information System Design Methodologies - A Comparative Review", North Holland Publ. Co., 1982.

[Wissing, 87a]
Wissing, R., "Modelling, tracking and predicting in software project management",BSO, BSO-G-RR-102, 1987.

[Wissing, 87b]
Wissing, R., "Tracker Tool Specifications", BSO, BSO-G-SP-104, 1987.

[Wissing, 88]
Wissing, R., "Predictor Specifications", BSO, BSO-G-SP-108, 1988.

[Yourdon, 79]
Yourdon, E., Constantine, L., "Structured Design", Prentice Hall, 1979.

Appendix 1

The TML description of the command EnterEndDate.

```
(:Command EnterEndDate
    (:MethodName ENTER_END_DATE)
    (:CommandExternalName 73)
    (:CommandWriteAccess t)
    ;(:UserStatus {<UserStatus>})
    (:CommandShortInfo 74)
    ;(CommandAims <PacketNumber>)
    ;(:CommandPreconditions <PacketNumber>)
    ;(:CommandEffects <PacketNumber>)
    ;(:CommandActivationCondition {<ActivationCondition>})
    (:ParameterDesc
      (:AtomParameter
        PersonKey
        (:ParameterPrompt 62)
        (:ParameterShortInfo 63)
        ;(:ParameterLongInfo <PacketNumber>)
        (:ParameterType Person)
        (:SelectionPath Key)
        ;(:DefaultValue
        ;   <Lispcode>
        ;) ;end DefaultValue
        ;(:Optional)
        (:ConditionCheck
          (tr::check_task_allocated $*PersonKey*$
                                    user::selfinst
          )64
        )
      )
      (AtomParameter Taskname
        (:ParameterPrompt 52)
        (:ParameterShortInfo 53)
        ;(:ParameterLongInfo <PacketNumber>)
        (:ParameterType Task)
        (:SelectionPath Name)
        ;(:DefaultValue <LispCode>)
        ;(:Optional)
        (:ConditionCheck
          (tr::check_task_not_ack $*TaskName*$)
          54
        ) ;end ConditionCheck
      ) ;end AtomParameter
```

```
        (:AtomParameter  EndDate
          (:ParameterPrompt  75)
          (:ParameterShortInfo  76)
          ;(:ParameterLongInfo  <PacketNumber>)
          (:ParameterType  Integer)
          ;(:SelectionPath  <SelectionPath>)
          ;(:DefaultValue  <LispCode>)
          ;(:Optional)
          (:ConditionCheck
            (tr::valid_date_check  $*EndDate*$
                                   user::selfinst)
            )77
          ) ;end  ConditionCheck
        ) ;end  AtomParameter
      ) ;end  ParameterDesc
      (:CommandRestriction
        (:ConditionCheck
          (tr::check_ResAllocation  $*PersonKey*$
           $*TaskName*$  user::selfinst
          )
          78
        ) ;end  ConditionCheck
      ) ;end  CommandRestriction
) ;end  Command
```

Appendix 2

The TML description of the Tracker Tool's main window.

```
(:SubWindows
     (:WindowSize 1100 732)
     (:WindowType
       (:SWDescriptor ResList
         (:SWType :LISTSUBWINDOW)
       )
       (:SWDescriptor ItemList
         (:SWType :LISTSUBWINDOW)
       )
       (:SWDescriptor Sched
         (:SWType :GRAPHICALSUBWINDOW)
         (:GDisplaySize 6000 3000)
       )
       (SWDescriptor Wbd
         (:SWType :GRAPHICALSUBWINDOW)
         (:GDisplaySize 4000 3000)
       )
       (:SWDescriptor Text
         (:SWType :TEXTSUBWINDOW)
       )
     )
     (:WindowGrid
       (Wbd       Wbd     Sched     Sched     Sched)
       (Wbd       Wbd     Sched     Sched     Sched)
       (Wbd       Wbd     Sched     Sched     Sched)
       (Wbd       Wbd     Sched     Sched     Sched)
       (ResList   ItemList  Text     Text      Text)
       (ResList   ItemList  Text     Text      Text)
     )
) ;end SubWindows
```

Index

Modelling, tracking, and predicting in software project management

ABSTRACT

This document discusses an analysis of modelling, tracking and predicting in software project management. It serves as starting point for describing the specifications of two tools, concerning tracking and predicting respectively.

Authors : RF Wissing, BSO/Eindhoven
Reader : RK Dixon
Level : BSO/Eindhoven BV
Status : InCirculation
Identifier: BSO-G-RR-102 (Release : 1.6)
Created : 87/10/01 (Edited : 88/01/22)
Printed : 22 January 1988

This document is part of a project partially funded by the Commission for the European Communities ESPRIT programme, as project number 814.

© 1988 BSO/Eindhoven BV

Distribution List

Christer Fernstrom	CSI
John Hawgood	PA
Robbert de Hoog	UvA
Patrick Humphreys	LSE
John Jenkins	938
Hans van de Klok	BSO
Frank Land	LBS
Eirik Naess-Ulseth	SI
Mark Reilly	tTI

Modelling, tracking, and predicting

Revision Information

Creation Date:
Author :
Reader :
Modifications:

Old ideas give way slowly; for they are more than abstract logical forms and categories. They are habits, predispositions, deeply ingrained attitudes of aversion and preference.

John Dewey

Modelling, tracking, and predicting

Table Of Content

When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.

Lord Kelvin

Modelling, tracking, and predicting

1. Modelling

1.1. Introduction

When a computer system has to be suitable for helping the project manager to manage his software project in an intelligent way, it is clear that a model must be made of the project (software development process). A model is a miniature representation of a complex reality [DeMarco, 82]. A model reflects certain selected characteristics of the piece of the real world it stands for. Obviously, a model is only useful if it represents accurately those characteristics that are of interest at a certain moment. If there is a need to examine different characteristics at different times, it may be necessary to build several models of the same 'system' (project). Naturally, these models have to be consistent.

Because it is impossible to oversee the project as a whole, one has to divide the project into pieces, the tasks, and define the interfaces between them, the deliverables. This means an initial Work Breakdown Structure (WBS) has to be made. To each task identified in the WBS an effort must be allocated. This task-effort will be estimated with the aid of an effort estimation model. Naturally, this model has to fit in the overall project model. Part of building an effort estimation model is specifying the parameters which are assumed to be relevant for the tasks, and allocating appropiate values to them. It is quite possible that the values of the estimation model parameters must be corrected during scheduling the tasks and/or resource allocation.

Since the models give the manager a way to study essential aspects of the project, such models play an important role in making the project plan. This plan will be used in tracking the work progress. When tracking indicates that there is a serious deviation from the plan, one has to know how the project plan has been made (in particular, what parameters are used and what values are assigned to them).

1.2. Macro-models for cost/effort estimation

Models can be grouped in four categories [Conte et al., 86]:

- historical-experiential models;
- statistically-based models;
- theoretically-based models;
- composite models.

In the next sections the above mentioned kinds of models will be discussed.

1.2.1. Historical-experiential models

Most of the effort estimation methods commonly used today fall into the category of experiential models. In its simplest form, one or more experts are asked to make estimates about the effort required, either for the total project or for the pieces into which the project has been divided. In doing so, the 'estimators' rely on their own experience with similar projects, and/or on intuition, and/or on historical information about completed projects. If more than one estimator is involved, there are a number of ways to combine them to the 'best' starting estimate. One way is simply to compute the mean or median estimate of all the individual estimates. This method is quick, but subject to adverse bias by extreme estimates. Another way is to hold a group meeting to get the 'estimators' to converge on, or at least agree to a single estimate. This procedure has the advantage of filtering out uninformed estimates in general, but it has two main drawbacks. Firstly, group members may be influenced by the more glib and assertive members. Secondly, group members may be overly influenced by figures of authority or political considerations. One technique that has been used to avoid the drawbacks of the group meeting is the Delphi method, a method of multi-source estimating [Boehm, 81]. The steps involved in the standard Delphi technique are the following:

1. Coordinator presents each expert with a specification and a form upon which to record estimates.
2. Experts fill out forms anonymously. They may ask questions of the coordinator, but should not discuss the situation with each other.
3. Coordinator prepares a summary of the experts' estimate, and the rationale behind the estimate (see figure 1.1).
4. Experts fill out forms, again anonymously, and the process is iterated for as many rounds as appropriate.
5. No group discussion is to take place during the entire process.

Boehm has proposed an alternative method which he called the Wideband Delphi method. The steps of this methods can be summarized as follows:

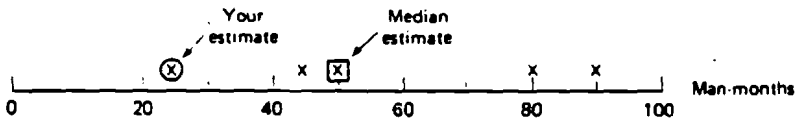
1. Coordinator presents each expert with a specification and an estimation form.
2. Coordinator calls a group meeting in which the experts discuss estimation issues with the coordinator and each other.
3. Experts fill out forms anonymously.
4. Coordinator prepares and distributes a summary of the estimates on an iteration form (similar to figure 1.1, but excluding the written rationale).
5. Coordinator calls a group meeting, specifically focusing on having the experts discuss points where their estimates varied widely.
6. Experts fill out forms, again anonymously, and Steps 4 to 6 are iterated for as many rounds as appropriate.

The Wideband Delphi method has subsequently been used in a number of

DELPHI COST ESTIMATION ITERATION FORM

PROJECT: Operating System DATE: 6/21/83

HERE IS THE RANGE OF ESTIMATES FROM THE 1st ROUND



PLEASE ENTER YOUR ESTIMATE FOR THE NEXT ROUND: 35 MM

PLEASE EXPLAIN ANY RATIONALE BEHIND YOUR ESTIMATE:

This looks like a standard process control operating system. The development team has had a lot of experience with such systems, and should have no trouble with this one.

Figure 1.1 Typical Delphi iteration form [Boehm, 81]

studies and cost estimation activities [Boehm, 81]. It has been successful in combining the free discussion advantages of the group meeting method and the advantages of anonymous estimates of the standard Delphi method.

Naturally, the estimating methods, discussed above, are subjective working-methods, which are highly dependent on the competence and objectivity of the 'estimators'. Such methods run the risk of overlooking activities which may be unique to the current project. The methods just described can be applied at either the overall project level or at the task level. These are commonly referred to as top-down estimating (overall project level) or bottom-up estimating (task level). The major advantage of top-down estimating is its overall project level focus. So, it will not miss the top level tasks such as integration, users' manuals, configuration management, quality assurance, project management, etc. The major disadvantages of top-down estimating are, that it often does not identify some low level problems that are likely to escalate costs, that it sometimes misses components of the software product to be developed, that it provides no detailed basis for cost justification and iteration, and that it is less stable than a multicomponent estimate, in which estimation errors in the components have a change to balance out [Wolverton, 74]. All in all the top-down method (with expert judgement) is not recommended for substantive estimates. Its principal use is for early, quick, and cheap (and inaccurate) estimates at the outset of a product development.

Bottom-up estimating is complementary to top-down estimating, in that its weaknesses tend to be top-down 's strengths, and vice versa. Thus the bottom-up estimate tends to cover just the effort associated with the bottom-level tasks, and to overlook many of the top-level functions. In the bottom-up estimate, the effort of each software component is estimated by an individual, often the software engineer, who will be responsible for developing the component. The usual way to ensure that all the tasks are included in a bottom-up estimate is to organize the software product into a Work Breakdown Structure (WBS) which includes

not only the product hierarchy of the software product components, but also the activity hierarchy of projects jobs to be done. This ensures that the costs of such activities as integration and configuration management are included. From the WBS, very accurate estimates can be done by experienced software engineers. The drawback to the bottom-up method is, that it requires much more effort than a top-down estimate does.

A well-known historically based model is the TRW Wolverton model [Wolverton, 74]. It derives the software development cost from a software cost matrix (see figure 1.2).

The elements of the matrix (Ci.j) are costs per line of code (clearly, effort per line of code is also possible), which depend on software type and software difficulty. The first step in the TRW - Wolverton method is to identify the types of all modules within the proposed software product. Six types are proposed: control, Input/Output, pre/post processor, algorithm, data management, and time critical. The second step is to estimate the complexity or difficulty of the module based on a six-point scale. The scale proposed in the Wolverton model consists of old-easy, old-medium, old-hard, new-easy, new-medium, and new-hard. The costs (Ci.j) are derived from historically maintained data from completed projects. The third step is to estimate the size of each module in lines of code. If k is a module with an estimated lines of code S(k),and if this module is of type i(k) and difficulty j(k), then the cost of developing module k is:

$$C(k) = S(k)C_{i(k)\,j(k)}$$

The overall software product cost will then be obtained by summing the cost over the modules:

$$\text{Overall Cost} = \sum_{\text{all modules}} C(k)$$

Although this approach does introduce a certain amount of objectivity, there remains a great deal of subjectivity in the entire method. Furthermore, the influence of several other factors on cost is not adequately taken into account (for instance computer availability). One could, of

	old easy	old medium	old hard	new easy	new medium	new hard
1. control	C1.1	C1.2	C1.3	.	.	.
2. I/O	C2.1	C2.2	.	.	.	.
3. pre/post processor	.	.	.	.	.	.
4. algorithm	.	.	.	.	.	.
5. data management	.	.	.	.	.	.
6. time critical	.	.	.	.	.	.

Figure 1.2 A software cost matrix

course, enlarge the size of the matrix by additional attributes that affect cost, but that would make the calibration of the matrix elements much more difficult.

Estimating based on historical-experiential models is also called estimating by analogy. The main weakness of estimating by analogy is the risk of a mistaken analogy. It is not clear to what degree the comparitive project is actually representative of the constraints, techniques, personnel, and functions to be performed by the software product of the project being estimated.

Finally, one has to be alert for two pseudo-estimating methods, namely Parkinsonian estimates and Price-To-Win estimates. The Parkinsonian estimate is based on Parkinson's Law [Parkinson, 57], that runs as follows: "Work expands to fill the time available for it". Clearly, this is not a way of estimating, but a philosophy of despair. The Price-To-Win estimate is, like Parkinson 's Law, not an estimating method at all. With this method an estimator determines, what the market (client or manager) will bear and then underbids that limit. This method has won a large number of software contracts. The inevitable result is, that the money or schedule runs out before the job is done, everybody gets mad at each other, a lot of compromises are made about the software product to be delivered, and a lot of programmers work long hours just trying to keep the job from a complete disaster. Whatever the reasons are for using the Price-To-Win estimate, establishing effort and timescale budgets on this basis is very perilous practice in software engineering. One must never take the Price-To-Win estimate as the factual estimate of effort and timescale required.

1.2.2. Statistically-based models

Regression analysis is often used to determine the relationship between programming effort and other parameters. A large number of models, both linear and nonlinear, is known for effort estimation. Some of these models will be discussed in this section.

1.2.2.1. Linear statistical models

A general linear model will have the form

$$E = c_0 + \sum_{i=1}^{n} c_i x_i \ ,$$

where E is some unit of effort (e.g. man-months), c_i's are weighing-factors and x_i's are software attributes (other names for this term include cost driver, compensation factor and bias factor), that are assumed to affect the software development effort. There are literally hundreds of factors that may affect effort. However, for a specific software product many of these factors are in all probability insignificant and can therefore be ignored. Other factors are strongly correlated and can be combined into a single factor. In this way the number of

factors can be reduced. One of the earliest studies of linear models was conducted at the Systems Development Corporation (SDC) in the mid-1960's [Nelson, 66]. This study investigated 104 candidate attributes. Using a database of 169 projects an attempt was being made to determine the relevant coefficients using the least squares method. Different combinations of attributes were investigated, resulting in a regression model based on 14 attributes and leading to the formula:

$$E = -33.63 + 9.15x_1 + 10.73x_2 + 0.51x_3 + 0.46x_4 + 0.40x_5 + 7.28x_6 - 21.45x_7 + 13.5x_8 + 12.35x_9 + 58.82x_{10} + 30.61x_{11} + 29.55x_{12} - 0.54x_{13} + 25.2x_{14}$$

where E is effort in Man-Months (MM). The attributes x and their possible values are as follows:

	attribute	scale of value
x_1	lack of requirements	0 - 2
x_2	stability of design	0 - 3
x_3	percent math instructions	actual percent
x_4	percent storage/retrieval instructions	actual percent
x_5	number of subprograms	actual number
x_6	programming language	0 - 1
x_7	business application	0 - 1
x_8	stand alone program	0 - 1
x_9	first program on computer	0 - 1
x_{10}	concurrent hardware development	0 - 1
x_{11}	random access device used	0 - 1
x_{12}	different host, target hardware	0 - 1
x_{13}	number of personnel trips	actual number
x_{14}	developed by military organization	0 - 1

Even when applied to its own database of 169 projects, this model has a standard deviation which was unacceptably high; in other words this model is not a very accurate predictor. Further, we see in this model more remarkable things, for instance: a project, in which all the factors are rated at zero or near zero, has a negative effort; changing language from a higher-order one to an assembly language adds 7 MM to the effort, independent of the product size. The conclusion one can draw from the SDC-study was, that there are too many nonlinear aspects of software development for a linear cost-estimation model to work very well. Still, the SDC-study provided a valuable base of information and insight for effort estimation and future models.

1.2.2.2. Nonlinear statistical models

Most of the nonlinear models that are described in literature can be expressed in the form:

$$E = (a + bS^c)m(X),$$

where S is the estimated size of the software product expressed in (thousands of) lines of code; a, b, and c, are constants usually derived by regression analysis; and m(X) is an adjustment multiplier that depends on one or more compensation factors denoted in the formula by X. In some cases a, b, and c may also be functions of one or more compensation factors. Since m(X) can also be a nonlinear function of several variables, the given nonlinear function is too complex to lend itself readily to standard regression analysis techniques. Instead, it is more customary to derive a nominal estimator (usually based on least squares techniques) of the form:

$$E_{nom} = a + bS^c$$

and then to adjust this nominal effort with the compensation factor m(X).

Some nominal estimators that have been reported in literature are:

$E = 5.2S^{0.91}$	IBM-FSD model [Walston - Felix, 77]
$E = 3.5 + 0.73S^{1.16}$	University of Maryland model [Bailey-Basili,81]
$E = 2.4S^{1.05}$	COCOMO-model, organic mode [Boehm, 81]
$E = 3.0S^{1.12}$	COCOMO-model, semidetached mode [Boehm, 81]
$E = 3.6S^{1.20}$	COCOMO-model, embedded model [Boehm, 81]
$E = 5.288S^{1.047}$	($S \geq 10$) Doty-model [Herd et al., 77]

In these models the primary factor affecting effort is assumed to be the size of the software product in lines of code in thousands. However, when comparing models, we must be alert that the definition of lines of code can be different. The COCOMO and Doty models, for instance, do not consider comment lines. On the other hand the IBM-FSD model does include comment lines. Further, S must be adjusted when a software product contains both new code and old or adapted code. Unfortunately, there is no general agreement about how to do this. Bailey and Basili for instance, define S as the sum of the new code plus 20 percent of all old code, including comment lines. However, COCOMO uses a much more complicated formula that takes into account the percentage of design, code and integration changes required to adapt the old code, excluding comment lines. Similary, we must be aware of just what effort is being estimated: that is, E may represent simply the coding effort or at the other extreme E may represent the total specification, design, coding, testing, and maintenance effort.

The most serious objection to any model, that uses lines of code as an

indication of volume, is that one cannot count lines of code at the beginning of a project; one has to estimate it. But it is highly questionable whether an estimator is better at estimating lines of code than he is at estimating resultant development cost.

One ought to realize, the fact that a model cannot be used quite successfully from the beginning of a project, this does not mean that it is of no use at all. It may still be a valuable aid for producing predictions while the project is in progress.

A suggestion to avoid the 'lines of code' problem is using function points as proposed by Albrecht [Albrecht 79, 83]. The main concept of Albrecht's Function Point's is counting the number of discrete functions made available to the final user of the software product. There are five function types:

- external input;
- external output;
- logical internal files;
- external enquiry;
- external interface file.

It is believed that this often relates to the work involved, at least after compensating for complexity and other characterizations.

To get the total (unadjusted) function point count the various counts are multiplied by the values shown in figure 1.3 and then added together.

The table given in figure 1.3 originated with Drummond [Drummond, 85]. The initial matrix proposed by Albrecht has, for instance, only three classes of complexity.

	COMPLEXITY					
Function Type	simple	moderate	average	complex	very complex	TOTAL
External Input (Count)	EICx2	EICx3	EICx4	EICx5	EICx6	=
External Output	EOCx3	EOCx4	EOCx5	EOCx6	EOCx7	=
Logical internal File	LFCx5	LFCx7	LFCx10	LFCx13	LFCx15	=
External Enquiry	EECx2	EECx3	EECx4	EECx5	EECx6	=
External inter-face File	EFCx4	EFCx5	EFCx7	EFCx9	EFCx10	=
Total unadjusted function points count (FC)						=

Figure 1.3 Complexity ratings for function point types

The total function points count may be adjusted with compensation factors. The compensation factors used by Albrecht are given in figure 1.4.

To calculate the resulting compensation factor, PCA, one has to use the following formula:

PCA = 0.65 + (0.01 x PC)

For a 'normal' product the resulting compensation factor will be PCA=1.

To get the total adjusted function points, FP, one has to make the following calculation:

FP = FC x PCA

The effort needed to realize a certain project can be expressed in the form:

E = a + b FP ,

characteristic	rating (0 to 5)
data communications	.
distributed functions	.
performance	.
heavily used configuration	.
transaction rate	.
online data entry	.
end user efficiency	.
online update	.
complex processing	.
reusability	.
installation ease	.
operational ease	.
multiple sites	.
facilitate change	.
Total PC	____

ratings: 0: not present, or no influence,
1: insignificant influence,
2: moderate influence,
3: average influence,
4: significant influence,
5: strong influence throughout

Figure 1.4 Compensation factors for function point types

where FP is the total adjusted function points; a, b, c are constants usually derived by regression analysis.

The drawback of counting function points is, that it often is costly and difficult. With this method it is neccessary to have a clear and detailed idea of what the software product is expected to do: this can normally only be achieved after the initial stage of design. To gain the experience needed to make estimates of the expected function points count for a new software product, function points must be repeatedly used for measuring size.

The IBM-FSD model (Walston-Felix)

The equation $\hat{E} = 5.2\ S^{0.91}$ is calculated in the standard way by assuming a relation of the form:

$$E = bS^c$$

Taking logarithms of both sides, we obtain:

$$^e\log E = {}^e\log b + c\,{}^e\log S$$

Setting $Y = {}^e\log E$, $B = {}^e\log b$, and $X = {}^e\log S$, we obtain the linear relationship:

$$Y = B + cX$$

Given a database of projects for which effort (E) in man-months and size (S) in lines of code are available, we can now obtain by the linear least square method the best values of B and c. Finally, we can calculate b using the exponential function.

The IBM-FSD model is not especially good, even when applied to its own database of 60 projects; it suggests that estimators based on size alone cannot be expected to produce good predictions [Conte et al., 86].

In an attempt to explain the observed disparity between predicted and actual effort based on lines of code (the program size), Walston and Felix made a study of factors other than size. Initially, they identified 68 factors that might account for the disparity. A multilinear regression analysis showed that only 29 of these factors were significantly correlated with productivity (is lines_of_code_produced divided by effort in MM). Project managers were than asked to indicate to what extent each of the 29 factors applied to their own project. For each factor, the responses were rated as normal, less than normal, or greater than normal. From these responses, Walston and Felix composed the table given in figure 1.5. This table lists each of the 29 variables, the rating scale used, the mean productivity for each rating, and the productivity change. Although the productivity ranges in figure 1.5 are useful indicators, we must remember that they reflect the projects in a specified environment. The impact of a particular factor may be

substantially different in another environment. When we interpret the given factors, it is important to realize that no attempt was made to factor out the interactions among the different factors; many of the factors are strongly related, thus making it difficult to isolate the effect of any one factor.

	Question or Variable	Response Group Mean Productivity Delivered Lines of code in MM			Productivity Change (PC)
1	Customer interface complexity	< Normal 500	Normal 295	> Normal 124	376
2	User participation in the definition of requirements	None 491	Some 287	Much 205	286
3	Customer originated program design changes	Few 297		Many 196	101
4	Customer experience with the application area of the project	None 318	Some 340	Much 206	112
5	Overall personnel experience and qualifications	Low 132	Average 257	High 410	278
6	Percentage of programmers doing development who participated in design of functional specifications	< 25% 153	25–50% 242	50% 391	238
7	Previous experience with operational computer	Minimal 146	Average 270	Extensive 312	166
8	Previous experience with programming languages	Minimal 122	Average 225	Extensive 385	263
9	Previous experience with application of similar or greater size and complexity	Minimal 146	Average 221	Extensive 410	264
10	Ratio of average staff size to duration (people/month)	< 0.5 305	0.5–0.9 310	> 0.9 173	132
11	Hardware under concurrent development	No 297		Yes 177	120
12	Development computer access, open under special request	0% 226	1–25% 274	> 25% 357	131
13	Development computer access, closed	0–10% 303	11–85% 251	> 85% 170	133
14	Classified security environment for computer and 25% of programs and data	No 289		Yes 156	133
15	Structured programming	0–33% 169	34–66% —	66% 310	141
16	Design and code inspections	0–33% 220	34–66% 300	> 66% 339	119
17	Top-down development	0–33% 196	34–66% 237	> 66% 321	125
18	Chief programmer team usage	0–33% 219	34–66% —	> 66% 408	189
19	Overall complexity of code developed	< Average 314		> Average 185	129
20	Complexity of application processing	< Average 349	Average 345	> Average 168	181
21	Complexity of program flow	< Average 289	Average 299	> Average 209	80
22	Overall constraints on program design	Minimal 293	Average 286	Severe 166	127
23	Program design constraints on main storage	Minimal 391	Average 277	Severe 193	198
24	Program design constraints on timing	Minimal 303	Average 317	Severe 171	132
25	Code for real-time or interactive operation or executing under severe timing constraint	< 10% 279	10–40% 337	> 40% 203	76
26	Percentage of code for delivery	0–80% 159	81–99% 327	100% 265	106
27	Code classified as nonmathematical application and I/O formatting programs	0–33% 188	34–66% 311	67–100% 267	79
28	Number of classes of items in the data base per 1000 lines of code	0–15 334	16–60 243	> 60 193	141
29	Number of pages of delivered documentation per 1000 lines of delivered code	0–32 320	33–88 252	> 88 195	125

Figure 1.5 IBM - FSD Software productivity ranges [Walston-Felix, 77]

Based on the 29 factors of figure 1.5, Walston and Felix define a productivity index I:

$$I = \sum_{i=1}^{29} W_i X_i \quad ,$$

where W_i is a weight defined by:

$$W_i = 0.5 * \log(PC)_i$$

In the latter $(PC)_i$ is the Productivity Change between the low and high rating for the i-th factor in figure 1.5; the rating X is either -1, 0, or +1, depending whether the factor i indicates increased, nominal, or decreased productivity.

Next, project managers were asked to rate the values of the productivity factors for their particular project from which an index I was produced for each project. Using linear regression Walston and Felix found the least squares best fit constants in the equation:

$$\log L = a + bI$$

The actual values found for a and b have not been described in literature. So, one has to calculate these values with the aid of a known (historical) database.

Finally, by using the estimated size S and productivity L, one can obtain an effort estimate from the formula:

$$E = S/L$$

An important result of the Walston-Felix study is the table of figure 1.5, which gives a valuable insight into the factors potentially affecting software development. Finally, one has to realize, that in any single domain, many of these factors will be constant, and thus can be ignored.

The University of Maryland model (Bailey-Basili)

Bailey and Basili were interested in deriving a methodology for effort estimation. Their contention was, that the constants in any effort estimator were very dependent on the environment and personnel, and consequently, that an estimator validated on a local database would more nearly reflect the local environment. They used as a database a set of 18 fairly uniform projects developed in the NASA-Goddard Software Engineering Laboratory. The formula:

$$\hat{E} = 0.73S^{1.16} + 3.5$$

gives the best fit to the NASA-Goddard database. To obtain the constants in their formula, they minimize the 'standard error estimate':

$$\text{standard\_error\_estimate} = \sum_{i=1}^{N} \left[\frac{1 - a + bS_i}{E_i} \right]^2$$

Bailey and Basili obtain a 'standard error estimate' of 1.25. Hence, any estimate obtained from the Bailey and Basili equation must be multiplied by 1.25 and divided by 1.25 to obtain the upper and lower bounds respectively.

Bailey and Basili also suggest a methodology for attempting to account for the differences between their equation and the actual effort. Their methodology for attempts to adjust for the error ratio are defined by:

$$ER_{adj} = \begin{cases} R - 1 & \text{if } R > 1 \\ 1 - R^{-1} & \text{if } R < 1 \end{cases}$$

where $R = E/\hat{E}$, the ratio between actual (E) and predicted ($\hat{E}$) values. After an estimate, we can improve the estimated effort using the formulas:

$$\hat{E}_{adj} = (1 + ER_{adj})\hat{E}, \quad R > 1$$
$$\hat{E}_{adj} = \frac{\hat{E}}{|1 + ER_{adj}|}, \quad R < 1$$

Bailey and Basili use some other project attributes to determine effort multipliers which account for as much as possible of the error in their equation. They combine significant attributes into three categories:

Total METHodology Attribute (9 subattributes). This attribute includes tree charts, top-down design, formal documentation, chief programmer teams, formal test plans, formal training, design formalisms, code reading, and unit development folders.

Cumulative CoMPLeXity Attribute (7 subattributes). This attribute includes customer interface complexity, application complexity, program flow complexity, internal communication complexity, database complexity, external communication complexity, and customer-initiated program design changes.

Cumulative EXPerience Attribute (5 subattributes). This attribute includes programmer qualifications, programmer machine experience, programmer language experience, programmer application experience, and team experience.

For each attribute, a score is obtained by assigning to each subattribute in the group a score of 0 to 5 depending on the extent to which that subattribute was judged by project managers. The METHodology group (METH) had a maximum score of 45 (9 x 5), the cumulative CoMPLeXity group (CMPLX) had a maximum score of 35, and the cumulative EXPerience group (EXP) had a maximum score of 25.

The methodology then consists of using multilinear least squares

regression to find the best fit constants, a, b, c, and d, in the equation:

ER = a x METH + b x CMPLX + c x EXP + d,

where METH, CMPLX, and EXP are estimated by project personnel and where ER is calculated from the prior given equation. If the constants are obtained, we can use the above equation to calculate the error ratio. These ratios can then be used to obtain an adjusted effort, which, in theory, should be closer to the actual effort.

1.2.3. Theoretically-based models

In this section we will discuss two models, those we call in imitation of Conte et al., theoretically-based models. These two models are:

- the SLIM - model [Putnam 78, 79];
- the Jensen model [Jensen 83, 84].

1.2.3.1. The SLIM-model (Putnam)

One of the earliest time-sensitive models was developed by P.V. Norden of IBM [Norden, 63]. He has shown that the Rayleigh distribution gives a good approximation to the labor distributions for many types of research and development activities. A Rayleigh curve (see figure 1.6) is modelled by the differential equation:

$$\dot{y}(t) = 2Kate^{-at^2}$$

where y is the manpower utilization rate, t is the elapsed time, a is a parameter that affects the shape of the curve, and K is the area under the curve in the interval $[0,\infty)$. Integrating the differential equation over the interval [0,t], one obtains:

$$y(t) = K(1 - e^{-at^2}),$$

where y(t) is the cumulative manpower used up to time t. Note, Norden's observations were entirely emperical; there was no evident reason why manpower should exhibit a Rayleigh characteristic as a function of time, it just did. Norden's results were extended by Putnam [Putnam, 78, 79], who showed that the labor distributions for a number of software projects were approximated rather well by the Rayleigh distribution. Putnam has also derived relationships between the software development effort and software development schedules, based on analysis of the Rayleigh distribution. Actually, if one divides the life cycle of a project into phases, each phase can be modelled by a curve of the form given in figure 1.7. The overall life cycle curve for manpower utilization then becomes a composite sum of all the individual phase manpower curves.

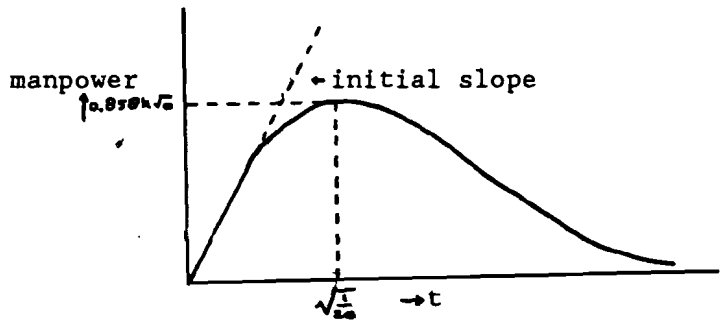


Figure 1.6 Rayleigh distribution

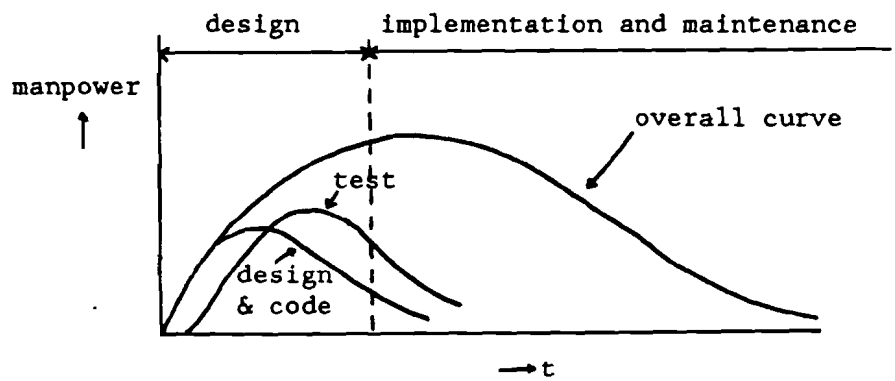


Figure 1.7 Rayleigh curve for software development

Put $a = 1/(2T^2)$, then T is the point at which the manpower utilization rate y is a maximum. Putnam found that the point T on the timescale corresponds very closely to the total project development time. For smaller projects, T can be expected to occur to the right of the peak. If we substitute T for t in the Rayleigh equation, we can obtain an estimate of the development effort:

$E = 0.4\ K$ (E is the cumulative manpower)

In the Putnam theory, the difficulty metric

$D = \frac{K}{T^2}$, where K is the total man-months of effort (area under the Rayleigh curve),

plays an important role in determining software development effort. Putnam assumed that there must be a relationship between D and the productivity of the form:

$L = CD^m$

Previously, L was defined as

$$L = \frac{S}{E} \quad \text{(see section 1.2.2.2)}$$

By using a nonlinear regression analysis applied to several U.S. Army projects, he found, that α must be -2/3 for some constant C. Combining the previous equations, we get

$$S = L\, y(T) = CD^{-\frac{2}{3}}(.3945K)$$

Using $D = K/T^2$, we find that

$$S = C\,[K/T^2]^{-\frac{2}{3}}.K = CK^{\frac{1}{3}}T^{\frac{2}{3}}$$

The constant C is called technology constant. This constant reflects the effect on productivity of numerous factors such as hardware constraints, program complexity, personnel experience levels, and the programming environment. Typical values of C are C = 4984 for 1973-style development and C = 10040 for 1978-style development [Boehm, 81]. Putnam has incorporated his approach to cost estimation into a software product called SLIM (Software LIfe-cycle Methodology; note that SLIM also includes a number of estimation aids). SLIM allows one to calibrate C to a historical project for which S, K, and T are known or to estimate it as a function of modern practices, hardware constraints, personnel experience, interactive development, and other factors. The required development effort is estimated as 40% of the life cycle effort, that is 0.4K. Using the latter equation with S, we obtain:

$$K = T^{-4}[S/C]^3$$

For a software project of a given size and fixed development environment, the equation implies that the effort K varies inversely as the fourth power of the development time. Clearly, one cannot expect this equation to apply over a very wide range. Doubling the development time will probably not allow one to decrease effort by a factor of 16 [DeMarco, 82]. Although it is well-known that compressing development times leads to greater total effort. For example, Boehm estimates that a compression of the development time by 20% to 25%, may increase total effort by a more nominal 23% [Boehm, 81].

Putnam [Putnam, 78] gives the following relationship for the minimum development time of a stand alone software project

$$K/T_{min}^2 = \alpha ,$$

which is essentially the magnitude of the gradient of D. For a system consisting of all new code and high complexity, the value of would be about 7.3, while for a relatively easy system with a large amount of reused code, the value of α would be about 27.

The two equations

$K^{\frac{1}{3}} T_{min}^{\frac{4}{3}} = S/C$, $K/T_{min}^{3} = \alpha$

can be solved for T_{min} and for K, which is designated the minimum time solution (if at least S, C and α are given). Of course stretching out the development time will reduce the effort substantially.

From the foregoing, it appears, there is a limit beyond which a software project cannot reduce its development time by 'buying' more personnel and equipment. Such a limit has been described by Boehm as an "Impossible Region" for development (figure 1.8 [Boehm, 81]).

In figure 1.8 we see, projects that try to squeeze effort into less than circa 1.9 x (effort in MM)$^{\frac{1}{3}}$ months are in the so called "Impossible Region". That does not mean they cannot possibly succeed, only that the emperical evidence is strongly against them. Tom DeMarco [DeMarco, 82] has once said in relation to this:

> I feel that I've spent half my professional life slogging away in the Impossible Region. I wish I've had the evidence of figure 1.8 fifteen years ago.

An alternative curve has been proposed by F.N. Parr of the Imperical College, London [Parr, 80]. This curve predicts essentially the same work profile as the Rayleigh curve, exept during the early part of the project (figure 1.9)

According to Parr the manpower utilization rate at time t is given by $y(t) = 0.25\ \mathrm{sech}^2((at + c)/2)$.

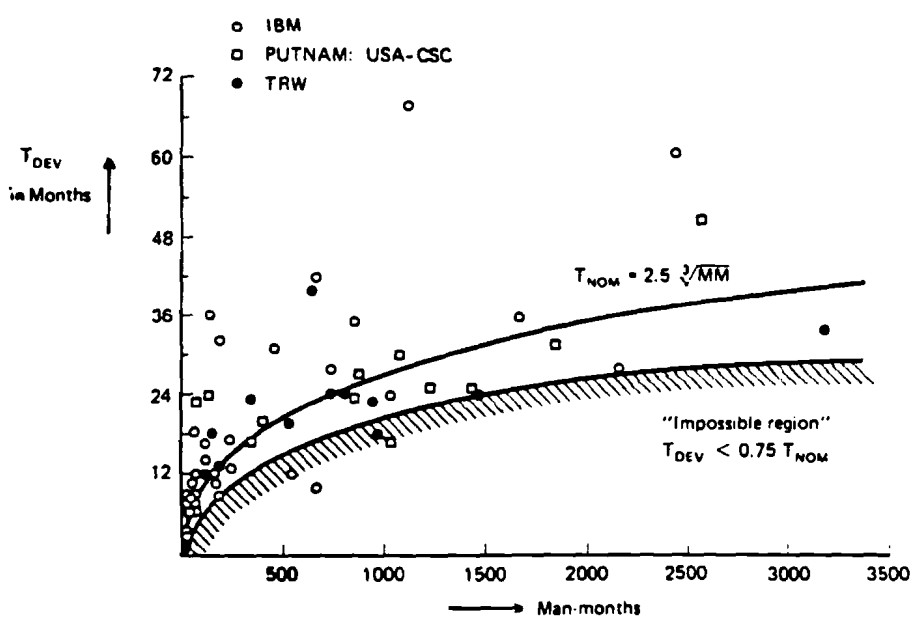


Figure 1.8 The "Impossible Region" [Boehm, 81]

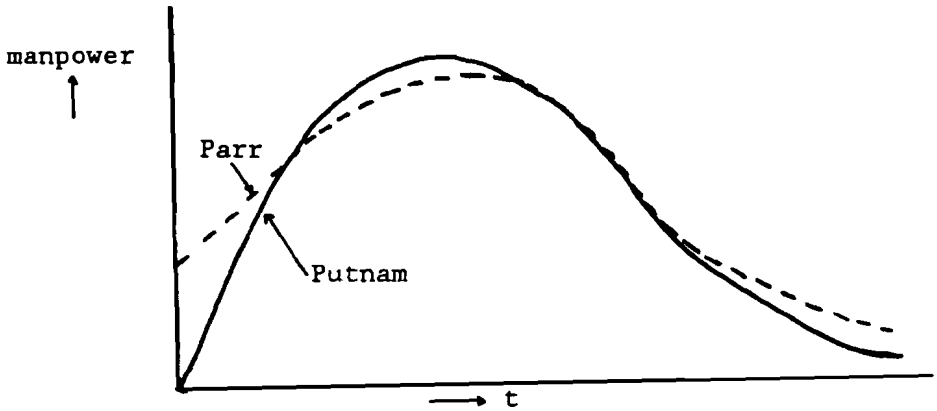


Figure 1.9 Parr and Rayleigh curve compared

For a sample of some two dozen such projects studied in the Yourdon 1978-1980 Project Survey, the Parr curve fits the composite manpower curve quite well [DeMarco, 81].

Note, both Parr and Putnam reflect observed evidence.

Acceleration of a project, in particular a late project, is mostly not possible, even when we stay outside the "Impossible Region". In this context one may think of the following popular law of Brooks [Brooks, 75]:

> Adding manpower to a late software project makes it later.

The argument supporting this law is that, when tasks are partitioned among several programmers, the effort associated with the coordination of the programmers must be taken into account. This additional effort may include programmer training, so that new members may be assimilated into the existing team, and the necessary communication between team members.

1.2.3.2. The Jensen - model

Jensen [Jensen 83, 84] has proposed a model that is very simular to Putnam 's model. His model has the following form:

$$S = C_{et} T K^{\frac{1}{2}},$$

where T is the development time, K is the development effort, S the code size and C_{et} the so called effective technology constant. This constant must be computed by using the following equation:

$$C_{et} = C_{bt} \prod_{i=1}^{n} f_i ,$$

where C_{tt} is the basic technology constant and f_i is a measure of the i-th environmental adjustment factor. These factors are similar to those used by Boehm (see section 1.2.4). They take into account the factors that affect effort.

1.2.4. Composite models

Composite models incorporate a combination of analytic equations, statistical data fitting (linear or nonlinear), and expert judgement to estimate software effort. Putnam's SLIM model, although based on the Rayleigh curve, is in fact a composite model. Another such model is the RCA PRICE S model [Freiman-Park, 79]. PRICE S uses project size, type (platform), and complexity as primary attributes to produce a top-down estimate of the cost of system functions for each phase of the project. The best known and best described of all composite models is COCOMO (COnstructive COst MOdel) [Boehm, 81]. In the next section we will discuss COCOMO and the recently published COPMO (COoperative Programming MOdel).

1.2.4.1. COCOMO

There are three versions of COCOMO, namely the basic model, the intermediate model, and the detailed model. The development effort estimation equations are of the form:

$$E = aS^b.m(X),$$

where S is source lines of code in thousands excluding comments and m(X) is a composite multiplier that depends on 15 cost driver attributes. The principal cost driver in the above equation is program size. Three development modes are identified by Boehm: the organic mode, the semidetached mode, and the embedded mode.

In the organic mode relatively small software teams develop software, needing little innovation, having relaxed delivery requirements, being small in size, in a stable in-house environment. An embedded software project must operate within tight constraints. The end-software product is embedded in a strongly coupled complex of hardware, software, regulations, and operational procedures, such as an air traffic control system. Projects of the semidetached mode fall somewhere in between the organic and embedded modes.

In the basic-model m(X) is 1. This is, thus, a simple model for a rough estimate of effort (in MM) and development time within the software project phases. The effort and time development equations for the organic, semidetached, and embedded modes of software development are as follows:

mode	effort	development time
organic	$MM = 2.4(KDSI)^{1.05}$	$T_{dev} = 2.5(MM)^{0.38}$
semidetached	$MM = 3.0(KDSI)^{1.12}$	$T_{dev} = 2.5(MM)^{0.35}$
embedded	$MM = 3.6(KDSI)^{1.20}$	$T_{dev} = 2.5(MM)^{0.32}$

In these effort equations KDSI is delivered source instructions in thousands.

Figure 1.10 shows the estimated percentage distribution of project effort and development time for the three modes [Boehm, 81]. The intermediate model considers, besides size, 15 cost driver attributes which could effect the effort. These attributes are the following [Boehm, 81]:

Product attributes
- RELY required software RELiabilitY
- DATA DATA base size
- CPLX product ComPLeXity

Computer attributes
- TIME execution TIME constraints
- STOR main STORage constraints
- VIRT VIRTual machine volatility, for a given software product this is the complex of hardware and software (OS, DBMS, etc) it calls upon
- TURN computer TURNaround time

Personnel attributes
- ACAP Analyst CAPability
- AEXP Applications EXPerience
- PCAP Programmer Capability
- VEXP Virtual machine EXPerience
- LEXP programming Language EXPerience

Project attributes
- MODP MODern programming Practices
- TOOL use of software TOOLS
- SCED required development SChEDule

Each of these cost driver attributes determines a multiplying factor which estimates the effect of the attribute on software development (figure 1.11). Of course m(X) = 1, if all cost drivers are given a nominal rating.

On comparing the COCOMO cost drivers with the Walston and Felix factors, we see that the reduction in the number of attributes was achieved partially by combining some attributes that appear highly correlated, and partially by omitting some. Since the COCOMO has fewer attributes, this

Mode	Effort distribution Phase	Size Small 2 KDSI	Inter-mediate 8 KDSI	Medium 32 KDSI	Large 128 KDSI	Very Large 512 KDSI
Organic	Plans and requirements (%)	6	6	6	6	
	Product design	16	16	16	16	
	Programming	68	65	62	59	
	Detailed design	26	25	24	23	
	Code and unit test	42	40	38	36	
	Integration and test	16	19	22	25	
Semidetached	Plans and requirements (%)	7	7	7	7	7
	Product design	17	17	17	17	17
	Programming	64	61	58	55	52
	Detailed design	27	26	25	24	23
	Code and unit test	37	35	33	31	29
	Integration and test	19	22	25	28	31
Embedded	Plans and requirements (%)	8	8	8	8	8
	Product design	18	18	18	18	18
	Programming	60	57	54	51	48
	Detailed design	28	27	26	25	24
	Code and unit test	32	30	28	26	24
	Integration and test	22	25	28	31	34
	Schedule distribution	2 KDSI	8 KDSI	32 KDSI	128 KDSI	512 KDSI
Organic	Plans and requirements (%)	10	11	12	13	
	Product design	19	19	19	19	
	Programming	63	59	55	51	
	Integration and test	18	22	26	30	
Semidetached	Plans and requirements (%)	16	18	20	22	24
	Product design	24	25	26	27	28
	Programming	56	52	48	44	40
	Integration and test	20	23	26	29	32
Embedded	Plans and requirements (%)	24	28	32	36	40
	Product design	30	32	34	36	38
	Programming	48	44	40	36	32
	Integration and test	22	24	26	28	30

Figure 1.10 Phase distribution of effort and time development [Boehm, 81]

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY Required software reliability	.75	.88	1.00	1.15	1.40	
DATA Data base size		.94	1.00	1.08	1.16	
CPLX Product complexity	.70	.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME Execution time constraint			1.00	1.11	1.30	1.66
STOR Main storage constraint			1.00	1.06	1.21	1.56
VIRT Virtual machine volatility[a]		.87	1.00	1.15	1.30	
TURN Computer turnaround time		.87	1.00	1.07	1.15	
Personnel Attributes						
ACAP Analyst capability	1.46	1.19	1.00	.86	.71	
AEXP Applications experience	1.29	1.13	1.00	.91	.82	
PCAP Programmer capability	1.42	1.17	1.00	.86	.70	
VEXP Virtual machine experience[a]	1.21	1.10	1.00	.90		
LEXP Programming language experience	1.14	1.07	1.00	.95		
Project Attributes						
MODP Use of modern programming practices	1.24	1.10	1.00	.91	.82	
TOOL Use of software tools	1.24	1.10	1.00	.91	.83	
SCED Required development schedule	1.23	1.08	1.00	1.04	1.10	

Cost Driver	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
RELY	Effect: slight inconvenience	Low, easily recoverable losses	Moderate, recoverable losses	High financial loss	Risk to human life	
DATA		$\frac{DB\ bytes}{Prog\ DSI} < 10$	$10 \leq \frac{D}{P} < 100$	$100 \leq \frac{D}{P} < 1000$	$\frac{D}{P} \geq 1000$	
CPLX	See Figure 1.12					
Computer attributes						
TIME			≤ 50% use of available execution time	70%	85%	95%
STOR			≤ 50% use of available storage	70%	85%	95%
VIRT		Major change every 12 months Minor 1 month	Major 6 months Minor 2 weeks	Major 2 months Minor 1 week	Major 2 weeks Minor 2 days	
TURN		Interactive	Average turnaround <4 hours	4–12 hours	>12 hours	
Personnel attributes						
ACAP	15th percentile[a]	35th percentile	55th percentile	75th percentile	90th percentile	
AEXP	≤4 months experience	1 year	3 years	6 years	12 years	
PCAP	15th percentile[a]	35th percentile	55th percentile	75th percentile	90th percentile	
VEXP	≤1 month experience	4 months	1 year	3 years		
LEXP	≤1 month experience	4 months	1 year	3 years		
Project attributes						
MODP	No use	Beginning use	Some use	General use	Routine use	
TOOL	Basic microprocessor tools	Basic mini tools	Basic mid/maxi tools	Strong maxi programming, test tools	Add requirements, design, management, documentation tools	
SCED	75% of nominal	85%	100%	130%	160%	

[a] Team rating criteria: analysts (programmers) ability, efficiency, ability to communicate and cooperate

Figure 1.11 Software development effort multipliers and Cost Driver Ratings [Boehm, 81]

Rating	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations
Very low	Straightline code with a few non-nested SP* operators: DOs, CASEs, IFTHENELSEs. Simple predicates	Evaluation of simple expressions: e.g., A = B + C * (D − E)	Simple read, write statements with simple formats	Simple arrays in main memory
Low	Straightforward nesting of SP operators. Mostly simple predicates	Evaluation of moderate-level expressions, e.g., D = SQRT (B**2-4.*A*C)	No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level. No cognizance of overlap	Single file subsetting with no data structure changes, no edits, no intermediate files
Nominal	Mostly simple nesting. Some intermodule control. Decision tables	Use of standard math and statistical routines. Basic matrix/vector operations	I/O processing includes device selection, status checking and error processing	Multi-file input and single file output. Simple structural changes, simple edits
High	Highly nested SP operators with many compound predicates. Queue and stack control. Considerable intermodule control	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, roundoff concerns	Operations at physical I/O level (physical storage address translations; seeks, reads, etc). Optimized I/O overlap	Special purpose subroutines activated by data stream contents. Complex data restructuring at record level
Very high	Reentrant and recursive coding. Fixed-priority interrupt handling	Difficult but structured N.A.: near-singular matrix equations, partial differential equations	Routines for interrupt diagnosis, servicing, masking. Communication line handling	A generalized, parameter-driven file structuring routine. File building, command processing, search optimization
Extra high	Multiple resource scheduling with dynamically changing priorities. Microcode-level control	Difficult and unstructured N.A.: highly accurate analysis of noisy, stochastic data	Device timing-dependent coding, micro-programmed operations	Highly coupled, dynamic relational structures. Natural language data management

* SP = structured programming

Figure 1.12 Complexity Ratings [Boehm, 81]

model should be easier to manage. For the rest, the attributes in the COCOMO appear to be more independent of each other. As we have seen earlier, Walston and Felix use multilinear regression in order to reduce the large number of candidate factors; on the contrary Boehm uses a more heuristic approach combined with a study of other models.

In the intermediate model the effort and time development equations for the organic, semidetached, and embedded modes of software development are as follows:

mode	effort	development time
organic	$MM = 3.2(KDSI)^{1.05}.X(m)$	$T_{dev} = 2.5(MM)^{0.38}$
semidetached	$MM = 3.0(KDSI)^{1.12}.X(m)$	$T_{dev} = 2.5(MM)^{0.35}$
embedded	$MM = 2.8(KDSI)^{1.20}.X(m)$	$T_{dev} = 2.5(MM)^{0.32}$

In intermediate COCOMO, the phase distribution of effort is determined solely by the size of the product. In practice however, factors such as required reliability, applications experience, and interactive software development affect some other phases more than others. That is why, detailed COCOMO provides a set of phase-sensitive effort multipliers for each of the 15 cost drivers. Further, detailed COCOMO provides a three level product hierarchy, for which [Boehm, 81]:

- some effects, which tend to vary with each bottom level module, are treated at the module level;

- some effects, which vary less frequently, are treated at the subsystem level;

- some effects, such as the effect of the total product size, are treated at the system level.

A complete description of the detailed COCOMO is given in [Boehm, 81]. According to Boehm detailed COCOMO is not noticeably better than intermediate COCOMO for estimating overall development effort. However, detailed COCOMO yields better phase distribution estimates.

A number of possible software cost driver attributes have not been included in COCOMO, for instance type of application, language level, management quality, personnel continuity, customer interface quality, amount of documentation, security and privacy restrictions, and requirements volatility (the amount of changes in the software requirements between the beginning and end of a software development project). Some justification for these omissions has given by Boehm in his book [Boehm, 81].

The COCOMO cost driver ratings and nominal effort equations were obtained by a process of calibration. So, it is possible, that one develops a specially calibrated and tailored version of COCOMO which will be more accurate and easy to use within a particular environment. The major actions for calibration and tailoring are:

- calibrating the nominal effort equations to the particular environment;

- consolidating or eliminating redundant cost driver attributes within the model;

- adding further cost driver attributes which may be significant in the new environment.

1.2.4.2. COPMO

COPMO uses an estimate of code size (S) and an estimate of the average personnel level ($\bar{P}$) to predict the effort (E). The general model has the form:

$$E = a_i + b_i S + c_i \bar{P}^{d_i}$$

Using the definition $\bar{P} = E/T$, where T is the project duration in months, and re-arranging the equation:

$$E - c_i E^{d_i}/T^{d_i} = a_i + b_i S$$

$$f(E) = E - c_i E^{d_i}/T^{d_i} - (a_i + b_i S)$$

The subscript i is used to denote an effort complexity class EC_i . The values of a_i , b_i , c_i and d_i can be found by a process of calibration, when a historical database with sufficiently large number of projects falling into class EC_i are known. The determination of complexity classes could be based on various software attributes that are believed to affect productivity; one can use the attributes given in the Walston - Felix model or COCOMO. If one has determined the complexity class, then the model parameters (a_i , b_i , c_i , and d_i) are fixed. Conte et al. have shown, that if $T = T_{min}$, one must have both $f(E) = 0$ and $f'(E) = 0$.

$$f'(E) = 1 - c_i d_i E^{d_i - 1}/ T^{d_i} ,$$

On setting $f'(E) = 0$ and solving for T, we obtain:

$$T = (c_i d_i)^{\frac{1}{d_i}} E^{\frac{d_i - 1}{d_i}}$$

Then substituting into $f(E) = 0$, we get:

$$E - (c_i / c_i d_i)(E^{d_i}/E^{d_i - 1}) = a_i + b_i S$$

$$E(1 - d_i^{-1}) = a_i + b_i S$$

or

$$E(T_{min}) = (a_i + b_i S)/(1 - d_i^{-1})$$

This is, thus, the effort required if the project is completed in time T_{min} . Substituting this effort in the above equation for T, we get

$$T_{min} = (c_i d_i)^{\frac{1}{d_i}} ((a_i + b_i S)/(1 - d_i^{-1}))^{\frac{d_i - 1}{d_i}} \quad \text{or}$$

$$T_{min} = (c_i d_i) ((a_i + b_i S)/(c_i (d_i - 1)))^{\frac{d_i - 1}{d_i}}$$

If one has calculated T_{min}, one has to choose some duration time $T_j > T_{min}$ and

solve f(E) for the corresponding required effort E_j and $\bar{P}_j$ (=E/T).

To solve f(E) one can use Newton iteration method:

$E_{n+1} = E_n - f(E_n)/f'(E_n)$

An initial approximation to E is given by:

$E_o = a_i + b_i S$

Conte et al. have tested the performance of the COPMO model on the COCOMO database. From this it appears that COPMO performs slightly better than Intermediate COCOMO, and clearly demonstrates that it is capable of producing acceptable estimates.

1.2.5. The P938 estimating tool

Within the ESPRIT project P938, entitled "INTEGRATED MANAGEMENT PROCESS WORKBENCH", an estimating tool will be developed that gives a flexible approach to software cost/effort estimating. It allows the projects to adapt existing estimation models to their own needs or to create new ones. Within this estimating tool the Fleximate modelling interpreter plays an important role. To be flexible in its model definition, Fleximate uses command files containing instructions of how to calculate the cost. These instructions would be written in a syntax similar to that used in spreadsheets, but using predefined names for the variables, such as EFFORT and COST. A Fleximate system might include example command files for a few already known estimation models, for instance COCOMO. Fleximate supports the project manager making cost/effort estimates by analogy (see section 1.2.1) as well.

In the near future expert systems may be used within the estimating tool to help with the modification and design of command files. The main support will be for the choice of estimating and modelling techniques, and their implication. The estimating tool could call an expert system to help the estimator to select the most suitable cost estimation model from the range of command files available. P938 has no plans to provide expert system support for the selection of suitable attribute values. Instead an extensive menu-help system will be provided [Cowderoy, 86].

In his book Tom DeMarco has proposed a concept for building cost estimation models [DeMarco, 82]. In this concept 'bangs' play an important role. A 'bang' is a quantitative indicator of a net usable function from the user's point of view; it is implementation independent (within the domain). The 'bang' metric has two major uses [DeMarco, 82]:

> It is used as an early, strong predictor of effort. Bang is the independent variable, that drives the cost estimation model to project development costs.

It is used to calculate useful, productive effectiveness: Bang Per Buck (BPB); this is a metric, that measures the total function delivered by the project per 'dollar' invested from project beginning until the system is retired. Optimization of the project BPB is the first of two quantitative goals that should be established for any development effort.

DeMarco has formulated bangs for function-strong, data-strong and hybrid projects; for the latter he suggests to treate such projects as two projects, one function-strong and the other data-strong. An immediate impression of DeMarco 's bang is, that there is a long list of things to measure. But in practice, most of the metrics can be ignored or simplified.

Obviously, even when we have a method for building (new) cost estimation models, it will not be easy and takes a lot of time to set up such a model and thoroughly test it.

1.3. Some conclusions

When a computer system has to be suitable for helping the projectmanager to manage his software project, a model must be made of the project. A part of building an estimation model is specifying the parameters and their weights that are assumed to be relevant for a specific task and is asserting estimated values to these chosen parameters. In the literature a lot of parameters are known (see for instance [Boehm, 81] and [Walston and Felix, 77]), that influence the effort. These parameters have derived of real projects. So, in first instance, we can use these parameters. If we have built our own historical project database, we can use this to select the parameters which are relevant in the domain we work. Further, this historical database makes it possible to make estimates by analogy.

The system must be alert for pseudo estimates (or better try to avoid them), like Parkinsonian and Price-To-Win estimates. To avoid that a lot of parameter values has to be estimated, one has to classify the several tasks which share the same parameter values. So, one has to choose the overall project parameters and thereupon one has to specify the parameters that are assumed to be relevant for each (sub)class of tasks. (In fact this can be compared with the product hierarchy in detailed COCOMO.) In this way one can reduce the number of estimates. What's more, these classes simplify within a running project the use of historical data derived from that project.

Finally, we have seen that also in this case the literature can give us 'management'-rules. Such an important rule is, that effort and duration are closely related to each other, and that there exists a global limit to acceleration. In his book Boehm has given a formula for this limit [Boehm, 81]. This formula runs as follows:

$T = 1.9 \times (\text{effort in MM})^{.33}$

That does not mean projects that try to squeeze effort into less than this limit cannot possibly succeed, only that the empirical evidence is strongly against them. Another important rule is, that for a lot of software projects, manpower seems to follow a Rayleigh curve as a function of time. If a project is governed by such a curve, the 'progress' as a function of time is given by figure 1.13 (Integral of the Rayleigh distribution function).

Figure 1.13 The integral of the Rayleigh distribution function

Note, that this figure looks like the figure which the University of Amsterdam found for a 'good' project in their knowledge acquisition [Jong et al., 87]. Obviously, one can use these rules with planning, risk analysis, tracking, and diagnosis.

2. Tracking

2.1. Introduction

Tracking comes into the picture during the actual running of the project. In figure 2.1 we see in a simplified scheme where we should place this tool with respect to the other tools of the project management process.

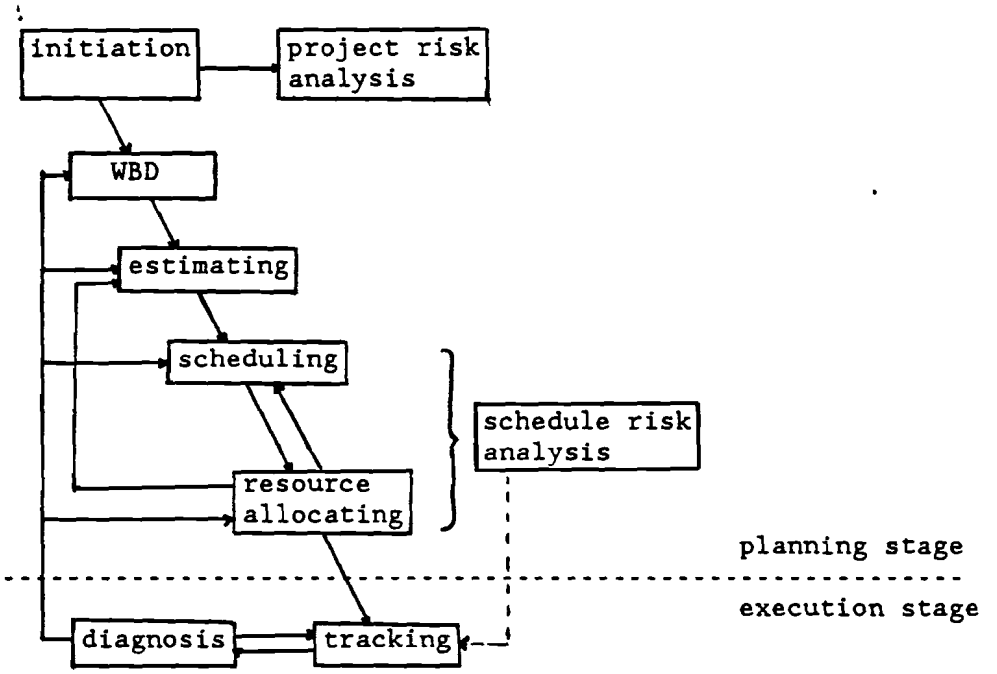


Figure 2.1 Tools of the project management process

The tracker tool indicates to the user and the system the status of the project. The main aim of the tracker is finding out, whether there exist discrepancies between the values of the model-metrics (variables and parameters), attached in the planning stage, and the reality. The input of the tracker consists, thus, of the data gathered from the actual running process and the project plan.

2.2. Measuring objectively

To be able to compare the predicted metric values, we are only interested in metrics that can be obtained more or less objectively. Naturally, it is possible to define metrics that are measured subjectively, such as the quality 'on a scale of 1 to 10' of a software product. But subjective measurements have a lot of disadvantages. These measurements are, by definition, based on individual ideas about what

the metric should be; most probably they will differ with different observers. Further, assigning a value can change as an observer changes over time; that means, given the same situation, an observer may asssign the metric different values today in comparison with, for example, one year from now. Obviously, it is then difficult for the tracker to compare the predicted values with the actual values. Thus, for tracking any observer at any time should assign the same value to the same metric.

It needs no explaining, that purely objective measurements are impossible. In fact, we must try to make the measurements so objectively as possible. A serious problem with this is, there are abstract factors for which there are no good procedures to transform the so called observables into these desired factors. For instance, to measure an abstract concept as software 'quality', one could measure this based on such observables as the number of errors discovered in testing per thousands lines of code, the time between the last two software failures, and so on. It is also possible to create a formula in which these observables are combined. Certainly, it can be argued, that to count the number of errors found in testing is objective. But, the question is, what is the relation between that number and the abstract factor 'quality'? Don't we measure in that case one characteristic of 'quality', namely reliability? Most users for example, do not consider 'error free code' that is not entirely complete, difficult to understand, use and modify, to be of a high 'quality'. Just as you select the characteristics that are assumed to be relevant for the project, the transformations have to be formulated in the planning stage as well.

2.3. Selecting the metrics to track

In the tracker the project manager has to select the project model characteristics (with their metrics and possibly required transformations), such as complexity and quality, which he wants to track for helping to determine the progress and costs. It is, for example, difficult to assess the progress accurately if we do not know what the quality of the delivered work is: slow progress during a design stage may be because the program is larger as we thought or it may be that the designers are producing a particularly good design that wil be easy to code and test.

An expert (knowledge based) system, which supports forward chaining, in view of the needed what-if character, can be used for the selection of the wanted characteristics. This expert system should give advice to the projectmanager concerning which influences the several characteristics possess with respect to the progress. This advice together with a decision support system, for instance based on the so-called Multi Attribute Theory (MAUT), can help the projectmanager to make the right decision with selecting the characteristics. The reason why mostly a selection will be made is to reduce the number of inputs. These inputs must be given to the tracker on a timely basis, probably weekly, so that the tracker is able to compare the predicted values with the actual ones.

One of the most important inputs, which will always be tracked, is the time each team member has spent on a project task. The time spent for other work related to the project, such as meetings, should also be given. Obviously, it is important, that lost time is reported correctly, rather than hidden; and that time spent on unanticipated tasks is reported correctly, rather than allocated to the nearest approximately similar task. In respect of this DeMarco has quoted in his book K.T. Orr [DeMarco, 82]:

> It is not the tasks you planned for that kill you (by costing more or taking longer than you expected); it is the things you never planned for at all.

In a Yourdon project survey [DeMarco, 81] unanticipated, unclassified and miscellaneous activities constituted fully fifteen percent of effort. Apart from that, it appears that, unanticipated other work tends to take up a disproportionate amount of near the end of the project. One way to handle unanticipated work is to build a profile of amount and timing of such work (during making the Work Breakdown Structure), and assume that projects will follow this pattern. When during tracking it appears that this assumption is not right this pattern must be corrected.

It is important to know the quality of the estimations that have been made. This means, we have to calculate a so called Estimate Quality Factor (EQF). A simple EQF can be defined as the reciprocal of the average deviation between estimated value and actual one, or:

$$EQF = \int_{\bullet}^{\tau} \frac{\text{Actual-Value} \times \text{Duration}(T)}{|\text{Estimated\_Value}(t) - \text{Actual\_Value}|\,dt}$$

In his book DeMarco has defined an alternative formula:

$$\text{Time\_Weighed\_EQF} = \frac{.5 \times \text{Actual\_Value} \times \text{Duration}^2}{\int_{\circ}^{\tau} |(\text{Estimated\_Value}(t) - \text{Actual\_Value}) \times (\text{Duration} - t)|\,dt}$$

Exactly which formula one chooses to represent estimating quality does not matter much. What does matter is, that such a formula exists and is consistently applied.

2.4. Discrepancy classes

If the tracker notices a discrepancy between the estimated value and the actual value of a metric, this discrepancy should be classified. The discrepancies can range from 'not serious' to 'very severe'. So, these classes tell whether an action should be undertaken or not [Jong et al., 87]. The main criterion for determining the severity of a discrepancy is the influence of the delay at the finish time of the project. Since the classification into discrepancy classes depends, among other things, on the history of a task, it is clear, that the previous states of the task must be stored. It makes a lot of difference when one notices a discrepancy between the amount of work planned and the amount of work completed, whether there is an improvement compared with the previous monitoring state [Jong et al., 87].

The classifying into discrepancy classes can be done with the aid of an expert system. Analyzing complex resource-constrained task schedules is not easy. (Note tasks possess both resource and deliverable dependencies.) The schedule analysis must be done in the schedule risk analysis tool. Remember with this, the "Impossible Region" (see figure 1.8) found by Boehm [Boehm, 81]. The results of the risk analysis will be needed to determine what the effect of a delay is on related tasks.

2.5. Determining the state of a task

In the scheduling/resource allocating tool a number of tasks is assigned to each of the team members. For each task a number of hours is estimated. Team members may work on several tasks at a time, so during one day or week it is possible time is booked on several tasks. The units of measurement are hours. During the execution of a task it is nearly impossible to determine whether a person is working on schedule or not. In this connection you might think of the so called 90% finished syndrome [Boehm, 81]. This means, that for instance a programmer says in his weekly progress report, that his task is 90% complete, but later it becomes clear, that this estimate was far too optimistic. Hence, it follows that only when a task is finished the projectmanager can determine the exact state of the task. One has to realize with this, that it is very important to have explicit standards for the quality of a deliverable. Without strict standards it is almost impossible to ascertain whether a task is 'finished' or not [Jong et al., 87]. Mostly the project manager is only interested in the tasks (better: milestones) on or near the critical path, because they will lead to a schedule delay. (What does it matter in general that a non-critical task of a project with an overall duration of three years goes on one week longer.) Naturally, the progress of the non-critical tasks must be observed as well. If enough of these fall behind, the lower production rate will eventually have an impact on the critical schedule dates. Obviously, there can be other reasons to observe a non-critical task, if for instance a task requires expensive equipment that has been hired per day, it is also important to observe that task conscientiously. To detect overrun early the tasks may not be too long. Tasks with an estimated duration

of 40 - 80 hours are generally seen as optimal [Jong et al., 87; Tausworthe, 80], anything less being excessively bureaucratic and anything longer insufficiently precise. Because, in practice, it is often not possible to break down all the workpackages into such small tasks, incompleteness (in meaning of level of detail) with respect to the workbreakdown (and thus the schedule) must be allowed until more knowledge is available. Therefore a typical schedule might consists of a detailed plan covering for example three months. As often as needed the project manager has to create a revised plan. (This working method is termed a rolling horizon procedure.)

2.6. Quality Assurance

The tracker tool has to signal when a watch-dog goes off. A watch-dog can be set for project-related events that require action before the set time. The tool has to check whether data, that should be available at the time the tool is activated, is indeed available. If it is not, the project manager is requested to make that data available. If the project manager is not able to render the requested data, he is requested to give the reason. This reason will be logged for Quality Assurance (Q.A.) purposes. Other important points for Quality Assurance can be the following:

- deviations from standards (system, designing, coding, testing, documentation): kind, description, detector_name, detection_date, tasks involved (these must be logged for each deviation);
- recognized new precedence links (incompleteness of the workbreakdown): description, recognizer_name, recognition_date, tasks involved;
- detected defects: detector_name, detection_date, sympton_description, removal date, hours spent, domain of each defect;
- addition of new tasks (incompleteness of the workbreakdown): description, start_date, finish_date, hours_spent;
- required changes in the specification, design or program: kind, description, applicant, request_date, domain of impact (tasks affected), implemented_change_date, hours spent;
- re-planned milestones, tasks or resources: kind, date, indicator;
- recognized new risks: description, recognizer_name, recognition_date, tasks involved.
- requested changes by the user: description, user_name, request_date, domain of impact, implemented _change_date, hours spent;

This information may be of some use in the diagnosis tool as we are going to explain. The diagnosis tries to determine why a certain discrepancy between plan and reality occurred. There are several possible causes for this:

- the value of one or more metric, that was attached during estimating, appears to be incorrect;
- the model, which is described by the metrics, is incorrect (for example the metrics are weighed wrong);
- the chosen metric set is incorrect.

Now, the Q.A.-log can be helpful to determine what the main reason(s) is (are) for a noticed discrepancy. So, this log may lead to new metric values, a new model, or new metrics. Obviously, a decision support system and an expert system, which supports both forward and backward chaining, can be of some use in the diagnosis tool too.

2.7. Schedule, cost, and resource tracking

Progress can be reported in many ways. For a task or work breakdown element (one or more tasks, remember that the top node represents the entire project) that is in progress the following variables can play a part.

Budgeted Work (BW):
The initial budget for a task, workpackage or the entire project in MM.

Work Spent To Date (WSTD):
At any specified point in time the actual effort incurred for the work.

Work Remaining Estimate (WRE):
An estimate of the effort needed to complete the remaining work.

Projected Total Work (PTW):
The sum of the actual effort to date, plus the estimate to complete (PTW = WSTD + WRE).

Budgeted Effort of Work Scheduled (BEWS):
At any specified point in time the percent complete that the item is scheduled to be at that time, times the BW, for that item (BEWS = Item_planned%C_e xBW).

Budgeted Effort of Work Performed (BEWP):
At any specified point in time the actual percent complete, times the BW, for that item (BEWP = Item_actual%C_e xBW, where

Item_actual%C_e is WSTD/PTW).

ProductiVitY (PVTY):

$$BEWP/BEWS = \frac{\text{Item-Actual\%}C_e}{\text{Item\_Planned\%}C_e}$$

Effort Overrun To Date (EOTD):
The difference between the actual effort spent to date and the value of the work that had been planned to be performed, at the measurement time (EOTD = WSTD - BEWS).

Projected Effort Overrun (PEO):
The difference between the projected total work and the budgeted work (PEO = PTW - BW).

Effort Variance (EV):
The difference between the value of the work performed and the actual effort spent for that work (EV = BEWP - WSTD).

Forecasted Effort To Complete (FETC):
A forecast of the effort to be incurred to complete the remaining work. Forecast refers to a computerized extrapolation of the performance to date, based on a built-in algorithm, and estimate refers to a judgmental expression of the effort of the remaining work (see section 3.3).

Forecasted Effort At Completion (FEAC):
The sum of the actual effort to date plus the forecasted effort to complete (FEAC = WSTD+FETC)

A problem here is to attach a value to WRE (in fact this has to be a task of the estimating tool), since any estimate is a mixture of facts and opinions. During the actual running of the project the project manager should modify the estimates to account for errors in the observed estimates. The eventually corrected project model can help the project manager to produce more accurate estimates of effort.

Clearly, for resource tracking the values of the variables mentioned above can be calculated for each task the resource is involved.

A way to look at the cost performance is the so called earned value approach. Earned value is the budget for any item times the percent complete of that item. A protocol that is based on the earned value approach is the Performance Measurement System (PMS) developed by the US Department of Energy. The performance measurement approach is to periodically measure progress and cost in comparable units against the plan.

The following variables are used in PMS:

Budget At Completion (BAC):
The initial budget for a task, workpackage or the entire project

Actual Cost of Work Performed (ACWP):
At any specified point in time the actual cost incurred for the work

Estimate To Complete (ETC):
An estimate of the cost to be incurred to complete the remaining work

Estimate At Completion (EAC):
The sum of the actual cost to date, plus the estimate to complete (EAC = ACWP + ETC)

Budgeted Cost of Work Scheduled (BCWS):
At any specified point in time the percent complete that the item is scheduled to be at that time, times the BAC for that item (BCWS = Item_planned%C$_c$ xBAC)

Budgeted Cost of Work Performed (BCWP):
At any specified point in time, the actual percent complete, times the BAC for that item. This is the earned value of the work performed (BCWP = Item_actual%C$_c$ xBAC, where Item_actual%C$_c$ is ACWP/EAC)

Production Rate (PR):

$$BCWP/BCWS = \frac{Item\_actual\%C_c}{Item\_planned\%C_c}$$

Cost Variance (CV):
The difference between the value of the work performed and the actual cost for that work (CV = BCWP - ACWP)

Schedule Variance (SV):
The difference between the value of the work performed, and the value of the work that had been planned to be performed, at the measurement time (SV = BCWP - BCWS)

Forecast To Complete (FTC):
A forecast of the cost to be incurred to complete the remaining work.

Forecast At Completion (FAC):
The sum of the actual cost to date plus the forecast to complete (FAC = ACWP + FTC)

We have added to this two other variables:

Cost Overrun To Date (COTD):
The difference between the actual cost to date and the value of the

work that had been planned to be performed at the measurement time (COTD = ACWP - BCWS)

Projected Cost Overrun (PCO):
The difference between the projected total cost and the budgeted cost (PCO = EAC - BW)

The resource costs (often the most important costs of a software project) are calculated by multiplying the resource use by the resource rate. A negative aspect with this is, that when the rates change from the plan, the PMS-method loses accurancy.

For the calculating of the variables related to the overall project progress we have two possibilities. Whether we include the tasks that are finished only or we include all the tasks that are started at any time (so also the tasks that are not finished yet).

In the first case we can calculate Work Spent To Date, Work Remaining Estimate, Actual Cost of the Work Performed, and Estimate To Complete in the following way:

Work Spent To Date = $\sum$ Actual Effort Spent belonging to the tasks that have been finished

Work Remaining Estimate = $\sum$ Initial budget of all the tasks in MM that are not finished yet

Actual Cost of Work Performed = $\sum$ Actual Cost belonging to the tasks that have been finished

Estimate To Complete = $\sum$ Initial budget of all the tasks in money that are not finished yet

In the second case we can calculate the value of these variables as follows:

Work Spent To Date = $\sum$ Actual Effort spent belonging to all the tasks that are started

Work Remaining Estimate = $\sum$ Remaining Effort of all the tasks that are started and not finished yet
+
$\sum$ Initial budget of all the tasks in MM that are not started

Actual Cost of Work Performed = Σ Actual Cost belonging to all the tasks that are started

Estimate To Complete = Σ Remaining Cost of all the tasks that are started and not finished yet
$+$
Σ Initial budget of all the tasks that are not started

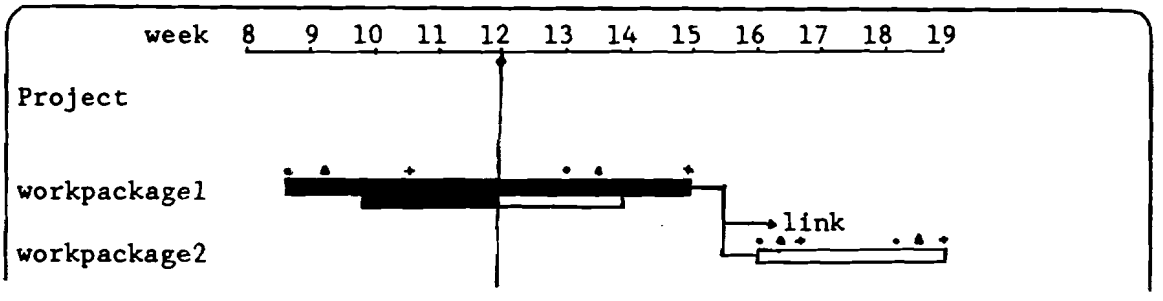
It is clear, that in the tracker a spreadsheet can be useful to make further calculations by the project manager. The report generator will be needed to show the project status in reports. These reports range from graphs (see next section) to tabular ones.

2.8. Graphs

Bar charts (or Gantt charts, so known after their originator Henry Gantt) can be used to show the current schedule against the target schedule. In these bar charts we can make links to show the dependencies between the workpackages. In figure 2.2 we see an example of a linked bar chart. Another important graph is the PercentComplete against time diagram (figure 2.3).

The curves in figure 2.3 enable the manager to compare the actual percentage complete with the planned percentage.

Other graphs that can be useful to determine the status of the project are a manloading histogram, a productivity against time diagram, cost



The activities on the critical path should be high - lighted.

♦actual time
•earliest start/finish date
▲planned start/finish date
+latest start/finish date

Figure 2.2 Instance of a bar chart

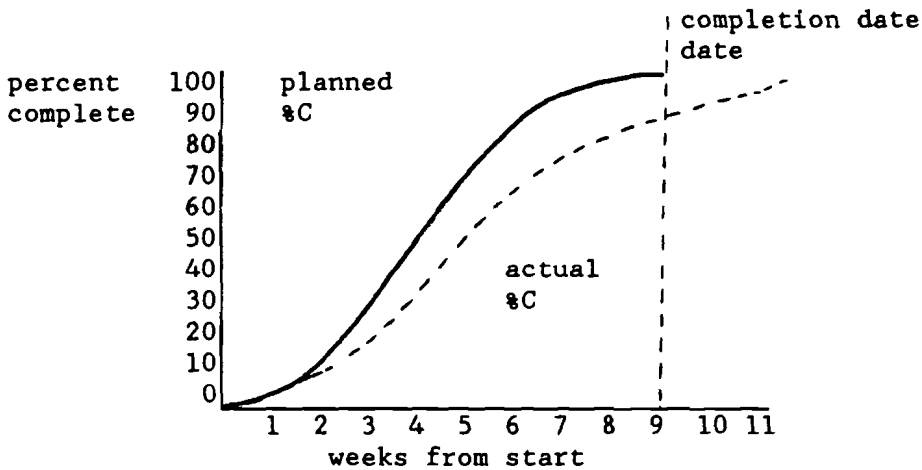


Figure 2.3 Instance of a progress chart

performance diagram (see figure 2.4), and so on.

In the tracker the Work Breakdown Structure (in the form of a tree) can be of some use to determine the (in)completeness of the work breakdown. The activity-on-node diagram, made in the scheduler can be used to show the assumed dependencies between the tasks. These two graphs can also be helpful with selecting the workpackages/tasks the project manager wants to see.

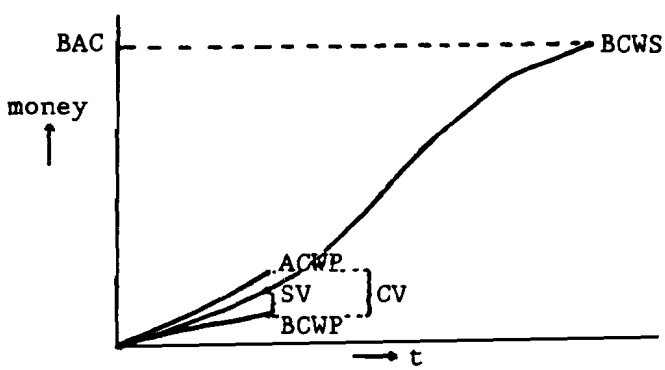


Figure 2.4 Cost performance diagram

3. Predicting

3.1. Introduction

If there is a (serious) discrepancy between the plan and the reality, generally the projectmanager wants to know what the influence of this delay is on the scheduled project completion date. In this case a new prediction of the effort and timescale has to be made. For that we see three possibilities.

The first possibility needs a diagnosis tool. As mentioned in section 2.5, a diagnosis tool tries to determine the cause(s) of a noticed discrepancy. If the cause(s) is (are) found and there is no 'remedy' that does not affect the plan, a replanning process has to start by which one or more of the following things can be needed: completing or re-structuring the workbreakdown, re-estimating the effort and duration of task(s), re-scheduling the tasks, re-allocating the resources. This replanning process produces a plan which appears more realistic. It will be useful, when the planning tools (especially the estimating tool) give the project manager the possibility to calculate the effects of changing the values of one or more model-parameters (like a spreadsheet does). Obviously, if the project manager has some knowledge that in the (near) future something will happen that may have an influence at the plan, he can use the planning tools as well. The foregoing shows, that if there is (or seems to appear) a serious deviation from plan, the planning tools can be used to make a new and hopefully more accurate, (long-term) prediction of the effort and timescale of the project.

The second possibility to make a long term prediction extrapolates the measured values on the basis of the equation (curve), that is assumed to represent the project model. So, in this case we suppose that a project follows the curve that is described by this equation.

The third possibility makes only short-term prediction possible. To predict for the immediate future, forecasting techniques will be used. The reason that those techniques are only useful for short term prediction is, that they are based on the assumption that existing pattern will continue into the future and this assumption is more likely to be correct over the short term than it is over the long term.

In this chapter we will discuss only the latter two possibilities.

3.2. Predicting on the basis of the project model

When the project is running the actual values can be used for (re-)calibrating the equation that represents (a part of) the project model. This calibrated equation can be used then to get an impression of the expected completion date. To calibrate the model equation, $y=f(x)$ least squares techniques may be used.

Suppose, we have m sets of measured y and x and we want to determine the optimal k coefficients c_1, c_2, ...c_k ($c_i \in R^k$) in the equation f, then we have to minimize the following equation:

$$\begin{aligned}\varphi(c) &= \sum_{i=0}^{m}(f_i(\underline{c}) - y_i)^2 \\ &= ||\underline{f}(\underline{c}) - \underline{y}||^2 \\ &= [\underline{f}(\underline{c}) - \underline{y}]^T[\underline{f}(\underline{c}) - \underline{y}] \qquad [3.1]\end{aligned}$$

So, we have to minimize the sum of squares. Assume for example, that a project follows a curve like figure 1.12. The equation of such a curve is given by:

$$E(t) = K(1 - e^{-at^2})$$

If E(t) is measured in t_1, t_2, ...t_m and we have to find the coefficients a and K, we have to minimize the function:

$$\varphi(a,K) = \sum_{j=1}^{m}(K(1 - e^{-at_j^2}) - E(t_j))^2$$

To minimize the sum of squares given in [3.1], we have to find the gradient F and the Hessian G of the function φ:

$$F_i(c) = \frac{\partial \varphi}{\partial c}(\underline{c}) = 2\frac{\partial f}{\partial c_i}(\underline{c})\,[\underline{f}(\underline{c}) - \underline{y}] \qquad \text{(a K - vector)} \quad [3.2]$$

$$G_{ij}(c) = \frac{\partial^2 \varphi}{\partial c_i \partial c_j}(c) = 2\frac{\partial \underline{f}^T}{\partial c_i}(\underline{c})\frac{\partial \underline{f}}{\partial c_j}(\underline{c}) + 2\frac{\partial^2 \underline{f}^T}{\partial c_i \partial c_j}(\underline{c})[\underline{f}(\underline{c}) - \underline{y}] \qquad [3.3]$$

If $\underline{F}(\hat{\underline{c}}) = 0$ en $G(\hat{\underline{c}})$ is definite positive, then $\hat{\underline{c}}$ is a relative minimum of φ.

The method of Newton to determine a zero point of $\underline{F}(\underline{c}) = \underline{0}$ is as follows.

If $\underline{c}_{n-1}$ is the last fixed approximation of $\hat{\underline{c}}$, $\underline{d}_n$ can be computed from:

$$G(\underline{c}_{n-1})\underline{d}_n = -\underline{F}(\underline{c}_{n-1}) \qquad [3.4]$$

As next approximation we can take:

$$\underline{c}_n = \underline{c}_{n-1} + \underline{d}_n$$

As stop criterion can be chosen:

$||\underline{c}_n - \underline{c}_{n-1}|| \leq \varepsilon$,where ε must be small

When the project equation is linear, for instance

$y = c_1 x + c_2$ (eventually after a simple transformation),

it is quite simple to minimize equation [3.1]. We will show this.

$\varphi(c_1, c_2) = \sum (c_1 x_i + c_2 - y_i)^2$, where $\sum$ means $\sum_{i=1}^{m}$

To minimize we have to solve the following equation:

$$\frac{\partial \varphi(c)}{\partial c_1} = \sum 2x_i (c_1 x_i + c_2 - y_i) = 0$$

$$\frac{\partial \varphi(c)}{\partial c_2} = \sum 2(c_1 x_i + c_2 - y_i) = 0$$

After converting we find:

$$c_1 \sum x_i^2 + c_2 \sum x_i = \sum x_i y_i$$

$$c_1 \sum x_i + c_2 M = y_i$$

So that

$$c_1 = \frac{(\sum x_i y_i) M - \sum y_i \sum x_i}{(\sum x_i^2) M - (\sum x_i)^2} \quad \text{and} \quad c_2 = \frac{\sum x_i^2 \sum y_i - \sum x_i^2 \sum x_i y_i}{(\sum x_i^2) M - (\sum x_i)^2}$$

Clearly, $(\sum x_i^2) M - (\sum x_i)^2$ must be unequal to zero.

There exists several relationships that are intrinsically linear: even though they are not linear, linear regression can be used to obtain the appropiate coefficients.

Take, for instance, the following equation:

$y = c_1 x^{c_2}$ (For example, the effort estimator found by Boehm [Boehm, 81] has this form)

This power equation can be linearized by taking the natural logarithm of both sides to yield:

$$^{e}\log y = {}^{e}\log c_1 + c_2 {}^{e}\log x$$

If we set

$Y = {}^{e}\log y$, $C1 = {}^{e}\log c_1$, $X = {}^{e}\log x$,

we obtain the linear form:

$Y = C1 + c_2 X$

After we have obtained the best fit constants C1 and c we can return to the power form by taking the inverse logarithm of C1:

$c_1 = \exp(C1)$

When the obtained coefficients of the different calibrations fluctuate heavily, we may assume the model is not correct.

3.3. Forecasting

We focus in this section on the method of exponential smoothing which is a simple but powerful approach to short-term forecasting. The so called simple exponential smoothing technique uses the following formula:

$A_t = \alpha d_t + (1 - \alpha) A_{t-1}$, where $0 \leqslant \alpha \leqslant 1$ [3.4]

In this equation A_{t-1} is the old average, d_t the actual observation in period t and α the so called smoothing constant. So, A_t is a weighed average of the actual observation in period t and A_{t-1} the old average.

The forecast for all future periods is equal to the forecast for the next period:

$F_{t+k} = A_t \quad k = 1, 2, 3, \ldots$ [3.5]

Equation [3.4] can be rewritten as:

$A_t = A_{t-1} + \alpha (d_t - A_{t-1})$ [3.6]

Since $F_t = A_{t-1}$, the term $(d_t - A_{t-1})$ represents the forecast error in period t; it is the difference between what actually occurred and what had been forecasted one period previously. The choice of α, the smoothing constant, indicates how much the new forecast has to be influenced by the size of the error. If we substitute in equation [3.6] A_{t-1} and A_t by F_t and F_{t+1} respectively, this equation runs as follows:

$F_{t+1} = F_t + \alpha (d_t - F_t)$ [3.7]

The equation [3.4] can be expanded as follows:

$A_t = \alpha d_t + (1 - \alpha) A_{t-1}$

So $A_{t-1} = \alpha d_{t-1} + (1 - \alpha) A_{t-2}$

so that $A_t = \alpha d_t + (1 - \alpha)[\alpha d_{t-1} + (1 - \alpha) A_{t-2}]$

or $A_t = \alpha d_t + \alpha (1 - \alpha) d_{t-1} + (1 - \alpha)^2 A_{t-2}$

Clearly, $A_{t-2} = \alpha d_{t-2} + (1 - \alpha) A_{t-3}$

so that $A_t = \alpha d_t + \alpha (1-\alpha) d_{t-1} + \alpha (1-\alpha)^2 d_{t-2} + (1-\alpha)^3 A_{t-3}$

Finally, we can obtain:

$$F_{t+k} = A_t = \alpha d_t + \alpha(1-\alpha)d_{t-1} + \alpha(1-\alpha)^2 d_{t-2} + \ldots + \alpha(1-\alpha)^x d_{t-x} + \ldots \quad [3.8]$$

So, A_t is an average of all previous observations.

A way to select α is to take the one that produces the most accurate forecasts. However, one should be cautious about using an equation with an α higher than about 0.5. If the most accurate forecast is for an α greater than this value, it is often the case a better model exists.

If the historical data have a trend in it, a simple exponential smoothing will not provide good forecasts. As long as this trend lasts, the forecast will lag the actual. By accounting for the trend pattern in the data, better forecasts can be produced. A method, that does so, is the Holt's method (Holt et al., 60). This method uses the following equations:

$$L_t = \alpha d_t + (1-\alpha)(L_{t-1} + T_{t-1}) \qquad 0 \leq \alpha \leq 1 \qquad [3.9]$$

$$\text{and } T_t = \beta (L_t - L_{t-1}) + (1-\beta)T_{t-1} \qquad 0 \leq \beta \leq 1 \qquad [3.10]$$

Where α is the smoothing constant used for the base level, and β is the smoothing constant used for the trend.

To get started the model requires initial estimates L_0 and T_0. These estimates can be based either on judgement or on past data. The forecast made at the end of period t for period t+k, k periods in the future, is therefore:

$$F_{t+k} = L_t + (T_t)k$$

In this equation we add one unit of trend for each future period we consider. A special case of the Holt method uses the same smoothing constant for smoothing both the level and the trend. The advantage of this is, that there is only a single constant to worry about. The disadvantage is, that the method may not produce forecasts that are as accurate as those produced by Holt's method, which uses two smoothing constants.

Other smoothing methods use nonlinear trend models for short-term forecasting. These may not do any better than Holt's method, because over short horizons a linear trend may be a good approximation of any trend. However, more complicated trend models may be useful for forecasting over longer horizons.

The quality of the forecasts/estimates should be evaluated how well they perform in the application for which the forecasts/estimates are used. For this purpose we can use a formula as given in section 2.3. However, we will give here a more statistical approach.

To see how the actual value of d and F relate to each other, we define the relative error in period t by:

$$RE_t = \frac{F_t - d_t}{d_t}$$

If $F_t > d_t$, then RE_t is negative and if $F_t < d_t$, then RE_t is positive. When $RE_t > 0$, it must be between 0 and 1, but when $RE_t < 0$, it is essentially unbounded in magnitude. We can also define the mean relative error of a set of n forecasts/estimates by the formula:

$$\overline{RE} = 1/n \sum_{i=1}^{n} RE_i$$

If the estimates/forecasts are good, they will lead to small values of RE and generally to a small $\overline{RE}$. However, since it is possible, that large positive RE_t's can be balanced by large negative RE_t's, a small $\overline{RE}$ may not imply that the estimates/forecasts are good ones.

In view of this problem with RE_t and $\overline{RE}$ we define the magnitude of the relative error as:

$$MRE_t = |RE_t| = \left| \frac{F_t - d_t}{d_t} \right|$$

Thus, the smaller the value of MRE_t, the better the prediction. Now, positive and negative relative errors do not balance out. For a set of n forecasts/estimates, we can calculate the mean magnitude of the relative error by:

$$\overline{MRE} = 1/n \sum_{i=1}^{n} MRE_i$$

If $\overline{MRE}$ is small, the forecasts/estimates are good predictions. However, even when $\overline{MRE}$ is small, there may be one or more predictions that can be very bad. We consider $\overline{MRE} < .25$ as acceptable.

If k is the number of forecasts/estimates in a set of n forecasts estimates whose $MRE_i \leq x$, then we define a measure as:

$$PRED(x) = k/n$$

For example, if PRED(.25)=.83, then 83% of the predicted values fall within 25% of their actual values. An acceptable criterion seems to be $PRED(.25) \geq .75$.

4. Glossary

The terms in this document are generally those of the Glossary of Software Engineering Terminology (ANSI/IEEE Std 729-1983) [IEEE, 84]. Extensions to this list include the following:

activity
: Individual item of work which along with other items of work comprises the project.

activity-on-node diagram
: A network diagram, where each node represents an activity and each arc shows logical precedence. This approach is most commonly used with the critical path method (CPM).

availability
: The time interval in which a resource can be used.

consumable resource
: A resource that can be used only once, e.g. processor time; thus a resource that is not re-usable (durable).

critical activities
: The activities along the critical path. A delay in any one of them will cause a delay in the completion of the entire project.

critical path
: The longest path in a network. It determines the project length in the sense that precedence relations permit no earlier completion of the project.

deliverable
: The final object of a task.

dependency
: The logical relationship between activities within a project due to time or causality or both.

durable resource
: A resource that can be used more than once. Contrast with consumable resource.

early finish time
: The earliest time at which the task can be completed.

early start time
: The earliest time at which the task can begin.

estimate
: A judgement that is equally likely to be above or below the actual result.

event
: The fact of a thing's happening, where the thing is not directly influenced by the project, like the delivery of a machine.

forecast
: An estimate of something in the near future based on a computerized extrapolation of the past.

free slack
: The allowable delay in a task start time that can be absorbed without delaying another task.

late finish time
: The latest time at which the task can finish without delaying the project.

late start time
: The latest time at which the task can begin without delaying the project.

milestones
: A point during the life cycle of a project at which actual progress is monitored against planned progress.

model
: A miniature representation of a complex reality.

monitoring the work progress
: The means by which feedback on actual progress against planned progress is provided to the users to inform them of the budgetary and technical development of the project. Monitoring can be undertaken at random intervals or at pre-determined points.

network diagram
: This is a graphical model for analysis of the scheduling problem. All diagrams use nodes and arcs.

path
: This is a sequence of connected activities in a network diagram, following a set of precedence arrows from the start of a project to its completion.

prediction
: A long term estimate.

project manager
: The individual who is responsible for the successful running and completion of the project.

project metric
: A quantify attribute of the project.

resource allocation
: The apportioning of the available and required resources to the various activities which form the project.

Software project
: A series of activities whose major deliverable is an item of software.

standards
: Prescribed rules that have to be followed during the development of a software product.

task
: Activity within the project (see activity)

total slack
: The allowable delay in a task start time that can be absorbed without delaying the project, assuming that all other tasks begin as early as possible.

tracking the work progress
: The recording and monitoring of how much work has been done, what resources were utilized, and what costs were incurred. At any update or measurement point, which should occur at regular intervals or at pre-determined points (milestones). It also entails keeping track of any changes in the project plan.

user
: The person or persons who operate or interact directly with the system.

Work Breakdown
: The organizing of the project activity elements into a hierarchical structure. This structure is composed of two hierarchies, which can be connected in whichever way best fits the project. These two hierarchies are:

    -the product hierarchy;
    -the activity hierarchy.

5. Bibliography

[Albrecht, 79]
Albrecht, A.J., "Measuring application development productivity", Proceedings of the Joint SHARE/GUIDE Symposion, October 1979, pp 83-92

[Albrecht, 83]
Albrecht, A.J., and Gaffney Jr., J.E., "Software function, source lines of code, and development effort prediction: a Software Science validation.", IEEE Transactions on Software engineering SE-9, 6, November 1983, pp 639-647

[Bailey-Basili, 81]
Bailey, J.W., and Basili, V.R., "A meta-model for software development resource expenditures", Proceedings of the Fifth International Conference on Software engineering, IEEE/ACM/NBS, March 1981, pp 107-116

[Boehm, 81]
Boehm, B.W., "Software engineering economics", Prentice Hall, 1981

[Conte et al., 86]
Conte, S.D., Dunsmore, H.E., and Shen, V.Y., "Software engineering metrics and models", The Benjamin/Cummings Publishing Company, 1986

[Cowderoy, 86]
Cowderoy, A., "The Fleximate estimation tool", Imperial College, IMPC-WP5-W-003(3), 1986

[DeMarco, 81]
DeMarco, T., "Yourdon 1979-80 project survey final report", Yourdon inc., September 1981

[DeMarco, 82]
DeMarco, T., "Controlling software projects", Yourdon Press, 1982

[Drummond, 85]
Drummond, S., "Measuring applications development performance", Datamation, February 1985

[Freiman-Park, 79]
Freiman, F.R., and Park, R.E., "PRICE software model-version 3 an overview", Proceedings, IEEE-PINY workshop on quantitative Software models, IEEE Catalog No. TH0067-9, October 1979, pp 32-41

[Holt et al., 60]
Holt, C.C., et al., "Planning production, inventories and work force", Prentice Hall, 1960

[IEEE, 84]
IEEE, "Software engineering standards", IEEE Inc, 1984

[Jensen, 83]
Jensen, R.W., "An improved macrolevel software development resource estimation model", Proceedings Fifth ISPA Conference, April 1983, pp 88-92

[Jensen, 83]
Jensen, R.W., and Lucas S., "Sensitivity analysis of the Jensen software model", Proceedings Fifth ISPA Conference, April 1983, pp 384-389

[Jensen, 84]
Jensen, R.W., "A comparison of the Jensen and COCOMO schedule and cost estimation models", Proceedings of the Int. Soc. Parametric Analysis, 1984, pp 96-106

[Jong et al., 87]
Jong, de T., Hoog, de R., and Schreiber, G., "Monitoring, Diagnosis, and Forward Tracking in Software Project Management", University of Amsterdam, UVA-D-RR-oo5a, 1987

[Nelson, 66]
Nelson, E.A., "Management handbook for the estimation of computer programming costs.", Systems Development Corporation, 31 october 1966

[Norden, 63]
Norden, P.V., "Useful tools for project management", Operations Research in Research and Development, John Wiley & Sons, 1963

[Parkinson, 57]
Parkinson, G.N., "Parkinson's Law and other studies in Administration", Houghton-Miffin, 1957

[Putnam, 78]
Putnam, L.H., "A general emperical solution to the macro software sizing and estimating problem", IEEE Transactions on software engineering SE-4, 4, July 1978, pp 345-361

[Putnam, 79]
Putnam, L.H., and Fitzsimmons, A., "Estimating software costs", Datamation, September 1979, pp 189-198, Continued in Datamation, October 1979, pp 171-178, and November 1979, pp 137-140

[Tausworthe, 80]
Tausworthe, R.C., "The work breakdown structure in software project management", The Journal of System and software, Vol. 1, 1980, pp 181-186

[Walston-Felix, 77]
Walston, C.E., and Felix, C.P., "A method of programming measurement and estimation.", IBM System Journal, Vol. 16, No.1, January 1977, pp 54-73

[Wolverton, 74]
Wolverton, R.W., "The cost of developing large-scale software", IEEE Trans. Computers, June 1974, pp 615-636

Specifications of tracking in software project management

Synopsis

This document describes the specifications of the tracking tool.

Authors : RF Wissing, BSO/Eindhoven
Reader :
Level : BSO/Eindhoven BV
Status : Draft
Identifier: BSO-G-SP-104.n (Release : 1.14)
Created : 19 october 1987 (Edited : 88/02/12)
Printed : 12 February 1988

This document is part of a project partially funded by the Commission for the European Communities ESPRIT programme, as project number 814.

© 1988 BSO/Eindhoven BV

Distribution List

Christer Fernstrom	CSI
John Hawgood	PA
Robert de Hoog	UvA
Patrick Humphreys	LSE
John Jenkins	938
Hans van de Klok	BSO
Frank Land	LBS
Eirik Naess-Ulseth	SI

Revision Information

Creation Date:
Author :
Reader :
Modifications:

Table Of Content

1. Introduction Tracker Tool Specifications

This chapter discusses the specifications for the tracker. The tracker is one of the PIMS-tools. It provides the means to feed the user's actual knowledge of the running project into the PIMS system and it indicates to the user the status of the project. This has been described extensively in document BSO-G-RR-102 [Wissing, 87].

It is (often) effective to specify the required behavior of a software product by a description of the inputs, outputs and processing (transforming the inputs into the outputs). Hence, the tracker tool requirements will be expressed mainly through a detailed description of these. The specifications will be split into two parts:

- PIMS #1 tracker tool specification;
- future extensions.

Since in PIMS #1 the tracker and the predictor will be integrated, the latter will be described in this document too. Only short term prediction will be possible in PIMS#1. To make a clear distinction between predicting and tracking, they will be discussed in separate sections.

2. PIMS #1 Tracker Tool Specifications

2.1. Tracker Part

2.1.1. Inputs

In the first instance we divide the inputs into two groups:

- Inputs required from the other tools and the system;
- User inputs.

2.1.1.1. Inputs required from the other tools and the system

To be able to determine the project status the tool requires the following inputs of the other tools via the information manager:

- System input:
    - Actual Date

- Session:
    - Current Project
    - Current Project View
    - Current Project Status

- Project:
    - Key

- ProjectView:
    - external Milestones

- ProjectStatus:
    - Workbreakdown
    - Schedule:
        - links
        - tasks scheduled

- Workpackage data:
    - Father
    - Sons
    - Key
    - Needs (a list of deliverables from other workpackages needed by the workpackage)
    - Produces (a list of deliverables produced by the workpackage)
    - Realisation (The task that implements the workpackage. We suppose that only the workpackages without sons have a task as realisation. With the other workpackages their sons represent their

realisation.)

- Task data:
    - Name
    - Description
    - Workpackage
    - ProjectStatus
    - AllocatedResources
    - Status
    - Schedule Dates: Early Start, Early Finish, Late Start, Late Finish, Planned Start, and Planned Finish Date
    - EstimatedEffort (initially estimated effort, BW)
    - EstimatedCost (initially estimated cost, BAC)

- Human Resource data:
    - Person:
        - Key
    - Cost by Hour
    - Reservations

- Allocation (Reservation) data:
    - Resource
    - list of:
        - planned start date
        - planned end date
        - percent of usage of the resource
    - Task

- Milestone data:
    - Key
    - NotLaterThan
    - Deliverables

- Deliverable:
    - ProducedBy (workpackage)

2.1.1.2. User inputs

The user may give the tool the following kinds of inputs:

- Overall project data:
    - External Milestones acknowledged date (The date a milestone has been formally achieved. An external milestone has been informally achieved when all its deliverables have been acknowledged.)

- Task data:

- Task Finish Date
- Task Acknowledged Date
- Deliverable Finish Date
- Deliverable Acknowledged Date
  (Internally a deliverable is the document that synchronizes the different tasks. So, it is possible a deliverable has been delivered, but the task is not finished yet. For instance, one has done a literature study and has finished the document belonging to it, but one has to bring the books back to the library.)

- Task/HumanResource data:
  - Measurement date
  - Effort Spent per task, per resource
  - Estimate of the effort to complete the work by resource, by task (Remaining Effort)
  - Actual Start Date of a resource on a task
  - Actual End Date of a resource on a task

- User query:
  - Graphs (see section 2.1.2.2)
  - Project/Task Control tabular Reports
  - Resource Control tabular Reports
  - data that can be useful for entering data (see section 2.1.2.2), a list of names of not_acknowledged_milestones for instance, can be useful with selecting these milestones

2.1.2. Outputs

We divide the outputs, just like the inputs, into two groups:

- outputs to the system;
- user outputs.

2.1.2.1. Outputs to the system

The tool will give the system the following kinds of outputs:

- Task data:
  - Status (A task has the status NotStarted as long as no effort has been carried to the task. When effort is spent on the task the status goes to InProgress. In the tracker the user can change the status in Finished (on the condition the deliverable has been finished) The Status goes back from Finished to Progress when effort is again spent on the task. If the deliverable is Finished the user can

change the status from Finished to Acknowledged. The Acknowledged status means that completion of the task is formally acknowledged, so the work is absolutely finished.
    - Start Date
    - Finish Date
    - FinishCount
      (For Q.A. reasons (note, a Q.A. tool is not available in PIMS #1) the system has to know how many times one thinks the task is ready, i.e. how many times its status has changed from InProgress to Finished.)
    - Acknowledged Date
    - Estimated Effort (BW)
    - Effort Spent (WSTD)
    - Remaining Effort (WRE)
    - Cost Spent (ACWP)
    - Remaining Cost (ETC)
    - Measurement Date (This date may be different from the dates the data have been entered)

- Human Resource data:
    - Actual Start Date of a resource on a task
    - Actual End Date of a resource on a task
    - Effort Spent (WSTD) by task
    - Remaining Effort (WRE) by task
    - Measurement Date

- Deliverable:
    - Finish Date
    - Acknowledged Date

Note, most of these outputs will also be required by the tracker as well.

2.1.2.2. User outputs

The tracker tool indicates to the user the status of the project. The user can get the following kinds of outputs:

- Tabular reports:
    - Overall project:
        - Time/Date, Project_Name
        - Effort/Cost Control Report: see Project Control Report
          (The top-node-workpackage of the WBS represent the total project)

    - Milestone:
        - Not Later Than Date
        - Acknowledged Date
        - Deliverables (needed to achieve the milestone) with

their tasks
- Acknowledged Date of the deliverables (a milestone has been informally achieved when all its deliverables have been acknowledged)

- Project Control:
  - Effort Control Report: BW, WSTD, WRE, PTW, FETC (see section 2.2), BEWP (earned value), PVTY, PEO, EV by workpackage by task or summarized (The abbreviations are explained in Chapter 5.)
  - Cost Control Report: BAC, ACWP, ETC, EAC, FAC (see section 2.2), BCWP, PR, CV, PCO by workpackage by task or summarized

- Task Control:
  - Effort Control Report: BW, WSTD, WRE, PTW, FETC (see section 2.2), BEWP (earned value), PVTY, PEO, EV by task by resource or summarized (The abbreviations are explained in Chapter 5.)
  - Cost Control Report: BAC, ACWP, ETC, EAC, FAC (see section 2.2), BCWP, PR, CV, PCO by task by resource or summarized

- Task/Deliverable Dates
  - Early Start, Early Finish, Late Start, Late Finish, Planned Start, Planned Finish, Actual Start, Task Actual Finish, Forecasted Finish (see section 2.2) and Task Acknowledged Date
  - Deliverable Finish and Acknowledged Date

- Human Resource Control:
  - Reserved Time Interval by resource by task
  - Actual Start Date of a resource on a task
  - Actual End Date of a resource on a task
  - Effort Control Report: BW, WSTD, WRE, PTW, FETC (see section 2.2), BEWP (earned value), PVTY, PEO, EV by resource by task or summarized
  - Cost Control Report: BAC, ACWP, ETC, EAC, FAC (see section 2.2), BCWP, PR, CV, PCO by resource by task or summarized

- Lists:
  - Used Resources (resources which are allocated to one or more tasks)
  - Tasks where a resource has been allocated to
  - Milestones Not Acknowledged
  - Started but Not Acknowledged Tasks
  - Not Acknowledged Tasks (project to do list)

- Graphs:
  - Overall project:
    - Work Breakdown Structure (helpful with selecting the

workpackages the PIMS user wants to see)
- Activity-On-Node diagram (helpful with selecting the tasks the PIMS user wants to see)

- Task/Workpackage:
  - Gantt chart of tasks

- Resource:
  - Gantt chart of resources

2.1.3. Tools required by the tracker

The tracker requires the following tools:

- Information Manager for providing the access to the objects that are read and written by the tracker [Fernstrom et al.,87a]
- Front End tool for handling communications with the user [Fernstrom et al., 87b]

2.1.4. Processing

The processing in the tool will be described with the help of the commands the tool offers. These commands can be grouped into five command groups:

- Enter Project Data
- Enter Task Data
- Enter Resource Data
- Control Reports
- Graphs

When a tool is activated these commands will be shown in the so-called Dialogue subwindow. Further, two static graphical subwindows containing an Activity-On-Node network and a Work Breakdown Tree respectively, and a list subwindow with the used resources will be created. (A resource is called used when he has been allocated to one or more tasks, which belong to the current project status.) This list window will have the header "USED RESOURCES" (figure 1) and the resources will be shown in alphabetical order.

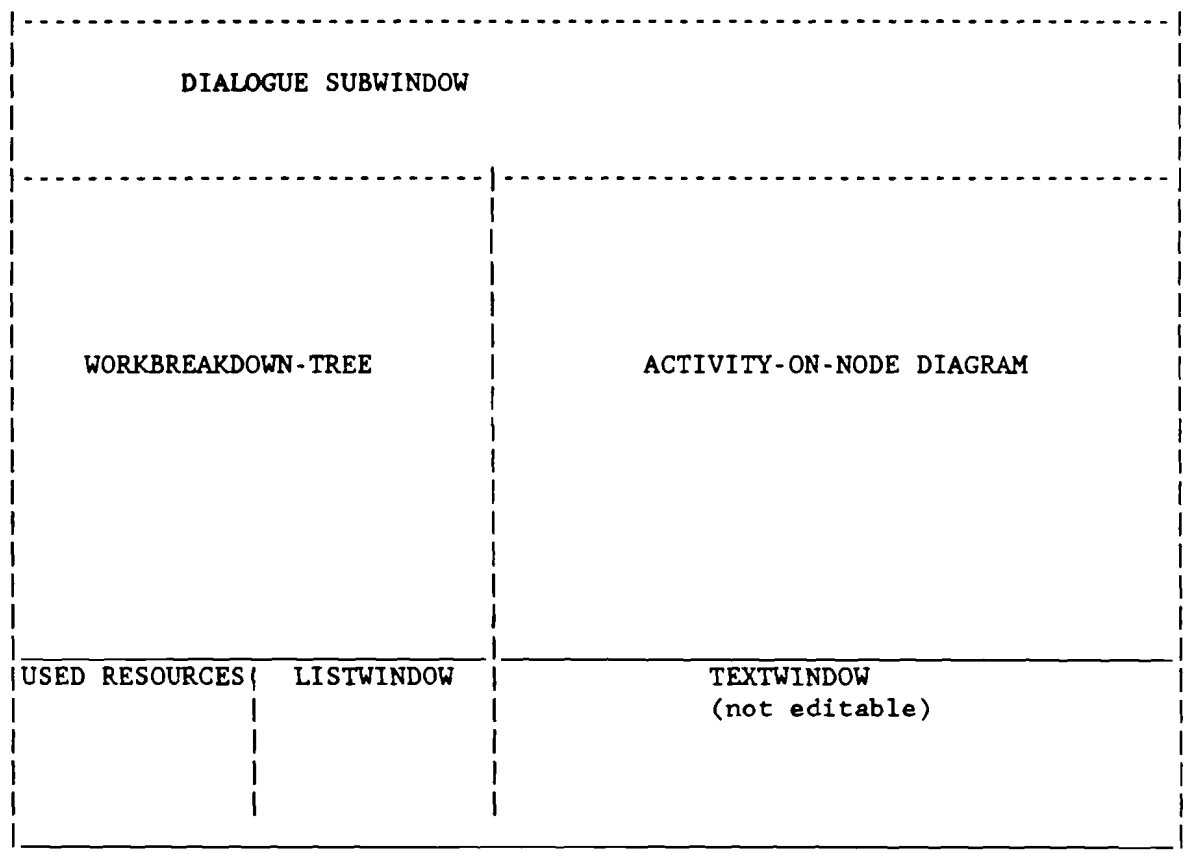


Figure 1 Display lay-out tracker

The PIMS-user can use the left button of the mouse to make a selection of the elements shown in the above mentioned windows. With the middle button of the mouse the user can get help information about the clicked element.

When the PIMS-user selects with the left button one of the command groups in the Dialogue subwindow, a pop up menu that contains the commands of the group will appear. Next, the user can select one of these commands.

In the following the commands of the tracker will be discussed by group.

2.1.4.1. Enter Project Data

Show Milestones Not Acknowledged List
This command has no parameter. When it is selected the tracker calls the Information Manager to obtain the keys and the not later than date belonging to the external milestones which have not been

acknowledged yet and belongs to the current project status. These keys, ordered by the not later than date, are displayed in the list subwindow. This listwindow, that gets the header "MILESTONES NOT ACK", can be helpful with selecting the milestones.

Show Milestone Data
This command has as parameter an external milestone key. If this key is the key of milestone that belongs to the current project view (checked by the front end), the tracker calls the information manager to obtain the milestone key value, the not later than date, the acknowledged date (which may be undefined) and the deliverables, needed to achieve the milestone, with their name, their possibly acknowledged date, and the workpackages that produce them. These data will be shown in the text subwindow, that is not editable.

Enter Milestone Acknowledged Date
This command has two parameters:

- Milestone Key
- Acknowledged Date

The front end performs the following parameters checks:

- if the entered key is the key of an external milestone, which belongs to the current project view
- if the date is an existing date and is not later than the current date

The tracker verifies:

- if on the entered date all the deliverables that belong to the milestone have been acknowledged (An external milestone has been informally achieved when all its deliverables have been acknowledged.)

If the parameters are valid, the tracker calls the Information Manager to store the acknowledged date and to obtain the milestone key value, the not later than date, the acknowledged date and the deliverables with their name, their acknowledged date, and the workpackages that produce them. Next, these data will be shown in the text subwindow, that is not editable and the possibly shown list of milestones not acknowledged will be updated.

2.1.5. Enter Task Data

Show Started Tasks But Not Acknowledged
This command has no parameter. When it is selected the tracker calls the Information Manager to obtain the data belonging to the tasks which have been started but have not been acknowledged yet and belong to the current project status. The names of these tasks, ordered by actual or planned finish date, are displayed in the list

subwindow with the header "STARTED TASKS BUT NOT ACKNOWLEDGED".

Show Task Data

This command has as parameter the name of a task which belongs to the current project status (checked by the front end). If the parameter is valid the tracker calls the Information Manager to obtain the following task data:

- the name and the description of the task
- Early Start, Early Finish, Late Start, Late Finish, Planned Start, and Planned Finish Date of the task
- if the task has been started:
    - the Actual Start Date
- if the task has been finished or acknowledged:
    - the Finish and the possibly Acknowledged Date of the task
- if the task is in progress:
    - the Forecasted Finish Date of the task (this date is derived from an extrapolation of the Estimate At Completion values and the average utilization of the allocated resources in the past, see section 2.2)
- if the deliverable of the task has been finished:
    - the Finish and the possibly Acknowledged Date of the deliverable

- the last measurement date
- if the estimated effort is available:
    - the estimated effort in manhours of the task
- the effort allocated in manhours
- if there has been spent effort on the task:
    - the effort spent in manhours
- if the remaining effort has been filled in:
    - the remaining effort in manhours
- if the task is in progress:
    - the forecast to complete in manhours (this forecast is derived from the built-in forecast algorithm, see section 2.2)

These data will be shown in the text subwindow, that is not editable.

Enter Deliverable Finish Date

This command has two parameters:

- Task Name
- Finish Date

The front end performs the following parameters checks:

- if the entered name is the name of an InProgress task

which belongs to the current project status
- if the date is an existing date and is not later than the current date

The tracker verifies:

- if the deliverable Finish Date is later than the task Start Date
- if the deliverable has not been acknowledged yet

If the parameter values are valid, the tracker calls the Information Manager to store the Finish Date. When the task finish date has been filled in and is earlier than the deliverable finish date, then this date is changed in undefined. Next, the tracker calls the function which generates the output for the command "Show Task Data".

Enter Deliverable Acknowledged Date

This command has two parameters:

- Task Name
- Acknowledged Date

The front end performs the following parameters checks:

- if the entered name is the name of an InProgress or a Finished task which belongs to the current project status
- if the date is an existing date and is not later than the current date

The tracker verifies:

- if the Acknowledged Date is later than the Finish Date of the deliverable

If the parameter values are valid, the tracker calls the Information Manager to store the Acknowledged Date. Next, the tracker calls the function which generates the output for the command "Show Task Data".

Enter Task Finish Date

This command has two parameters:

- Task Name
- Finish Date

The front end performs the following parameters checks:

- if the entered name is the name of an InProgress or a Finished task which belongs to the current project status

- if the date is an existing date and is not later than the current date

The tracker verifies:

- if the Finish Date of the task is later than or equal to the Finish Date of the deliverable (which must be filled in)

If the parameter values are valid, the tracker calls the Information Manager to store the Finish Date, update the FinishCount and to change the Task Status in Finished. Next, the tracker calls the function which generates the output for the command "Show Task Data".

Enter Task Acknowledged Date

This command has two parameters:

- Task Name
- Acknowledged Date

The front end performs the following parameters checks:

- if the entered name is the name of an InProgress or a Finished task which belongs to the current project status
- if the date is an existing date and is not later than the current date

The tracker verifies:

- if the Acknowledged Date is later than the Finish Date (which must be filled in)
- if the Acknowledged Date of the task is later than or equal to the Acknowledged Date of the deliverable (which must be filled in)
- if the Acknowledged Date of the task is later than or equal to the End Dates of the allocated resources on the task

If the parameter values are valid, the tracker calls the Information Manager to store the Acknowledged Date and to change the task status in Acknowledged. Next, the tracker calls the function which generates the output for the command "Show Task Data" and updates the possibly shown list with the header "STARTED TASKS BUT NOT ACKNOWLEDGED".

Enter Task Revised Budget

This command has two parameters:

- Task Name
- Estimated Effort (optional) in manhours

The front end performs one parameter check:

- If the entered name is the name of a NotStarted task which belongs to the current project status

If the parameter values are valid, the tracker calls the information manager to store the estimated effort; by default the amount of allocations is taken. Next, the tracker calls the function which generates the output for the command "Show Task Data".

2.1.6. Enter Resource Data

Show Task Allocated List

This command has as parameter a Person Key. The front end verifies if this key belongs to a human resource which has been allocated to one or more tasks, which belong to the current project status. If the parameter is valid the tracker calls the Information Manager to obtain the names of the tasks allocated. The names of these tasks, ordered by actual or planned start date, are displayed in a list subwindow with the header "ALLOCATIONS OF" followed by the person key.

Show Allocations

This command has as parameter a Person Key. The front end verifies if this key belongs to a human resource which has been allocated to one or more tasks, which belong to the current project status. If the parameter value is valid, the tracker calls the Information Manager to obtain the following data per task allocated:

- a list of:
    - the start date, end date, and degree of using in percent
- if the Actual Start Date of the resource on a task has been filled in:
    - the Actual Start Date
- if the Actual End Date of the resource on a task has been filled in:
    - the Actual End Date

  else:
    - the Forecasted End Date
      (this date is derived from an extrapolation of the Estimate At Completion values and the average utilization of the resource in the past, see section 2.2)

- the last measurement date
- the estimated effort in manhours
- the effort reserved in manhours
- if there has been spent effort on the task:
    - the effort spent in manhours
- if the remaining effort has been filled in:
    - the remaining effort in manhours

- the forecast to complete in manhours
  (this forecast is derived from the built-in
  forecast algorithm, see section 2.2)

These data will be shown in the text subwindow, that is not editable.

Enter Start Date

This command has three parameters:

- Person Key
- Task Name
- Start Date
- Revised estimated effort (optional) in manhours

The front end performs the following parameters checks:

- if the Person Key belongs to a human resource which has been allocated to one or more tasks, which belong to the current project status
- if the Task Name is the name of a not acknowledged task, which belongs to the current project status
- if the date is an existing date and is not later than the current date

The front performs one command check:

- if the key belongs to a resource which has been allocated to the task which belongs to the entered Task Name

The tracker verifies:

- if the End Date has not been filled in

If the parameter values are valid the tracker calls the Information Manager to store the Resource Start Date. If the task status is NotStarted, the tracker calls the Information Manager to store this date as Task Start Date. When the task status is InProgress or Finished, the tracker searches the minimum of the start dates of the resources allocated to the entered task. This minimum start date will be stored as Task Start Date. Next, the tracker calls the information manager to store the revised estimated effort; by default the amount of reservations is taken. Finally, the tracker calls the function which generates the output for the command "Show Allocations".

Enter End Date

This command has three parameters:

- Person Key
- Task Name
- End Date

The front end performs the following parameters checks:

- if the Person Key belongs to a human resource which has been allocated to one or more tasks, which belong to the current project status
- if the Task Name is the name of a not acknowledged task, which belongs to the current project status
- if the date is an existing date and is not later than the current date

The front performs one command check:

- if the person key belongs to a resource which has been allocated to the task which belongs to the entered Task Name

The tracker verifies:

- if the RemainingEffort is zero
- if the End Date is equal to or later than the Start Date

If the parameter values are valid the tracker calls the Information Manager to store the End Date. Next, the tracker calls the function which generates the output for the command "Show Allocations".

Enter Task Allocated Data

This command has five parameters:

- Person Key
- Task Name
- Measurement Date (default: the last measurement date, or, when this date is not available, the current date)
- Effort Spent in the last period in manhours
- Remaining Effort in manhours

The front end performs the following parameters checks:

- if the Person Key belongs to a human resource which has been allocated to one or more tasks, which belong to the current project status
- if the Task Name is the name of a not acknowledged task, which belongs to the current project status
- if the date is an existing date and is not later than the current date

The front performs one command check:

- if the person key belongs to a resource which has been allocated to the task which belongs to the entered Task Name

The tracker verifies:

- if the resource has a filled in Start Date
- if the resource has not a filled in End Date
- if the Measurement Date is later than the Start Date

If these parameter values are valid the tracker calls the Information Manager to update the Effort Spent and the Remaining Effort per resource and to update the Effort Spent, Cost Spent, the Remaining Effort and Remaining Cost per task, and to store the Measurement Date, which belongs to the resource data. When the Resource Remaining Effort has not been filled in, then this effort is calculated from the reservations made for this resource. If the Resource Measurement Date is later than the Measurement Date, which belongs to the task, then the latter date is changed in the entered date. If the finish date of the task has been filled in and this date is earlier than the measurement date, then the Task Status will be changed in InProgress. Next, the tracker calls the function which generates the output for the command "Show Allocations".

2.1.7. Control Reports

Resource Control Report

This command has three parameters:

- Person Key (a list of values) (the tracker removes all the entered duplicates)
- unit of measure (manhours, mandays and money) (default: manhours; money is not implemented in PIMS#1)
- summarized (yes or no) (default: no)

The front end performs one parameter check:

- if the Person Key belongs to a human resource which has been allocated to one or more tasks, which belong to the current project status

If the parameter values are valid, the following variables by resource by tasks allocated are shown in a popup textwindow, that is not editable:

- Effort/Cost Spent
- Remaining Effort/Cost
- Revised Effort/Cost
- Forecast At Completion
- Estimated Effort/Cost
- Projected Overrun
- Earned Value
- Variance
- Production Rate

The Effort Spent and the Remaining Effort can be found in the PIMS Project Conceptual Scheme via the Information Manager. The Estimated Effort is calculated from the reservations made for the resource. The Forecast At Completion refers to an extrapolation of the Estimate At Completion values in the past, based on the built-in forecast algorithm (see section 2.2). The remaining variables can be calculated as follows:

Revised Effort = Effort Spent + Remaining Effort

Projected Overrun = Revised Effort - Estimated Effort

$$\text{Earned Value} = \frac{\text{Effort Spent}}{\text{Revised Effort}} * \text{Estimated Effort}$$

Variance = Earned Value - Effort Spent

$$\text{Production Rate} = \frac{\text{Estimated Effort}}{\text{Revised Effort}}$$

If the unit of measure is money the cost can be easily calculated by multiplying the resource use by the resource rate. The shown tasks where the resource is allocated to, will be ordered by resource actual or planned start date. Summarized means that for each resource the Revised Effort, the Effort Spent, the Remaining Effort, the Forecasted Effort, the Estimated Effort, the Earned Value of every task allocated to the resource are added together. Further, the Resource Control Report will include one of the following triples of dates:

Actual Start - Actual Finish of a resource on a task - Late Finish of a task

or, if the task has not been Finished:

Actual Start - Forecasted Finish of a resource on a task - Late Finish of a tas

or, if the Actual Start date is not available:

Planned Start - Planned Finish of a resource on a task - Late Finish of a task

(How the forecasted finish date is calculated, will be explained in section 2.2.) If one of the succeeding tasks are delayed beyond their late start date (in other words these tasks become critical), the user will be warned.

Task Control Report [random task selection]
This command has three parameters:

- Task Name (a list of values) (the tracker removes all the entered duplicates, the names can be picked everywhere on the screen, except from the workbreakdown tree.)
- unit of measure (manhours, mandays and money) (default: manhours; money is not implemented in PIMS#1)
- summarized (yes or no) (default: no)

The front end performs the following parameter check:

- if every task belong to the current project status

If the parameter values are valid, the following variables by task are shown in a popup textwindow, that is not editable:

- Effort/Cost Spent
- Remaining Effort/Cost
- Revised Effort/Cost
- Forecast At Completion
- Estimated Effort/Cost
- Projected Overrun
- Earned Value
- Variance
- Production Rate

The Effort/Cost Spent, the Remaining Effort/Cost and the Estimated Effort/Cost can be found in the PIMS Project Conceptual Scheme via the Information Manager. The Forecast At Completion refers to an extrapolation of the Estimate At Completion values in the past, based on the built-in forecast algorithm (see section 2.2). The remaining variables can be calculated as follows:

Revised Effort = Effort Spent + Remaining Effort

Projected Overrun = Revised Effort - Estimated Effort

$$\text{Earned Value} = \frac{\text{Effort Spent}}{\text{Revised Effort}} * \text{Estimated Effort}$$

Variance = Earned Value - Effort Spent

$$\text{Production Rate} = \frac{\text{Estimated Effort}}{\text{Revised Effort}}$$

If the user has chosen for Not Summarized, the tracker generates a report by task by resource. The resources are ordered by actual or planned start date. The variables per resource are calculated as described in the section "Resource Control Report". Note, the sum of the Effort Spent of the resources allocated to a particular task must be equal to Effort Spent of that task. The sum of the Remaining Effort of the resources allocated to a task does not necessarily have to be equal to the Remaining Effort of that task. The reason for this is, that the resource allocator does not verify if the allocation is complete and consistent with the task data.

Further, the Task Control Report will include one of the following triples of dates:

Actual Start - Actual Finish - Late Finish of a task

or, if the Actual Finish date is not available:

Actual Start - Forecasted Finish - Late Finish of a task

or, if the Actual Start date is not available:

Planned Start - Planned Finish - Late Finish of a task

(How the forecasted finish date is calculated, will be explained in section 2.2.) If one of the succeeding tasks are delayed beyond their late start date (in other words these tasks become critical), the user will be warned.

Task Control Report [workpackage selection]
This command has three parameters:

- Workpackage (a list of values) (the tracker removes all the entered duplicates, the names can only be picked from the workbreakdown tree.)
- unit of measure (manhours, mandays and money) (default: manhours; money is not implemented in PIMS#1)
- summarized (yes or no) (default: no)

The front end performs the following parameter check:

- if every workpackage belong to the current project status

If the parameter values are valid, the following variables by task are shown in a popup textwindow, that is not editable:

- Effort/Cost Spent
- Remaining Effort/Cost
- Revised Effort/Cost
- Forecast At Completion
- Estimated Effort/Cost

- Projected Overrun
- Earned Value
- Variance
- Production Rate

The Effort/Cost Spent, the Remaining Effort/Cost and the Estimated Effort/Cost can be found in the PIMS Project Conceptual Scheme via the Information Manager. The Forecast At Completion refers to an extrapolation of the Estimate At Completion values in the past, based on the built-in forecast algorithm (see section 2.2). The remaining variables can be calculated as follows:

Revised Effort = Effort Spent + Remaining Effort

Projected Overrun = Revised Effort - Estimated Effort

$$\text{Earned Value} = \frac{\text{Effort Spent}}{\text{Revised Effort}} * \text{Estimated Effort}$$

Variance = Earned Value - Effort Spent

$$\text{Production Rate} = \frac{\text{Estimated Effort}}{\text{Revised Effort}}$$

If the user has chosen for Not Summarized, the tracker generates a report by task by resource. Tasks can be found directly via the Realisation of the workpackage or indirectly via the sons of the workpackage. The tracker removes all the duplicated tasks. The shown tasks are ordered by actual task start date or by planned task start date. The resources are ordered by actual or planned start date. The variables per resource are calculated as described in the section "Resource Control Report". The resources are ordered by actual or planned start date. Note, the sum of the Effort Spent of the resources allocated to a particular task must be equal to Effort Spent of that task. The sum of the Remaining Effort of the resources allocated to a task does not necessarily have to be equal to the Remaining Effort of that task. The reason for this is, that the resource allocator does not verify if the allocation is complete and consistent with the task data.

Further, the Task Control Report will include one of the following triples of dates:

Actual Start - Actual Finish - Late Finish of a task

or, if the Actual Finish date is not available:

Actual Start - Forecasted Finish - Late Finish of a task

or, if the Actual Start date is not available:

Planned Start - Planned Finish - Late Finish of a task

(How the forecasted finish date is calculated, will be explained in section 2.2.) If one of the succeeding tasks are delayed beyond their late start date (in other words these tasks become critical), the user will be warned.

Project Control Report

This command has three parameters:

- Workpackage (a list of values) (the tracker removes all the entered duplicates, the names can only be picked from the workbreakdown tree.)
- unit of measure (manhours, mandays and money) (default: manhours; money is not implemented in PIMS#1)
- summarized (yes or no) (default: no)

The front end performs the following parameter check:

- if every workbreakdown element belong to the current project status

If the parameter values are valid, the following variables by task are shown in a popup textwindow, that is not editable:

- Effort/Cost Spent
- Remaining Effort/Cost
- Revised Effort/Cost
- Forecast At Completion
- Estimated Effort/Cost
- Projected Overrun
- Earned Value
- Variance
- Production Rate

The Effort/Cost Spent, the Remaining Effort/Cost and the Estimated Effort/Cost can be found in the PIMS Project Conceptual Scheme via the Information Manager. The Forecast At Completion refers to an extrapolation of the Estimate At Completion values in the past, based on the built-in forecast algorithm (see section 2.2). The remaining variables can be calculated as follows:

Revised Effort = Effort Spent + Remaining Effort

Projected Overrun = Revised Effort - Estimated Effort

$$\text{Earned Value} = \frac{\text{Effort Spent}}{\text{Revised Effort}} * \text{Estimated Effort}$$

$$\text{Variance} = \text{Earned Value} - \text{Effort Spent}$$

$$\text{Production Rate} = \frac{\text{Estimated Effort}}{\text{Revised Effort}}$$

Tasks can be found directly via the Realisation of the workpackage or indirectly via the sons of the workpackage. The shown tasks are ordered by actual task start date or by planned task start date.

If the user has chosen for Not Summarized, the tracker generates a report by workpackage by task. Summarized means that for each workpackage the effort/cost spent, remaining effort/cost, forecasted effort/cost, estimated effort/cost, earned value and the revised effort/cost of every task belonging to the workpackage are added together.

Show project to do list

This command has no parameter. When it is selected the tracker calls the Information Manager to obtain data belonging to the tasks which have not been acknowledged yet. The names of these tasks, ordered by actual or planned finish date, are displayed in a list subwindow.

Destroy popup window

This command has no parameters. When it is selected, it destroys all the popup windows generated.

2.1.8. Graphs

Resource Gantt Chart

This command has as parameter a list of Person Keys. The front end checks if the Person Key belongs to a human resource which has been allocated to one or more tasks, which belong to the current project status. If the parameter values are valid, the tracker displays a TimeGraph with the following dates by resource by tasks allocated:

- the Planned Start Date, the Planned End Date
- if the Actual Start and the Actual End Date have been filled in:
    - the Actual Start Date, and the Actual End Date
- if only the Actual Start Date has been filled in:
    - the Actual Start, the Current, and the Forecasted End Date

This TimeGraph will be displayed in a pop up graphical subwindow. The tasks per resource are ordered by actual start date or planned start date.

Task Gantt Chart [random task selection]
This command has as parameter a list of Task Names. The tracker removes all the entered duplicates, the names can be picked everywhere on the screen, except from the workbreakdown tree. The front end checks if the entered tasks belong to the current project status. If the parameter values are valid, the tracker displays a TimeGraph with the following dates of the tasks:

- the Planned Start, the Planned End, and the Late Finish Date
- if the planned dates have not been filled in:
    - Early Start, Early Finish, and Late Finish Date
- if the Actual Start and the Actual End Date have been filled in:
    - the Actual Start, the Actual Finish and the Late Finish Date
- if only the Actual Start Date has been filled in:
    - the Actual Start, the Current, and the Forecasted Finish Date

This TimeGraph will be displayed in a pop up graphical subwindow.

Task Gantt Chart [workpackage selection]
This command has as parameter a list of workpackages. The tracker removes all the entered duplicates, the names can be picked only from the workbreakdown tree. The front end checks if the entered workpackages belong to the current project status. If the parameter values are valid, the tracker displays a TimeGraph with the following dates of the tasks:

- the Planned Start, the Planned End, and the Late Finish Date
- if the planned dates have not been filled in:
    - Early Start, Early Finish, and Late Finish Date
- if the Actual Start and the Actual End Date have been filled in:
    - the Actual Start, the Actual Finish and the Late Finish Date
- if only the Actual Start Date has been filled in:
    - the Actual Start, the Current, and the Forecasted Finish Date

This TimeGraph will be displayed in a pop up graphical subwindow. Tasks can be found directly via the workpackage or indirectly via the sons of the workpackage. The shown tasks are ordered by actual or planned task start date.

Project Gantt Chart

This command has as parameter a list of workpackages. The tracker removes all the entered duplicates, the names can be picked only from the workbreakdown tree. The front end checks if the entered workbreakdown elements belong to the current project status. If the parameter values are valid, the tracker displays a TimeGraph with the following dates by workpackage by tasks:

- the Planned Start, the Planned End, and the Late Finish Date
- if the planned dates have not been filled in:
    - Early Start, Early Finish, and Late Finish Date
- if the Actual Start and the Actual End Date have been filled in:
    - the Actual Start, the Actual Finish and the Late Finish Date
- if only the Actual Start Date has been filled in:
    - the Actual Start, the Current, and the Forecasted Finish Date

This TimeGraph will be displayed in a pop up graphical subwindow. Tasks can be found directly via the workpackage or indirectly via the sons of the workpackage. The shown tasks are ordered by actual or planned task start date.

Destroy popup window

This command has no parameters. When it is selected, it destroys all the popup windows generated.

2.2. Predictor Part

2.2.1. Introduction

If there is a (serious) discrepancy between the plan and reality, generally the projectmanager wants to know what the influence of this delay will be on the scheduled project completion date. In this case a new prediction of the effort and timescale has to be made. For that we see the next three possibilities:

- starting a re-planning process
- extrapolating the measured values on the basis of the project curve
- exponential smoothing (short-term forecasting)

The first possibility needs a diagnosis tool to determine the cause(s) of a certain discrepancy between plan and reality. If the cause(s) is (are) found and there is no 'remedy' that does not affect the plan, a replanning process has to start by which one or more of the following things can be needed: completing or re-structuring the workbreakdown, re-estimating the effort and duration of task(s), re-scheduling the tasks, re-allocating the resources. The foregoing shows, if there is (or seems to appear) a serious deviation from plan, the planning tools can be used to make a new and hopefully more accurate, (long-term) prediction of the effort and timescale of the project. This possibility will be discarded here, because in PIMS#1 a diagnosis tool is not available and re-planning is beyond the scope of the planning tools.

The second possibility to make a long term prediction extrapolates the measured values on the basis of the equation (curve), that is assumed to represent the project model (for project modelling techniques, see for instance [Wissing, 87a]). So, in this case we suppose that a project follows the curve that is described by this equation. Since in PIMS #1 the project curve is unknown, the predictor will also not cover this long-term prediction.

The third possibility makes only short-term prediction possible. To predict for this immediate future, forecasting techniques will be used to estimate the effort/cost to complete a task. The reason that those techniques are only useful for short term prediction is, that they are based on the assumption that existing pattern will continue into the future and this assumption is more likely to be correct over the short term than it is over the long term.

The third possibility will be the one that will be supported by the predictor in PIMS#1. It will be based on the revised effort (the revised effort is equal to the effort spent plus the remaining effort). From this and the average utilization of the resource in the past, a prediction is made of the expected finish date. The prediction mechanism will never by itself modify any scheduling or allocation data.

2.2.2. Inputs and Outputs

To be able to extrapolate the past performance using forecasting techniques, the following inputs are required:

- Overall project Data:
  - the project calendar
- list of Reservations:
  - Task
  - the Remaining Effort at the measuring points of time (We assume that the measurements are performed at regular time intervals, for instance weekly; this assumption is not essential, but increases the reliability of the forecasting)
  - the Effort Spent (WSTD) at the measuring points of time
  - actual start date of the resource (where the reservation belongs to) on the task
  - actual finish date of the resource (where the reservation belongs to) on the task
  - list of
    - planned startdate
    - planned finish date
    - percent of usage of the resource
- Task Data
  - the Task Estimated Effort
  - actual finish date
  - planned finish date
  - status

The forecasting mechanism gives the following outputs:

- the Forecasted Effort To Complete of a resource on a task
- the Forecasted Effort To Complete of a task (this Forecast is based on the forecasts made per resource allocated)
- forecasted finish date of a resource on a task
- forecasted finish date of a task

2.2.3. Processing

For forecasting the revised effort (the revised effort is equal to the effort spent plus the remaining effort) we use the Holt's method as has been described in [Wissing, 87]. On the basis of this forecasted effort, the average effort spent in the previous period and the project calendar, the expected finish date of the resource on the task will be calculated.

The forecasted effort and forecasted finish date of a task will be derived from the forecasted effort and forecasted finish date of the allocated resources on the task (if the forecasted effort and thus the forecasted finish date of a resource on task is not available, the initially estimated effort and the planned finish date will be taken as the forecasts).

The forecast algorithm fails, when according to the historical data the remaining effort increased more than the effort spent, in other words: one has made negative progress. No sensible forecast can be made then, since in this situation, the task would never be completed.

3. Future extensions of the tracker

In view of the time available and the possibilities of the planning tools in PIMS#1 the possibilities of the tracker have been reduced to what is mentioned in Chapter two. In the document BSO-G-RR-102 there has been described what a tracker might do. In this chapter will be discussed the supplemental inputs and outputs required to make the tracker more advanced. Further, the processing in the advanced tracker will be described globally.

3.1. Inputs

Again, we divide the inputs into two groups:

- Inputs required from the other tools and the system;
- User inputs.

3.1.1. Inputs required from the other tools and the system

The advanced tracker requires the following supplemental inputs of the other tools via the information manager:

- Project:
  - list of Project Risk Factors and their predicted values (derived from the Risk tools)

- ProjectStatus:
  - list of WatchDogs (In the PIMS#1 architecture WatchDogs (Deamons) have not been implemented)
  - list of Internal Milestones (In PIMS#1 none of the planning tools 'works' with internal milestones, the scheduler/resource allocator for instance does not take them into account)

- Project Model data:
  - the Project Model
  - list of Project Metrics, their predicted values, and eventually needed transformations (see section 2.3 in [Wissing, 87]) by task/workpackage class

- Workpackage data:
  - class where the workpackage belongs to

- Task data:
  - slack (with respect to the milestones)
  - class where the task belongs to

- Risks:
  - list of Schedule Risk Factors and their predicted values

- Durable, Consumable Resource data:
    - Name
    - Resource Availability
    - Cost by Hour
    - list of tasks that have reserved the resource and a (list of) degree of use and slack per task
    - Measurement Date

- Human Resource data
    - Slack per task

- Deliverable:
    - the milestone where the deliverable belongs to

3.1.2. User inputs

The user may give the tool the following kinds of additional inputs compared with Chapter two:

- Overall project:
    - Internal Milestones Achieved Date
    - New WatchDogs

- Project Model data:
    - the Project Metrics which have to be tracked
    - actual values of the selected Project Metrics, by task/work-package class

- Task data:
    - New one-off costs (In PIMS#1 none of the planning tools works with one-off costs)

- Task/Resource (Durable, Consumable) data:
    - Hours Spent per task, per resource
    - Current Estimated Time to go by resource, by task
    - Actual Start Date of a resource on a task
    - Actual End Date of a resource on a task

3.2. Outputs

We divide the outputs, just like the inputs, into two groups:

- outputs to the system;
- user outputs.

3.2.1. Outputs to the system

The tool will give the system the following additional outputs:

- Overall project:
  - Milestones Achieved Dates
  - New WatchDogs

- Project Model:
  - the Project Metrics which have to be tracked
  - actual values of the selected project metrics at the measuring points of time

- Resource (Consumable, Durable) data:
  - Actual Start Date of a resource on a task
  - Actual End Date of a resource on a task
  - WSTD by task
  - WRE by task

- quality assurance data:
  - quality of the estimates
  - quality of the model
  - warnings from WatchDogs
  - anticipated slippages (for instance milestones that never can be reached on time according to the data now available)

Note, most of these outputs will also be required by the tracker as well.

3.2.2. User outputs

The user can get the following kinds of additional outputs (compared with chapter two):

- Tabular reports:
  - Overall project:
    - Milestones Achived Dates
    - Warnings from WatchDogs
    - Anticipated slippages

  - Project Model:
    - The help to select the project metrics
    - The advices concerning the influences the several metrics posses with respect to the progress
    - Discrepancies between the actual and estimated metric values and their class (indicating the severity of the observed discrepancy)

  - Resource (Consumable, Durable) Control:
    - Reserved Time Interval by resource by task

        - Actual Start Date of a resource on a task
        - Actual End Date of a resource on a task
        - BW, WSTD, WRE, PTW, BEWP, PVTY, PEO, EV by resource by task or summarized

    - Quality Assurance:
        - quality of the estimates
        - quality of the model

- Graphs:
    - Task/Workpackage:
        - PercentComplete diagram: Item_Actual%C, Item_Planned%C against time ('Against time' diagrams are not possible in PIMS #1, because historical data is a problem.)
        - EffortPerformance diagram: BEWP, WSTD against time
        - CostPerformance diagram: BCWP, ACWP against time
        - Productivity diagram: PVTY against time

    - Resource:
        - EffortPerformance diagram: BEWP, WSTD against time
        - Productivity diagram: PVTY against time

3.3. Tools required by the advanced tracker

Besides the Front End tool and the Information Manager the advanced tracker requires the following tools:

- Report Generator for generating the formal reporting
- Decision Support System for helping to select the project model metrics that must be tracked
- Expert System:
    - for giving advices concerning what influences the several metrics possess with respect to the progress
    - for determining the severity of discrepancies
- Spreadsheet for making further calculations by the user.

3.4. Processing

In this section the processing in the advanced tracker will be described globally. Initial the PIMS-user has to select the project model metrics he wants to track for helping to determine the progress and costs. An expert (knowledge based) system, which supports a forward reasoning strategy, in view of the needed what-if character, can be used for the selection of the wanted metrics. This expert system should give advice to the PIMS-user concerning which influences the several metrics possess with respect to the progress. This advice together with a decision support system can help the user to make the right decision with selecting the metrics [Wissing, 87]. During the following runs of the tool the user has to enter the new data that are available at that time. If the

tracker notices a discrepancy between the estimated value and the actual value of a metric, this discrepancy should be classified. Since the classification into discrepancy classes depends, among other things, on the history of a task, it is clear, that the previous states of the task must be stored. It makes a lot of difference when one notices a discrepancy between the amount of work completed whether there is an improvement compared with the previous state [Jong et al., 87]. The classifying into a discrepancy class can be done with the aid of an Expert System. For expressing the performance of cost and effort the tool has to calculate the values of the several variables (see Chapter 5). These values together with the slack can be used to determine future slippages, like milestones that never can be reached on time, according to the data now available.

4. Glossary

The terms in this document are generally those of the Glossary of Software Engineering Terminology (ANSI/IEEE Std 729-1983) [IEEE, 84]. Extensions to this list include the following:

activity
: Individual item of work which along with other items of work comprises the project.

activity-on-node diagram
: A network diagram, where each node represents an activity and each arc shows logical precedence. This approach is most commonly used with the critical path method (CPM).

availability
: The time interval in which a resource can be used.

consumable resource
: A resource that can be used only once, e.g. processor time; thus a resource that is not re-usable (durable).

critical activities
: The activities along the critical path. A delay in any one of them will cause a delay in the completion of the entire project.

critical path
: The longest path in a network. It determines the project length in the sense that precedence relations permit no earlier completion of the project.

deliverable
: The final object of a task.

dependency
: The logical relationship between activities within a project due to time or causality or both.

durable resource
: A resource that can be used more than once. Contrast with consumable resource.

early finish time
: The earliest time at which the task can be completed.

early start time
: The earliest time at which the task can begin.

estimate
: A judgement that is equally likely to be above or below the actual result.

event
The fact of a thing's happening, where the thing is not directly influenced by the project, like the delivery of a machine.

forecast
An estimate of something in the near future based on a computerized extrapolation of the past.

free slack
The allowable delay in a task start time that can be absorbed without delaying another task.

late finish time
The latest time at which the task can finish without delaying the project.

late start time
The latest time at which the task can begin without delaying the project.

model
A miniature representation of a complex reality.

monitoring progress
The means by which feedback on actual progress against planned progress is provided to the users to inform them of the budgetary and technical development of the project. Monitoring can be undertaken at random intervals or at pre-determined points.

network diagram
This is a graphical model for analysis of the scheduling problem. All diagrams use nodes and arcs.

path
This is a sequence of connected activities in a network diagram, following a set of precedence arrows from the start of a project to its completion.

predicting
A long term estimate.

project manager
The individual who is responsible for the successful running and completion of the project.

project metric
A quantifiable attribute of the project.

resource allocation
The apportioning of the available and required resources to the various activities which form the project.

Software project
: A series of activities whose major deliverable is an item of software.

standards
: Prescribed rules that have to be followed during the development of a software product.

task
: Activity within the project (see activity)

total slack
: The allowable delay in a task start time that can be absorbed without delaying the project, assuming that all other tasks begin as early as possible.

tracking progress
: The recording and monitoring of how much work has been done, what resources were utilized, and what costs were incurred. At any update or measurement point, which should occur at regular intervals or at pre-determined points (milestones). It also entails keeping track of any changes in the project plan.

user
: The person or persons who operate or interact directly with the system.

Work Breakdown
: The organizing of the project activity elements into a hierarchical structure. This structure is composed of two hierarchies, which can be connected in whichever way best fits the project. These two hierarchies are:

    -the product hierarchy;
    -the activity hierarchy.

5. Abbreviations

In this document the following abbreviations are used:

ACWP (Actual Cost of Work Performed):
At any specified point in time the actual cost incurred for the work

BAC (Budget At Completion):
The initial budget for a task, workpackage or the entire project

BCWP (Budgeted Cost of Work Performed):
At any specified point in time, the actual percent complete, times the BAC for that item. This is the earned value of the work performed (BCWP = Item_actual%C$_c$ xBAC, where Item_actual%C$_c$ is ACWP/EAC)

BEWP (Budgeted Effort of Work Performed):
At any specified point in time the actual percent complete, times the BW, for that item (BEWP = Item_actual%C$_e$ xBW, where Item_actual%C$_e$ is WSTD/PTW).

BW (Budgeted Work):
The initial budget for a task, workpackage or the entire project in MM.

CV (Cost Variance):
The difference between the value of the work performed and the actual cost for that work (CV = BCWP - ACWP)

EAC (Estimate At Completion):
The sum of the actual cost to date, plus the estimate to complete (EAC = ACWP + ETC)

ETC (Estimate To Complete):
An estimate of the cost to be incurred to complete the remaining work

EV (Effort Variance):
The difference between the value of the work performed and the actual effort spent for that work (EV = BEWP - WSTD).

FAC (Forecast At Completion):
The sum of the actual cost to date plus the forecast to complete (FAC = ACWP + FTC).

FEAC (Forecasted Effort At Completion):
The sum of the actual effort to date plus the forecasted effort to complete (FEAC = WSTD + FETC).

FETC (Forecasted Effort At Completion):
A forecast of the effort to be incurred to complete the remaining work. Forecast refers to a computerized extrapolation of the

performance to date, based on a built-in algorithm, and estimate refers to a judgemental expression of the effort of the remaining work.

FTC (Forecast To Complete):
A forecast of the cost to be incurred to complete the remaining work. Forecast refers to a computerized extrapolation of the performance to date, based on a built-in algorithm, and estimate refers to a judgemental expression of the cost of the remaining work.

PCO (Projected Cost Overrun):
The difference between the projected total cost and the budgeted cost (PCO = EAC - BAC).

PEO (Projected Effort Overrun):
The difference between the projected total work and the budgeted work (PEO = PTW - BW).

PTW (Projected Total Work):
The sum of the actual effort to date, plus the estimate to complete (PTW = WSTD + WRE).

PR (Production Rate):

$$\frac{BAC}{EAC}$$

PVTY (ProductiViTY):

$$\frac{BW}{PTW}$$

SV (Schedule Variance):
The difference between the value of the work performed, and the value of the work that had been planned to be performed, at the measurement time (SV = BCWP - BCWS)

WBD (Work Break Down):
The organizing of the project activity elements into a hierarchical structure. This structure is composed of two hierarchies, which can be connected in whichever way best fits the project. These two hierarchies are:

- the product hierarchy;
- the activity hierarchy.

WBS (Work Breakdown Structure):
The result of a WBD.

WRE (Work Remaining Estimate):
An estimate of the effort needed to complete the remaining work.

WSTD (Work Spent To Date):
At any specified point in time the actual effort incurred for the work.

6. References

[IEEE, 84]
IEEE, "Software engineering standards", IEEE Inc, 1984

[Fernstrom et al., 87a]
Fernstrom, C., Doize, M., "User's manual for the PIMS Information Manager", CSI, CSI-F-SP-004d, 1987

[Fernstrom et al., 87b]
Fernstrom, C., Konc, S., Paris, J., "Tool builder's handbook for PIMS#1", CSI, CSI-F-SP-003c, 1987

[Jong et al., 87]
Jong, de T., Hoog, de R., and Schreiber, G., "Monitoring, Diagnosis, and Forward Tracking in software Project Management", University of Amsterdam, UVA-D-RR-005a, 1987

[Wissing, 87]
Wissing, R., "Modelling, tracking, and predicting in software project management", BSO, BSO-G-RR-102, 1987

Specifications of predicting in PIMS#1

Synopsis

This document describes the specifications of short-term predicting in PIMS#1.

Authors : RF Wissing, BSO/Eindhoven
Reader : H vd Klok
Level : BSO/Eindhoven BV
Status : InCirculation
Identifier: BSO-G-SP-108.n (Release : 1.4)
Created : 88/01/05 (Edited : 88/01/22)
Printed : 22 January 1988

This document is part of a project partially funded by the Commission for the European Communities ESPRIT programme, as project number 814.

© 1988 BSO/Eindhoven BV

Distribution List

Christer Fernstrom	CSI
John Hawgood	PA
Robert de Hoog	UvA
Patrick Humphreys	LSE
John Jenkins	938
Hans van de Klok	BSO
Frank Land	LBS
Eirik Naess-Ulseth	SI

Revision Information

Creation Date:
Author :
Reader :
Modifications:

Table Of Content

1. Introduction

If there is a (serious) discrepancy between the plan and reality, generally the projectmanager wants to know what the influence of this delay will be on the scheduled project completion date. In this case a new prediction of the effort and timescale has to be made. For that we see the next three possibilities:

- starting a re-planning process
- extrapolating the measured values on the basis of the project curve
- exponential smoothing (short-term forecasting)

The first possibility needs a diagnosis tool to determine the cause(s) of a certain discrepancy between plan and reality. If the cause(s) is (are) found and there is no 'remedy' that does not affect the plan, a replanning process has to start by which one or more of the following things can be needed: completing or re-structuring the workbreakdown, re-estimating the effort and duration of task(s), re-scheduling the tasks, re-allocating the resources. The foregoing shows, if there is (or seems to appear) a serious deviation from plan, the planning tools can be used to make a new and hopefully more accurate, (long-term) prediction of the effort and timescale of the project. This possibility will be discarded here, because in PIMS#1 a diagnosis tool is not available and re-planning is beyond the scope of the planning tools.

The second possibility to make a long term prediction extrapolates the measured values on the basis of the equation (curve), that is assumed to represent the project model (for project modelling techniques, see for instance [Wissing, 87a]). So, in this case we suppose that a project follows the curve that is described by this equation. Since in PIMS #1 the project curve is unknown, the predictor will also not cover this long-term prediction.

The third possibility makes only short-term prediction possible. To predict for this immediate future, forecasting techniques will be used to estimate the effort/cost to complete a task. The reason that those techniques are only useful for short term prediction is, that they are based on the assumption that existing pattern will continue into the future and this assumption is more likely to be correct over the short term than it is over the long term.

The third possibility will be the one that will be supported by the predictor in PIMS#1. It will be be based on the revised effort (the revised effort is equal to the effort spent plus the remaining effort). From this and the average utilization of the resource in the past, a prediction is made of the expected finished date. The prediction mechanism will never by itself modify any scheduling or allocation data.

In fact a forecast can be considered as an extra variable belonging to the reports generated in the tracker [Wissing, 87b]. So, in order to minimize the number of tools in PIMS#1, we propose to integrate both tools.

2. Inputs and Outputs

To be able to extrapolate the past performance using forecasting techniques, the following inputs are required:

- Overall project Data:
  - the project calendar
- Resource Data:
  - the NonAvailability of a Resource
- list of Reservations:
  - Task
    - the Remaining Effort at the measuring points of time (We assume that the measurements are performed at regular time intervals, for instance weekly; this assumption is not essential, but increases the reliability of the forecasting)
    - the Effort Spent at the measuring points of time
    - actual start date
    - actual finish date
    - list of
      - planned startdate
      - planned finish date
      - percent of usage of the resource
- Task Data
  - the Task Estimated Effort
  - actual start date
  - actual finish date
  - planned startdate
  - planned finish date
  - the Late Finish Date
  - status

The forecasting mechanism provides for the following additional outputs in the reports generated in the tracker:

- Resource Control Report:
  - the Forecasted Effort To Complete
  - actual start date or planned start date
  - actual finish date or forecasted finish date or planned finish date of the resource on a task (see Chapter 4)
  - late finish date of a task as produced by the scheduler
  - warnings with respect to the anticipated difficulties (see Chapter 4)

- Task Control Report:
  - the Forecasted Effort To Complete (this Forecast is based on the forecasts made per resource)
  - actual start date or planned start date
  - actual finish date or forecasted finish date or planned

finish date
- late finish date as produced by the scheduler
- warnings with respect to the anticipated difficulties

3. Changes required in the tracker

Forecasting requires a few minor changes in the tracker:

- the tracker has to store the Remaining Effort per resource and the Effort Spent per resource at the measurement points of time in a historical sequence [Dixon, 87]
- when the allocated resource starts working on a task, it must be possible to change the initially estimated effort of every resource per task. (By default the amount of allocations will be taken.) To allow a good forecast, the person who makes this revised initial estimate has to be the same person who makes the further estimates.
- it must be possible to change the estimated effort of every task, as long as the effort spent is equal to 0.

4. Processing

For forecasting the revised effort (the revised effort is equal to the effort spent plus the remaining effort) we use the Holt's method as has been described in [Wissing, 87a]. On the basis of this forecasted effort, the average effort spent in the previous period and the project calendar, the expected finish date of the resource on the task will be calculated. With the help of the expected finish date and the forecasted effort several checks will be made. If any check fails, the user will be warned in the resource control report and in the task control report for the potential difficulties. These checks refer to:

- the availability of a resource when a task is delayed or takes longer than planned (check if the availability of the resource is sufficient; this check will be based on the assumption that the average utilization in the past still follows this pattern in the future)
- the succeeding tasks that are delayed beyond their late start date (in other words these tasks become critical)

Further, the resource control report will be extended with one of the following triples of dates:

Actual Start - Actual Finish of a resource on a task - Late Finish of a task

or, if the task has not been Finished:

Actual Start - Forecasted Finish of a resource on a task - Late Finish of a tas

or, if the Actual Start date is not available:

Planned Start - Planned Finish of a resource on a task - Late Finish of a task

The task control report will be extended with one of the following triples of dates:

Actual Start - Actual Finish - Late Finish of a task

or, if the Actual Finish date is not available:

Actual Start - Forecasted Finish - Late Finish of a task

or, if the Actual Start date is not available:

Planned Start - Planned Finish - Late Finish of a task

The forecasted effort and forecasted finish date of a task will be derived from the forecasted effort and forecasted finish date of the allocated resources on the task (if the forecasted effort and thus the

forecasted finish date of a resource on task is not available, the initially estimated effort and the planned finish date will be taken as the forecasts).

The forecast algorithm fails, when according to the historical data the remaining effort increased more than the effort spent, in other words: one has made negative progress. No sensible forecast can be made then, since in this situation, the task would never be completed. If this occurs, the reason why no forecast is given, will be displayed in the reports.

5. Glossary

The terms in this document are generally those of the Glossary of Software Engineering Terminology (ANSI/IEEE Std 729-1983) [IEEE, 84]. Extensions to this list include the following:

availability
: The time interval and percentage in which a resource can be used.

critical activities
: The activities along the critical path. A delay in any one of them will cause a delay in the completion of the entire project.

critical path
: The longest path in a network. It determines the project length in the sense that precedence relations permit no earlier completion of the project.

deliverable
: The final object of a task.

dependency
: The logical relationship between activities within a project due to time or causality or both.

early finish time
: The earliest time at which the task can be completed.

early start time
: The earliest time at which the task can begin.

estimate
: An approximate judgement that is equally likely to be above or below the actual result.

forecast
: An estimate of something in the near future based on a computerized extrapolation of the past.

free slack
: The allowable delay in a task start time that can be absorbed without delaying another task.

late finish time
: The latest time at which the task can finish without delaying the project.

late start time
: The latest time at which the task can begin without delaying the project.

model
: A miniature representation of a complex reality.

monitoring progress
: The means by which feedback on actual progress against planned progress is provided to the users to inform them of the budgetary and technical development of the project. Monitoring can be undertaken at random intervals or at pre-determined points.

network diagram
: This is a graphical model for analysis of the scheduling problem. All diagrams use nodes and arcs.

path
: This is a sequence of connected activities in a network diagram, following a set of precedence arrows from the start of a project to its completion.

prediction
: A long term estimate.

project metric
: A quantifiable attribute of the project.

resource allocation
: The apportioning of the available and required resources to the various activities which form the project.

software project
: A series of activities whose major deliverable is an item of software.

tracking progress
: The recording and monitoring of how much work has been done, what resources were utilized, and what costs were incurred. At any update or measurement point, which should occur at regular intervals or at pre-determined points (milestones). It also entails keeping track of any changes in the project plan.

user
: The person or persons who operate or interact directly with the system.

6. References

[Dixon, 87]
Dixon, R.K., "The Maintenance of Historical Project Data in PIMS#1", BSO, BSO-I-RR-102, 1987

[Holt et al., 60]
Holt, C.G., et al., "Planning production, inventories and work force", Prentice Hall, 1960

[IEEE, 84]
IEEE, "Software engineering standards", IEEE Inc, 1984

[Wissing, 87a]
Wissing, R., "Modelling, tracking, and predicting in software project management", BSO, BSO-G-SP-102, 1987

[Wissing, 87b]
Wissing, R., "Specification of tracking in software project management", BSO, BSO-G-SP-104, 1987