SYSTEM PROGRAMMING WITH A COMPUTER MANUFACTURER

Mr. J. Thow

Manager,
Computer and Programming Systems,
I.B.M. United Kingdom Limited,
Hursley Laboratories,
Near Winchester, Hants.

Abstract:

The scope of the work open to University graduates entering the system programming department of a large computer manufacturer is described, and the opportunities for further training provided within the company are outlined. The technical environment in which this work is carried out is illustrated by a description of the program development process. Finally, some aspects of the physical environment are discussed.

Rapporteurs:

Mr. M. J. Elphick
Mr. J. F. Dunn

1. The scope of the work

Mr. Thow began his talk with an apology on behalf of his colleague Mr. John Nash (well known for his work on the development of PL/1), who had been scheduled to give this talk but was unable to do so. Mr. Thow then went on to say that he had enjoyed the seminar very much, and although he felt that some of the arguments made had been indefensible, he had been vastly encouraged by the sheer enthusiasm shown for computer science which must surely be passed on to students. For a young man entering his department, enthusiasm is essential. 'We can channel enthusiasm, with apathy we can do nothing'. The ability to think without prejudice and to discuss matters clearly and forcibly is also important, as much work in systems programming with a manufacturer is done by talking in groups and at meetings. It is immaterial what programming languages are taught at colleges since students entering a programming department will be trained in the languages they are to use.

Figure 1 shows the four broad types of work classed as systems programming. General applications packages for the major industries are included in 'System Programming' because, as Mr. Jackson had remarked, many of these packages are becoming so large that they are in effect becoming systems. In advanced technology, new techniques in the production and development of products are investigated, and new languages are simulated using old ones to see how they will work and what will be their use and impact. The personalities required are very diverse, and each entrant to the department tends to gravitate into whichever of the four areas of work shown is suited to his personality: the more aggressive 'task oriented' people go into product development and the more abstract and academic into language; the 'inventors' choose advanced technology and the 'salesmen' systems support.

In Figure 2 are shown the sort of things people do in a systems programming department. Most of these need no explanation. Programmers are encouraged to enter any of the sections shown, even though the connection with programming may not be obvious. For example, if more programmers join the publications writing section, the readability for other programmers of the literature produced should be improved. Test development is regarded by many programmers as a secondary activity, but it is important that good programmers join this department and help to develop new tools for testing products. The difficulty in test development lies in the size of many of the programs to be tested; these programs are required before release to be capable of handling correctly practically all of the known test cases and should be able to cope reasonably with the few remaining cases. Nevertheless, customers will still find errors in the programs, showing that better tests are needed. Programmers are useful in the systems integration department, where components developed at various centres are put together and tested prior to release. They can also help develop programming tools which control the development of a product, its updating and correction, and the backtracking necessary in the event of a major error.

2. Further training

Apart from the normal induction course that most companies put new recruits through, I.B.M. provide a four month 'graduate course'. This is divided into four modules:

1. An Assembler course (including the use of macros).
2. A PL/1 course.
3. The basics of JCL (Job Control Language).
4. Computer room procedures.

The first two courses each take almost two months, while the last two together occupy only about one week. Teaching is by lectures and practical work; during the latter people work together in small teams. The ability to cooperate in team work, and the realisation that different orders of magnitude give rise to different problems, are two things that companies like I.B.M. would like to see encouraged in Computing Science courses.

After this initial course, the graduate is now able to make effective use of the programming languages adopted by I.B.M. and to work efficiently in this environment. During his second and third years as a programmer a set of intermediate level courses are available, and Figure 3 illustrates the range of topics covered by these. This is an extract from a handbook describing the European Laboratories Intermediate Professional Training (ELIPT) program. Some of the courses might be considered to be purely educational — for example, course 060 (Formal Definition) covers the 'Vienna technique' for the formal definition of programming language semantics — while others are more practical, such as course 070 (Implementation of PL/1) which is intended to show someone using PL/1 how to make the best use of the current compilers. Not all programmers do all courses; on average, a programmer will spend one to two weeks annually on such courses. As Mr. d'Agapeyeff observed for CAP, the company finds that this is about as much formal classroom training as can be afforded after the initial four months course.

Referring to Figure 3, Professor Duijvestijn asked what was taught under the heading of course 055, 'Systems Implementation Languages'. In reply, Mr. Thow said that at the moment the emphasis was on 'clever ways of using macros' to help in writing systems, together with some theoretical speculation on, e.g. how one could use PL/1 to write a PL/1 compiler, and what features a good systems implementation language ought to have.

3. The technical environment

On joining a computer manufacturer, the new programmer will almost certainly find himself part of a team, probably workong on a rather large system. As is well known, the complexity of such large systems grows rapidly with their size and requires the imposition of standards, procedures and controls to ensure their orderly development.

(At this point in his lecture, Mr. Thow illustrated some of the problems of management and control in large organisations by a number of slides, of which we reproduce one in Figure 4.

This demonstrates several points, including:

1. The tendency for the 'workers' to sit down and cheer from the sidelines; in a large organisation, it is often difficult to appreciate that things can be changed.
2. The need for complete (as well as exact) specifications — here, each team has followed its specification, building the second rail on the right of the central one.
3. The need for a pragmatic solution to such calamities — as a pragmatist, the speaker would insert an S-bend (of a length depending on the time left before completion) and issue an appropriate warning. An academic might possibly devise an inversion operator which, when applied to a train at the junction, would reflect it about its left-hand wheels!)

Standards cover a wide range, from the way code should be laid out (for ease of reading) and the precise amount of documentation required to the naming of variables and the maximum size for a module. They are collected together in a manual for the programmer's guidance. Procedures and controls are as essential as they are numerous, and can all be rather confusing to a new programmer. One aspect particularly difficult for someone coming from a University is the change from an environment in which knowledge is unrestricted and publication encouraged to one in which everything he knows and learns about some particular concept is restricted, and will not become public until some future 'announcement date'. This is illustrated by Figure 5, showing the variation in the information available for a typical product over the period from its design and development to final release.

In order to demonstrate the sort of control encironment that the new programmer will find, Mr. Thow went briefly through the program development process as used by I.B.M., and illustrated by Figures 6 to 9. In Figure 6 the various stages in this development process are listed; with a large system, the average time elapsing between acceptance of detailed programming objectives and final system test is about two years. Programming objectives are prepared by marketing people and programming specifications by those who will implement the system — there is an iteration round here until agreement has been reached, before detailed logic specifications are prepared and coding starts. Tests are down at both component and system level; the 'alpha' test is an initial test of feasibility which may be on paper, or may include some running code, while 'beta' tests are designed to prove that the product is ready for shipment. These final acceptance tests are followed by release, maintenance and eventually withdrawal.

An analysis of how resources are used in the successive phases of development is shown by Figure 7. It is interesting to note that at the peak of the project the real programming activity (in the top shaded area) accounts for rather less than half of the total resources being used. The next figure (Figure 8) details the manner in which the development of such a system is controlled, and shows the division of the project into 'phases' (0 to VI). Control is applied by having a 'phase meeting' of all people concerned at the management level who must agree that the project has met the objectives of the current phase (the end-points) before moving onto the next one. Even more detail is shown in Figure 9; basically, this indicates which documents (plans, estimates, objectives, etc.) will be required at the various phase meetings listed along the top. The number of people involved in such meetings will typically be around 20; they will get detailed presentations by the project manager and his team, raise objections if necessary, and eventually decide whether the project is ready to move to the next phase. A lot of the documentation for a project will be gathered into a System Manual; the responsibility for this lies with the project manager but the programmers themselves will be heavily involved in the preparation of the various documents.

4. The physical environment

Mr. Thow continued with a number of slides showing some of the very attractively situated buildings in which I.B.M. carries on its business; these included the laboratories at La Gaude (France, Uithoorn (Netherlands), Yorktown (U.S.A.) and Hursley (England). Although the atmosphere and surroundings in which the programmer finds himself are important, they are not as important as what he finds inside the buildings.

Some experiments in interior planning had been carried out in I.B.M., attempting to introduce the concept of the open plan 'landscaped office', in which a single large space is broken up into work, discussion and rest areas by the layout of furniture, plants, etc. rather than by internal partitions. The noise level in such an office is an important factor, and most programmers have not been convinced that their style of working, alternating between intense (and often noisy) discussion and long periods of uninterrupted concentration, would fit into this environment.

In general, programmers still work in 'conventional' offices holding between two and six (occasionally eight) people. Where possible, access to computing services will be via a terminal, serving two to four people and providing both interactive and remote job entry facilities.

Continued thinking about the needs of programmers has given rise to the idea of a 'programmer work station', providing in a compact unit lots of storage space and flat working surfaces. Several prototypes had been built; these were in most cases L-shaped, for reasons both of privacy and of concentrating the working surfaces.

This, then has been a very brief outline of the type of work that awaits your graduates and some idea of the environment in which they will find themselves.

5. <u>Discussion</u>

<u>Professor Pengelly</u> remarked that there was clearly a very large information-handling problem in this sort of work, and asked what role the computer might play in solving this.

<u>Mr. Thow</u> replied that this was a difficult program in its sheer size, as was the type of commercial data-processing problem discussed earlier; in his department, a Laboratory Information Systems Group had been set up, with the aim of getting the computer to assist in controlling the vast amount of information generated and presenting it in a timely fashion.

<u>Professor Randell</u> observed that Mr. Jackson had referred earlier to 'those programmers who won't read, don't know and don't <u>want</u> to know'. Clearly, I.B.M. put a lot of effort into training programmers — should the company be their sole source of information, or should there be journals and other sources of information available, which programmers themselves regard as essential to keep up with?

<u>Mr. Thow</u> pointed out that the Company was interested in training programmers to do a job - and the education given was directed towards this end. They could do no more than encourage people to join societies (and give them time off for committee work) and to read the journals provided in the libraries. The initiative <u>must</u> come from the individual programmer.

<u>Mr. Jackson</u> asked how the speaker regarded the classical dilimma of management: either one has professional managers who are not themselves competent programmers, or one promotes good programmers to manage others and loses their talents?

In reply, <u>Mr. Thow</u> said that he tried both approaches (and perhaps fell between two stools in doing so!) As with all dilemmas, one adopted the best solution to the problem at the time it presented itself.

TYPES OF PROGRAMMING

PRODUCT DEVELOPMENT

System Control Programs	Schedulers
	Dispatchers
	Access Methods
	Utilities
Program Products	Assemblers
	Compilers
General Application Packages	Banking
	Insurance
	Manufacturing

LANGUAGE

Development

Control

Standardisation

ADVANCED TECHNOLOGY

Investigate New Techniques

Model New Language

SYSTEM SUPPORT

Technical Support to Sales Force

After-Sales Service

Figure 1

PROGRAMMING FUNCTIONS

Product Development

Product Programming (Maintenance)

Market Planning

Mission Planning

Test Development

Computer Services

Systems Integration and Release

Programming Tools

Publication Writing

Systems Management

Product Measurement and Analysis

Figure 2

LIST OF AVAILABLE COURSES

Courses Specifically for Programmers

The ELIPT Handbook Page 11

Figure 3

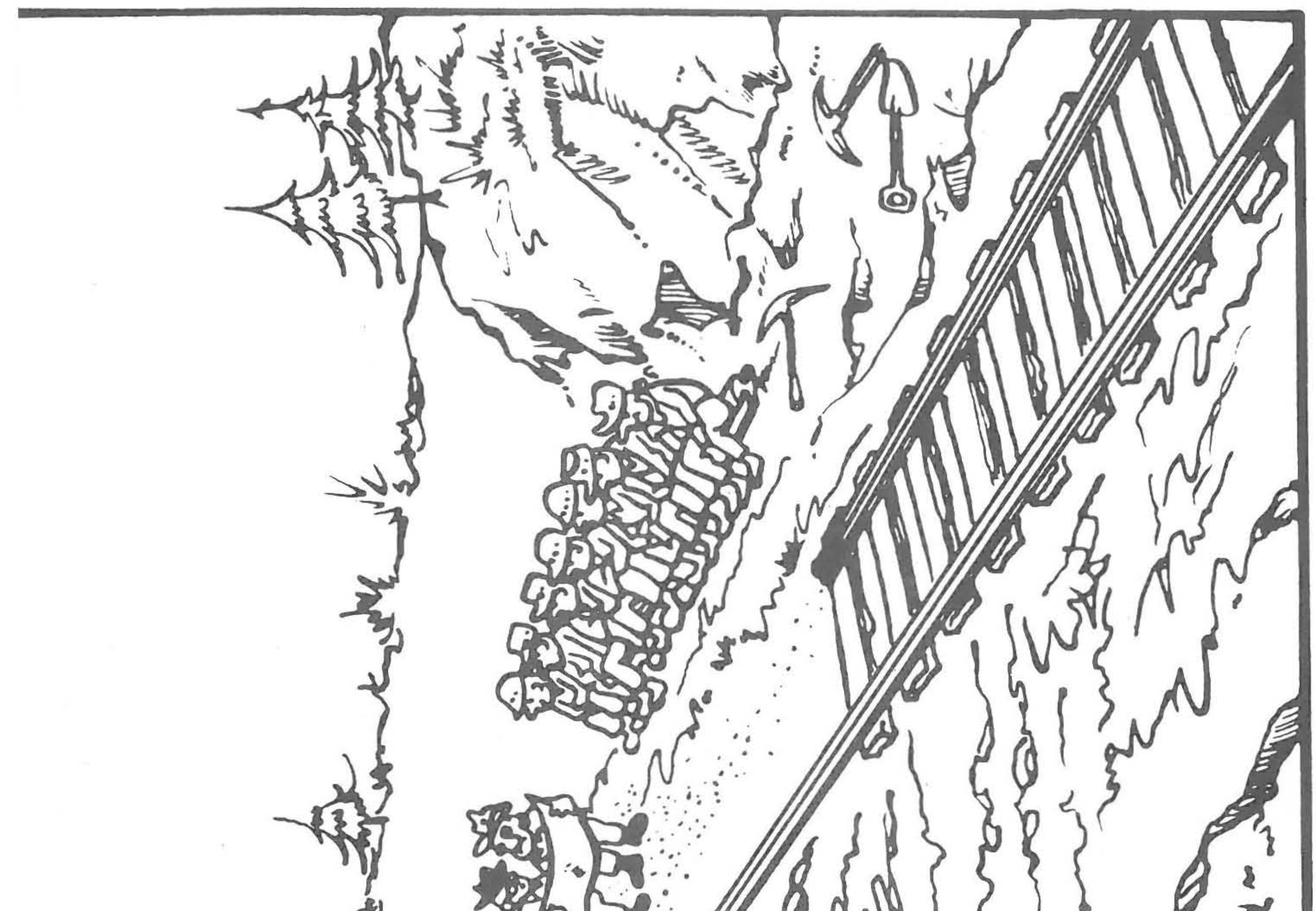

Figure 4

INFORMATION OUTPUT

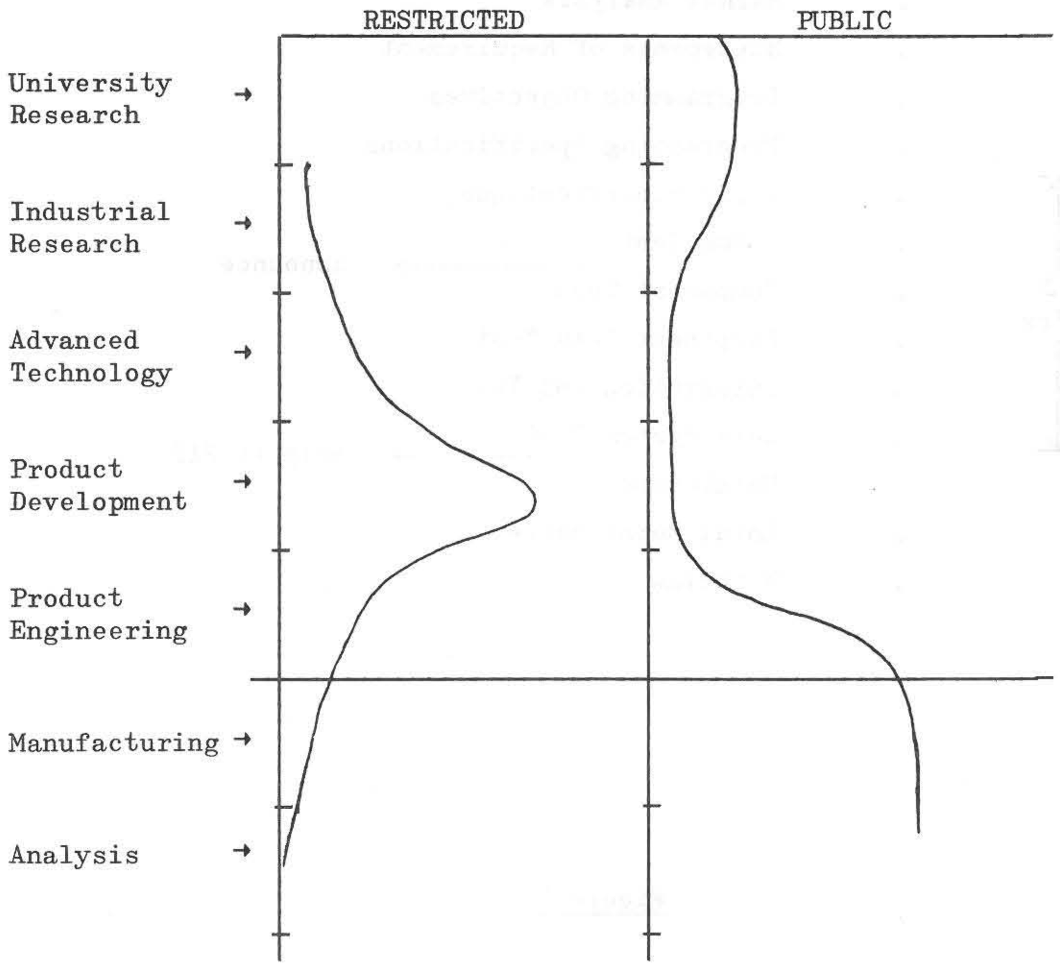

Figure 5

THE DEVELOPMENT PROCESS

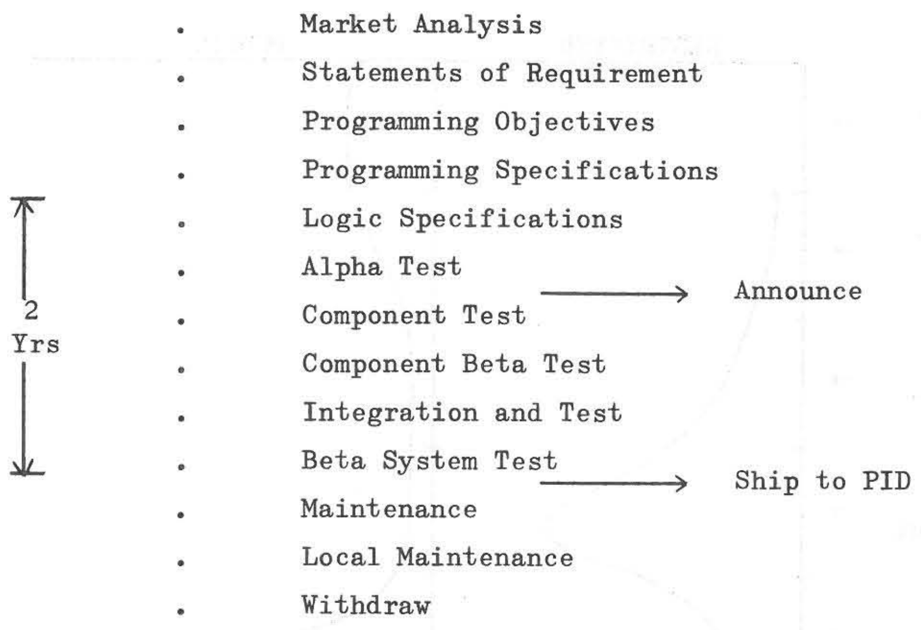

Figure 6

The Programming Development Process

Phases and Functions

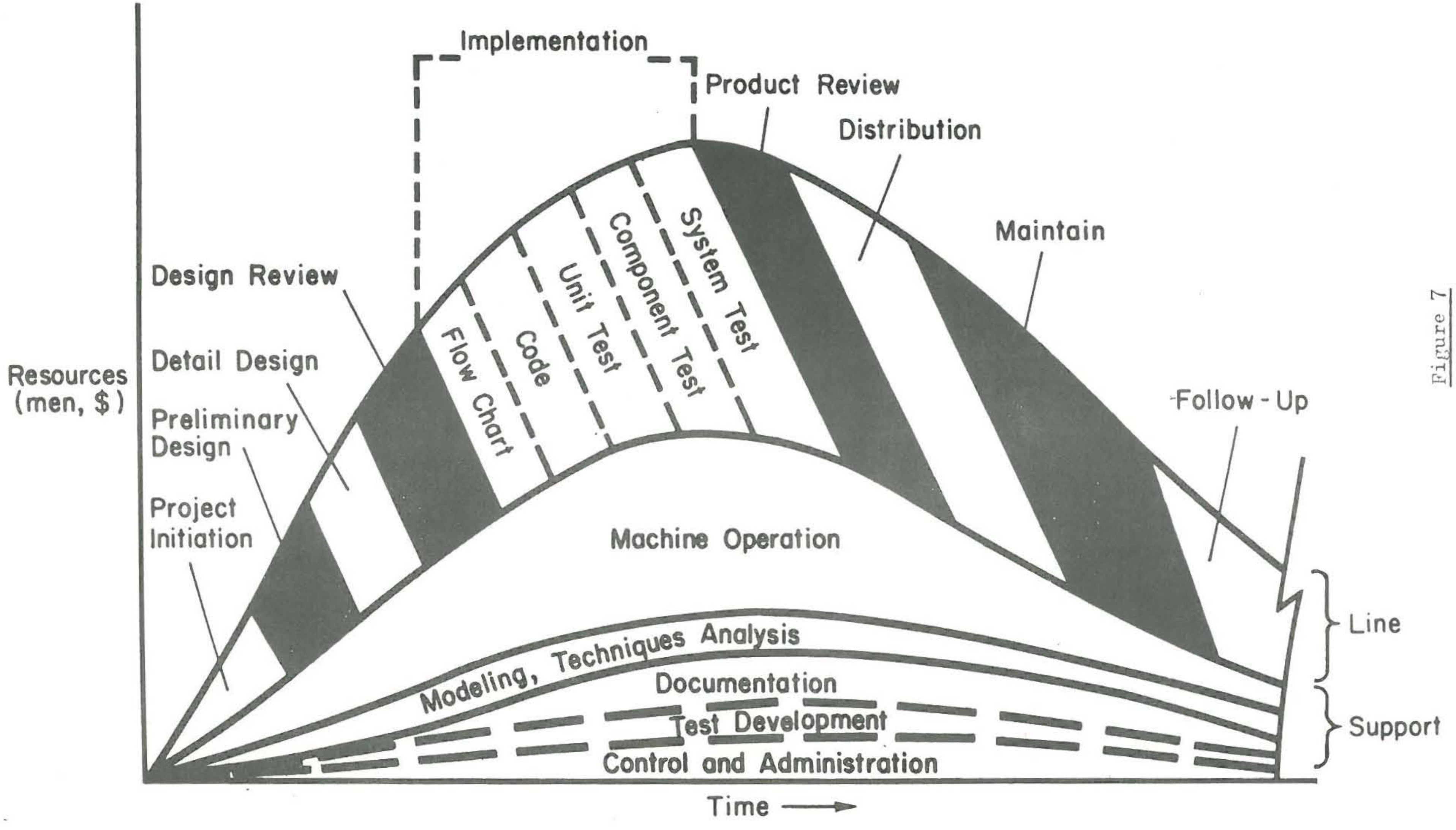

Figure 7

PHASES

Phase No.	Title	End Point
0	Planning	Entry 2-Year Plan
I	Architecture	Approved Program Objectives and Initial Program Functional Specifications
II	Specification	Approved Final Programming Functional Specifications
III	Design/Implementation	Alpha Test Complete
IV-V	Implementation/ Integration	Beta Test Complete
VI	Maintenance	Transfer to Local Maint.

Figure 8

SUMMARY OF REQUIREMENTS

Summary Description	0 Initial	I Update	II Update	III Update	IV–V Update
Justification	Estimate	Update	Group Forecast	Group Forecast	
System Plan		Comp and Subcomp Level	Detail Level	Update	
Programming Objectives	Prelim.	Approved	Revisions		Analysis
Program Functional Specs.		Approve Initial	Approve Final	Revision	Analysis
Program Logic Specs.				Complete	Revisions
Documentation Plan			Alpha Plan	Beta Plan	Analysis
Publiclations Plan			Alpha Plan	Beta Plan	Analysis
Program Publications				Alpha	Beta
Integration and Test Plan			Prelim.	Complete	Analysis
Distribution and Maintenance Plan			Prelim.	Approve	
Schedules					Analysis
- Firm	I	II	III	IV - V	
- Tentative	II - V	III - V	IV - V	VI	
Cost Estimate	Informal	Formal	Formal	Formal	Analysis
- Firm	I	II	III	IV - V	VI
- Tentative	2-Year	III - IV	IV - VI	VI	
Hardware Obj/Spec.		(Identify Technical Problem)			
Market Review		Obj.	Spec.		
Cost to Date		I	I - II	I - III	I - IV
Phase Review and Proj. Summary	Entry into 2-Year Plan	Approved Obj. and Initial Spec.	Approved Final Spec.	Alpha Test Complete	Beta Test Complete

Figure 9