RM-75-50

SOFTWARE SYSTEMS VIEWED AS AN ANALOGY TO INDUSTRIAL ORGANIZATIONS

Wm. Orchard-Hays

September 1975

Software Systems Viewed as an Analogy to Industrial Organizations

Wm. Orchard-Hays

Abstract

This paper is not "scientific" in any usual sense. Rather, software systems are described by means of analogies with large industrial and other organizations. The curious nature of software is first pointed out, and then its major dimensions are listed. Typical attributes of a large organization and its functions are briefly set forth and then these abstractions are related to systems of programs. The dual nature of an organization and its technology is suggested and then applied to systems of programs and the data structures on which they operate. The role of the user is discussed, several aspects being shown. Finally, a few maxims for building, maintaining, and using software systems are given.

"An idea, like a ghost, ..., must be spoken
to a little before it will explain itself."

- Dickens

Foreword

The use of computers is entirely dependent on software. A computer system itself--that is, the hardware--is a kind of miniature world of a special sort in which a particular class of activities occur. These activities are essentially the transformation of data and their transmission, i.e. shipment and handling. We need not get into abstract discussion of what data are, what distinctions may exist between data and information, or how data are represented and recorded. Neither a semanticist's nor an electronic engineer's viewpoint is to our purpose. Nor is there any need to be awed by the computer's prodigious feats of arithmetic and data processing--these are what computers are designed to do. Rather, given these kinds of capabilities, the theme is how to organize our thinking and our approaches to programming in order to make best use of them.

Programmers have, in fact, made good use of computers from the beginning, though it must be admitted that excellence has not been universal or consistent. But as computers have improved in speed, capacity, reliability and standardization--and computer applications have grown in scope, complexity and importance--software has become more massive, intricate and overwhelming. Most analysts and researchers, and even many application programmers, make no pretense of understanding software systems in any but the most superficial way. While it is true that it takes years of experience--and perhaps a special sort of aptitude--to become an expert system programmer and designer, it must be possible for the user to understand in some effective way the nature of the system he utilizes. This is particularly true of interactive systems which hold so much promise for man-machine interplay in attacking important planning and control problems.

This writer has long sought for a useful analogy for a software system both to organize his own thinking and to give meaningful explanations to others. As early as 1959, he attempted to characterize software in terms of a management structure but the idea was premature and incomplete and fell flat. However, continuing experience and reflection, plus the elaborate and seasoned software in being, make it appear that the idea may now be mature and valuable enough to set forth in some detail. It is hoped that the scope of the analogies drawn will not offend the reader's intellect or sensibilities. There is no intent to construe anthropomorphisms but it will be necessary to see similarities in structure between human and abstract organizations--both of which are human inventions.

"Allegories are fine ornaments and good illustrations, but not proof."

- Luther

Why an Analogy?

The question may arise why an analogy should be used as the main basis for discussing a subject and not merely to illustrate. The answer is twofold: the curious nature of software and the lack of fundamental principles in a scientific sense. The field of management shares the latter weakness but, since it has a long history both as a subject for study and as a practical arena of activity, it has principles and guidelines which are widely accepted and proven by experience. Hence it makes an excellent type for an analogy if it fits the antitype. The thesis of this discussion is that it does.

Software is curious mainly because it is an active agent and yet one cannot point at it or any physical representation of it--such as regarding a generator as electrical energy. It is even hard to pinpoint where computer programs exist. Anyone using a computing facility is familiar with huge listings of assembled or compiled programs, tabulations of data, and run submission forms, card decks, etc. But these are more in the nature of delivered products or order forms. When a programmer receives an assembly listing, it is like receiving delivery of some item or subassembly which he previously ordered and which was produced by another software complex. The actual program which he assembled resides on some magnetic device in the hardware complex but it is only stored there. When he causes it to be executed, we say it is "in the computer" but one would be hard pressed to identify its particular electrical impulses. Anyway, the electrical impulses are not the program either, any more than neuro-muscular impulses are people. A program carries out a specific piece of work for some purpose. But even this "work" cannot be interpreted in the sense of physics since it costs neither more nor less to have the computer on whether any program is executing or not, except for printed output. (We ignore the inertial aspects of such devices as card readers and tape drives.)

A program has an author but it is not like a book nor a system like a library. This is why copyright and trademark laws have proven inadequate for software. Software carries out useful, often complex and sometimes novel procedures, but it is not a machine. Hence patent laws are inadequate. Programs have a personality, endowed by the programmers and analysts, but this persists and may be duplicated long after the programmer finished checking it out. Indeed, the programmer may no longer be alive. Something similar is true of books, photographs and recordings but these do not continue to carry out actual tasks or to work dynamically with similar items with different origins. All in all, it seems fair to say that software is a rare, if not unique, product of human ingenuity.

"The knowledge we have acquired ought not to resemble a great shop without order, and without inventory; we ought to know what we possess and be able to make it serve us in our need."

- Leibnitz

The Major Dimensions of Software Systems

The terms software and systems are subject to a variety of definitions and scopes of meaning. There are dangers in construing them either too broadly or too narrowly but perhaps no firm limits are possible. Software is sometimes understood to include manuals, procedures, forms, methods, and all the rules and regulations common in large computing centers. While this may seem much too broad for a discussion of actual systems of programs, some aspects of each of these items must be taken into account. Good manuals, for example, are most assuredly a necessary part of the delivery of any software system, yet one might tend to exclude them from a discussion of the organization of the actual programs. However, one of

the manuals, or a part of it, will contain just such a discussion and this information will exist nowhere else. Its status is entirely analogous to the status of organization charts, job descriptions and procedure manuals in a company. These are a part of the company even though the company itself is their subject. More importantly, the functioning of the company is partly dependent on these documents, the more so the more rigid is company discipline. This is perhaps as good a place as any to point out the biggest weakness in our analogy: the discipline of software executing in a computer is virtually perfect. However, it is the user, not the computer, who reads the manuals.

In any event, a software system has several major components or dimensions, including the following:

- The Hierarchy of Routines; their purposes and responsibilities.
- Categories of Data Sets; their relationships and access methods.
- User Controls; man-machine interfacing and activation.
- Documentation and User Training.
- Execution Controls; storage allocation and operating system.
- System Maintenance and Extension; programming languages, system integrity, program libraries, dissemination.
- Testing and Experimentation.

The first four of these will take up most of the sequel. The other three, though essential, are less important to the user except for general concepts. However, the appropriate analogies will be indicated.

"Good order is the foundation of all good things."

- Burke

Hierarchical Organization

It is hard to say whether organization precedes or follows the growth of an enterprise. This is a chicken-and-egg question. In reality, they grow and evolve together but it seems apparent that some concept of an organization must exist before it can come into being. The most elementary undertaking by two or more people almost invariably has a boss, if not explicitly then tacitly or de facto. In contemplating more extensive operations, a leader will mentally organize the effort, either from instinct or experience, or he will not remain the leader. It seems unnecessary to belabor the point.

An organization of humans nearly always takes the form of a hierarchy, that is, a pyramid or inverted tree. Many variations exist but these fall mainly into two types: more or fewer branches from one node, and the size and number of auxiliary branches. This has led to innumerable discourses on "span of authority" and relative importance of "line and staff." There would be no point in entering into such controversies here; we merely accept such concepts as valid principles drawn from extensive experience. However, the notion of centralization and decentralization will receive special comments.

Software from the beginning appeared with an embryonic hierarchical structure, although it started in the middle and grew both up and down. One of the first little gems of the programming art was to devise a method for one routine to "call"--i.e. to command execution by--another routine. It was some years before computer architecture made this relatively automatic so software preceded hardware in establishing rank. In fact, except for I/O operations and later a class of privileged instructions, computers still treat everything with impassive equality.

At the top of an hierarchical pyramid there is always a big boss--president, chairman, commander-in-chief, pope, or whatever. His authority is seldom absolute in a general sense but the chief prerogatives are his and the office commands respect almost irrespective of the incumbent. This office has a number of functions attached directly to it, before one gets down to the next level of command. Their avowed purposes, in addition to necessary services to the chief executive, is to give the organization consistent direction, policy and administration. The heads of these functions do not constitute a chain of command, which is a decentralizing force, but the senior staff level which fosters centralization. Their actual authority depends on a number of things but, usually, it is indirect. There are other ways to enforce centralization, or more properly, standardization, which does involve direct command. For example, Henry Ford was willing, even anxious, to decentralize many operations but he was insistent on a central foundry. There were good reasons: it is a separable, largely self-contained activity requiring highly developed techniques and enormous capital investment and its output impacts the quality of the entire production of the corporation. However, running the foundry is not a staff position even if its manager reports directly to the top and has no subsidiary divisions under him.

The chief executive, once appointed, is usually not left to his own devices with no reporting function. (When this happens it almost always leads ultimately to disaster, the delay being proportional to the capability of the executive.) First, in modern terms, there is a board of directors to whom the chief executive is responsible. Sometimes the board is mostly a rubber stamp but, second, there will also be a senior executive council or some inner circle with whom the chief executive must deal regularly and whose wishes and opinions he must respect. Thirdly, the past history of the organization will have built up precedents, philosophies, principles and

methods which limit the freedom of the chief executive. He is himself a product of this history.

As one goes down to the next level of command, say the operating or group vice-presidents, these executives find themselves in a similar position with the senior staff playing the role of the board of directors, their own lieutenants playing the role of the inner circle, and the same organizational history constraining their actions. Additionally, of course, they must also report to the chief executive and follow his general directions.

A principle of organization which seems to be universally valid is that each element should have the same basic structure, even though specialized in function, and that larger aggregations should be similarly formed from smaller aggregations. This is really what a hierarchical organization chart depicts. It is also the way living things grow. It seems to be what gives cohesion, integrity and identity to any complex structure.

Be that as it may, an organization can be extended downward several levels with the same general structure, provided each echelon need only report directly to its immediate superior. Also, two or more entire organizations can, provided they are of similar structure and philosophy, be brought together to form a larger organization by adding a super-executive cap and combining, paring and slightly realigning certain functions of the prior chief executive offices. At least this seems true in principle; in practice, it is often traumatic and less than successful.

Similarly, a branch from one organization may, in principle, be cut off and grafted into another organization, as when a corporation sells a division. The same problems may arise here as in combining two organizations into a larger one.

Some Software Analogies

The chief executive of a software system is the operating monitor or executive routine. This is often called the

"control program" which is an understatement giving the effect of an overstatement. (Builders of operating systems are noted for their arrogance.) Its board of directors (it is hard not to say "his") are real human beings. We must be careful to draw our analogies meaningfully. Operating systems are normally provided by the computer manufacturer and of course there are executives and technicians responsible for this function of the manufacturer's business. From the present point of view, however, it is the manager and technicians of the computing center who play the role of the board. They decide what options and features of the operating system (and also hardware) available from the manufacturer (analogous to current technology) will be activated, specialized and possibly modified in the particular installation. Presumably they are guided by the purposes and requirements of the installation's users, i.e. customers. The analogy cannot be pushed too far here since we are moving from human to abstract organizations.

The operating monitor has a large and powerful staff and certain important line functions directly under its control (strong centralization). Chief among the latter is the I/O monitor which is charged with carrying out all actual data transmission operations (shipping and handling). The authority of the I/O monitor is virtually absolute in this function and all transmission, including that for the operating monitor, must conform to its regulations. Lower echelons usually have "departments" specifically organized to deal with the I/O monitor and, in turn, enforce their regulations on their peers and subordinates. There are many reasons for this strict discipline--some technical, some historical and some to protect the proprietary interests of the computer manufacturer. It is as though the transportation industry were run by an army under a powerful general. By and large it runs well and reliably, but not too efficiently and certainly not considerately.

The operating monitor also has a central accounting department, which receives detailed invoices from the I/O

monitor and other sources, and a central planning office which schedules all operations both in a gross sense and in resolving immediate conflicts for capacity. It will accept priority designations in addition to applying its own elaborate rules. Its principle guideline is to optimize the utilization of equipment without too greatly impeding the carrying out of production in a timely fashion. Although an attempt is made to meet demand, it can be heavy handed in granting authorizations for the use of facilities. The operating monitor can be tedious in its processing of orders and maddening in its disposition of discovered errors. It displays many of the attributes of a bureaucracy in a planned economy or the management of a monopoly.

All orders for production are funnelled through the chief executive's office. In most systems these orders must include commitments or good estimates for the facilities required: main and auxiliary storage, and amount of central processor time. Many systems also require complete specification of the source and nature of input data (special raw materials), though some may be included with the order (job deck). The complete request is reviewed carefully for correctness and consistency and, if not in order, it is rejected. A charge is made for this review. The job is not authorized until all is in order and all required facilities and input are available.

Once the job is authorized, it must be scheduled. There are two parts to this. The initial scheduling is not done until necessary facilities are free and only then is the job initiated. During execution of the job, however, there may be insufficient processor time or transmission capacity for all active jobs. Resources are then allocated piecemeal, giving note to any priorities. Thus production delays can occur even after a job is initiated.

When a job is initiated, control (i.e. authorization to proceed) is given to its main routine. This routine can be

regarded as an executive directly subordinate to the chief executive. Operating monitors can, and do, supervise a great many such subordinates over time, far more than the span of authority of any human would permit. However, we must take into account another flaw in our analogy here. At any one time, the operating monitor has only a limited number of executive routines under it, say seven or eight. Each job specifies what executive routine it requires which may be any one of many in storage. Hence the active organization actually changes as each job is initiated and terminated.

Depending on the nature of the job, its subhierarchy of routines may be very simple (conceivably only one routine plus a canned package of I/O routines) or very elaborate. An example of the latter is a Mathematical Programming System (MPS) which may have a structure rivalling the operating system. A number of elaborate structures are also standard items of the software system, such as compilers, linkage editors, and sort-merge programs. However, these are treated no differently from any other application subsystem, such as an MPS.

Henceforth, we will use an MPS as an example of a subsystem. Typically, its top routine, often called EXECUTOR, fills a similar role with respect to math programming jobs as the operating monitor plays to all jobs. However, there is one important difference: the human user, or customer, interacts with EXECUTOR much more intimately than with the operating monitor. This becomes particularly true with an interactive system. The user does submit job decks to the operating monitor (or logs in and initiates subsystems with an interactive setup) but this is very stereotyped and formalized. With batch operations, it is mainly a confounded nuisance to the user and he may even relegate the details to an aide. But if he is interested in math programming jobs, he gains some virtuosity in communicating with EXECUTOR, or he should. More will be said later on the role of the user.

"It is much easier to design than to perform. A man proposes his schemes in a state of abstraction..., and is in the same state with him that teaches upon land the art of navigation, to whom the sea is always smooth, and the wind always prosperous."

- Johnson

What Is Really Going On?

There are those who believe management is an art (some might even claim a science) which has an existence of its own, and can be learned and then applied to any kind of enterprise. One could point to some monumental failures to refute this, but all that has been said so far about organizations would seem to confirm it. Apparently one can draw a blank organization chart, with some provision for more or fewer boxes, and then fill in the appropriate titles for most any organization, including abstract ones. Such charts depict lines of authority and responsibility and indicate ranks but the trouble with them is that they never show what is going on, what all these people are about.

Let us set aside such organizations as government, military or church, for which we would have to pile metaphor on analogy. In industrial and commercial organizations, what is going on is a series of transformations of some kind of structured aggregations--whether it is converting steel, glass, etc. to automobiles, converting fuel to electrical energy, converting goods to money, or whatever. It is these transformations which are the raison d'etre of the organization. On the other hand, the transformations will not occur without the system. There is a reflexive nature to organized activities.

In the case of a software system, what is going on is the transformation of data; there is no other function which a computer can perform. Of course, we attribute all sorts

of meanings to the different forms and aggregations of data but that occurs only in our minds. A software system is an abstract organization which carries out transformations of data which are deemed to be of some purpose and meaning by its users, just as an industrial organization transforms raw materials to products which are deemed to be of value by society.

The transformations which an organization carries out are somehow in a different dimension or plane than is the organizational hierarchy. It is difficult to view them both at once. If one goes into the executive offices of, say, a steel company and talks with the people there, he gets one impression of the operation. If he then takes a tour of the mills, foundries and yards, he gets a completely different impression. It is likewise difficult to consider both a linkage and control chart for a system of routines and a flow diagram of the data on which they operate. One needs three dimensions to show all the paths. Even then, other aspects of the total system must be neglected. To try to project these onto one plane is only confusing.

Generally speaking, a chain of command such as depicted by the echelons of an organization chart is a decentralizing force. The vice-president does not do exactly what the president said and the general manager does some things which he does not tell the vice-president. The farther one gets from the source of a general order, the less precise its execution becomes. But then no one wants an organization of robots except on a parade ground.

On the other hand, the transformations of structured aggregations are a strong centralizing force, assuming normal incentives exist. It is almost a truism that the first design of a process is too elaborate and cumbersome. It is by experience and continual refinement that methods are perfected. They then become building blocks for more elaborate processes and thus technology grows. It is almost impossible for a

newcomer, however well backed, to break into a seasoned industry, as witness Henry Kaiser's bid in the auto industry.

It is an error to regard centralization and decentralization as antonyms. They bear a relationship more akin to duality. So do an organization and the technology or business in which it is engaged. The same is true with software. The hierarchy of routines should be capable of extension, modification and innovation, within limits, but the transformations of data structures should become more refined and standardized.

The Front Organization

Neither the formal organization nor the technical operations are what most outsiders see in dealing with a company. In a department store, one deals with clerks and cashiers; an airline passenger deals with reservation agents, gate agents and stewardesses, with occasional glimpses of the pilot in his PR role. In ordering equipment, one deals with salesmen and technical representatives. Furthermore, internal management deals mainly with records, reports, studies, etc., rather than with actual physical things. Apart from our own specialities and private lives, the world we deal with is largely one of paper, numbers and brief, impersonal conversations.

These front organizations we deal with are not something separate from or superimposed on the real enterprise. They are the projections of those parts whose function it is to carry out activities with exogenous attributes. The ticket agent has a spot on the organization chart and the pilot really flies the airplane. The production reports which the general manager reads are summaries of real work done by real machines and real people, and most of the records would be produced whether the manager reads them or not. All these things are merely our perception of the normal activities of the work-a-day world. Of course, they are often embellished to make them more attractive and convenient but this is just a special case of technical improvement.

The situation is really no different with a well-designed software system. The system cannot just sit there and run with no outside contact. Also, the internal record-keeping is voluminous. If the system is well built, it is even possible to get summaries and reports for special purposes. Unfortunately, the external projections have not yet, in many cases, been made as attractive and convenient as they could be. For example, the IBM OS/360 operating system is very elaborate and powerful and has a wealth of capabilities. Its job control language (JCL), however, is particularly ugly and difficult to read and write. The result is that many people think the system is very bad, and in a sense it is for this reason alone.

Every industry or other broad area has its own jargon. Some of it is very technical and understood only by inside experts. Part of it, however, rises to the surface and becomes public property. Every seasoned air traveller knows what a holding pattern is and can tell a 747 from a DC-8. With respect to computing, everyone now recognizes an IBM card and most people have some concept of what computerized accounting means (probably an incorrect one). However, if one is to actually utilize a computer, he must be familiar with the jargon at a deeper level. A computer user is more like an industrial customer. A bridge builder ordering steel must be familiar in depth with a good deal of terminology from the steel industry.

Use of a computer, that is, of a software system, has one added conceptual difficulty not common to many other arenas of activity: it is hard to distinguish the product from the records. When the programmer gets his assembly listing (the product), there is another page or two giving the files accessed, the CPU seconds used, the number of I/O operations, the total charge, etc. All of the material he receives is chicken-tracks on paper. Also, the assembly listing itself is not <u>really</u> the product, which is a routine

stored somewhere, but just a printed representation of it. One is forced to think abstractly and to make mental classifications.

A related difficulty is that the user plays many roles. When he is assembling routines, he is building software. When he submits production runs, he is using software. He may do both plus other functions on one submission or terminal session. In fact, in all the parts, he is also using existing software, even when he is building more. It can be a complicated game.

The Role of the User

The purpose of this entire discussion, of course, is to try to give the user a better view of his role in achieving meaningful results with a computer. As already suggested above, a user does not necessarily play a single role and, of course, there is a wide variety of users. However, from an organizational viewpoint, something approaching a single role can be defined.

First and foremost, a user is a customer, but he is a customer with rather unusual prerogatives. He supplies his own raw materials, in prescribed forms, or on occasion may specify input from available sources either public or authorized. He orders standard processes to be applied to his input but the results are uniquely his. If no standard processes are appropriate, he can create his own, using other standard processes, and have them installed either temporarily or permanently for his use or the use of others. Furthermore, and particularly with interactive systems, he can personally enter into the higher level decision making and sequencing activities in carrying out his production. Perhaps the nearest analogy is that of the government's role in a large research and development contract.

The user may also store material in a suitable place and use it whenever he wishes. It is never used up but continues

to replicate itself as necessary. If he is through with certain material, he must explicitly destroy it. (This is another unique feature of data processing.) He may order certain finished products to be delivered.

Nevertheless, the user must not forget that he is, after all, a customer and that a whole complex organization is at work in filling his requests. In spite of his prerogatives, he must act within rigidly defined rules and regulations. To the extent that he may and chooses to enter into internal decision making, he must act as a part of the organization and not superior to it. His decisions and requests must make sense technically. In short, he must be both knowledgable and polite if he expects good results.

When the user has perfected a scheme of production, he can order production runs at any time. The system usually has facilities for automating this so that simplified order forms may be used. In effect, this production scheme becomes another standard process of the system though it may have security locks or command special charges.

Thus a user, in the most general sense, is part of a highly accelerated evolutionary process with extreme flexibility. This is accomplished with a basic structure which is very rigid, formal and highly centralized. More fundamental improvements come more slowly, of course. It takes years to make substantial improvements in the basic operating system or even in elaborate application systems like an MPS. Improvements in the underlying hardware may take place concurrently and asynchronously with quantum jumps every several years. But the user need not concern himself with these matters, except as he wishes to stay abreast of the state of the art. He has a highly useful and fascinating milieu in which to work now, provided he understands the facilities available and their organization.

Some Maxims for Building Software

We present here, rather dogmatically, some opinions about good design principles. Analogy will be used freely as appropriate.

- Segregate Activities Cleanly

In building software, and later extending it, it is important that each routine and procedure have a clearly defined function, and that this function be significant in the overall purpose of the package. This is easier said than done and has little relation to size nor necessarily to complexity. For example, a major procedure in an MPS is an elaborate primal simplex algorithm. A critical function is to select, from a set of candidate vectors, the one which will make the most improvement when substituted into the basis. This has, in turn, three main parts: calculation and/or validation of reduced costs, selection of pivot in each column necessary to maintain or improve feasibility, and selection of the best column based on priority sifting rules. For a number of technical reasons, combining these into one subroutine would be a mistake, primarily because the scanning schemes are entirely different and the middle routine is useful separately. However, this middle routine is itself very complex and quite long and at first glance might seem to be further decomposable. However, the code is highly integrated which is important to overall efficiency. Such observations can be made only after a great deal of experience but it is important to make an analysis on an operational basis. The fact that the middle subroutine is longer and more complex than some entire procedures is no argument against either. The engine department of the Chevrolet division of GM may well be larger and more complex than the entire Frigidaire division.

- Use the Concept of Responsibility to Segregate Functions

  When something goes badly wrong in a complicated process, the source of the trouble must be found and corrected. As soon as it is determined what exactly did go wrong, or which are possible, the first question is: Who is responsible for that? If the responsibility is divided or confused, it may be hard to track it down and even harder to correct it. The same is true when the function is to be changed. If the desired change is made in the apparently appropriate place and then it is overriden somewhere else, endless confusion can result. Any good manager would avoid this problem. A computer routine has definite responsibilities or else it should not exist.

- Do Not Cross Departmental Lines

  Any manager who found members of another department giving orders to his people would know something was wrong. A routine that has long, gangling tentacles reaching into other parts of the system with which it should not be concerned is a trouble maker.

- Put Responsibility at the Lowest Appropriate Level

  A chief executive who takes personal responsibility for matters properly handled by a project director is destroying his own organization. The greatest source of trouble in a system of routines is putting a function at the wrong level, usually too high. This often happens from patching the most accessible routine for some special purpose. A few years of such patching renders a system arthritic and incapable of further extension or modification.

- Delay Decisions Till the Latest Possible Time

  This might seem to violate the maxim to Plan Ahead. But there is a difference between planning and doing. A decision must not be delayed beyond the time it is required to determine the next action but it should not be made before all pertinent information is available. The counter-maxim is older: "The best-laid plans of mice and men oft times go awry."

- Complete A Set of Options Even if Not Now Required

  This rule must be applied with common sense, of course. But suppose there are two concurrent binary decisions and only three outcomes are defined. One should always consider the implications of the fourth possibility and what would happen if somehow that choice were actually made. (Rest assured it will be, if not during debugging, then on an important run.) Occasionally one finds a truly valuable function or insight with this policy.

- Do Not Design Yourself into a Corner

  Some old-time programmers (including this one) had a passion for using up every last bit in a data array. The evolution of whole technologies have been affected because there was no place to mark an additional case. It is like renting new office space with exactly enough room for all the desks now needed. Of course, the reader can think of less trivial examples of the same mistake.

- Remember that Overhead and Waste Motion Cost Money

  This appears somewhat counter to the prior admonition. A balance must be made between stinginess and wastefulness. But many analysts and programmers have a tendency to reduce a new

situation to a previously solved case. With the speed of modern computers this may often be justified, but not always. Everytime one branches to a subroutine, there is a nontrivial amount of overhead. If this is large in proportion to the amount of useful work the subroutine does and it must be done many times, perhaps the subroutine should be put in line. The situation is somewhat analogous to having to move work in process to another shop for an intermediate operation. A situation particularly to be avoided is the writing of intermediate data to an external file so some standard program can be used, say a sort-merge. If a big sort-merge is really required, fine. But if a small local sort routine, even if not too efficient, will do, it should be installed. Writing an external file, calling another main program and then reading back the results is like packing up unfinished items in special containers, shipping them a hundred miles to another plant, unpacking them, doing the operation and then sending them back the same way. It does not take very much of this to justify installing the necessary machinery locally.

Some Further Analogies

The full extent of a software system is much greater than has been indicated up to now. Much of it has not been of interest to the user in batch-mode operations, unless we include application system builders themselves as users. However, with application systems designed for experimental work on interactive computers, it will be necessary for users to have a deeper understanding of the total system.

In another dimension, there is an elaborate structure of execution controls and storage allocation. When a job is

initiated, it is much like assigning a factory building (main storage partition) and warehouse space (scratch files) to a division. Also, the necessary machinery (subhierarchy of programs) is identified and made available as well as the source of raw materials (input data). A number of standard facilities are also available which may be used as required subject to capacity limitations.

One of the tasks of an application system like an MPS is to further allocate these assigned resources according to the nature of the work to be done. It is as though it had its own Facilities and Maintenance department which is always setting up, tearing down and moving things around. This aspect of systems has been one of the most confusing to users and admittedly it is one of the more difficult aspects of system design. (One of the goals of virtual memory is to eliminate much of this problem by pretending that space is unlimited. Thus far, practice has not matched theory.)

We would be led too far afield if an attempt were made to describe loading and overlay mechanisms or the facilities of the operating system for assigning and relinquishing storage for temporary purposes. However, one can get some grasp of it by imagining a limited amount of floor space which must be used serially for a variety of processes. Each process needs room for machinery and often a large amount of material which flows in and out. Sometimes much of the material must be left in place while a whole new battery of machinery is hauled in. There are also the supervisors' and foremen's offices which must be left intact with their records throughout all this.

Another division of the entire enterprise (in fact one for each major application area) is concerned with the maintenance and improvement of the machinery itself and, to complete the analogy, one should also say the training of the operating crews. (A routine can be regarded as both the machinery and its crew for some specific kind of operation.)

This is like an engineering division. The human analysts and programmers have their own viewpoints and lingo which is usually reflected in their routines and subsystems. In finalizing their work, they also become users of the entire system. They are largely responsible for system integrity and have such tasks as maintaining and disseminating program libraries. They also have a heavy responsibility for documentation.

The final area we will comment on is testing and experimentation. This is like quality control and advanced design (often called that). Not enough work of this kind has been done in many application fields. Much of the current thrust in MPS development is in this area. The great difficulty, in addition to cost--a great deal of computer time can be consumed--is that very few people have the requisite breadth of knowledge. One must understand the application area with its methods, algorithms and context, and also systems with their hierarchies of routines, data set structures and dimensions of control. In addition, they must have imagination and be able to conduct well-designed, meaningful experiments. This is too much to ask of one person. It will be necessary to have teams of people who can work effectively in man-machine interplay. One of the members, at least, must have heavy experience in software systems but it will be a great advantage if all members have some meaningful grasp of their nature. Perhaps analogies, such as have been attempted here, may be helpful.