Planning a Computer Network: The Octopus Experience*

John G. Fletcher

Lawrence Livermore Laboratory, University of California

The author of this article is engaged in software design and implementation for the decade-old Octopus computer network. This network remains in a continual state of growth and change in a constant effort to take advantage of the most advanced hardware and software available from an ever-developing technology. Those planning a computer network or other complex computer system should benefit by considering the Octopus experience with regard to the needs of computer users, methods of design, network and operating system structure, security and privacy, the management of limited resources, and the advantages of locally generated hardware and software.

THE LAWRENCE LIVERMORE LABORATORY (LLL) is operated by the University of California; the major portion of the Laboratory's funding is derived under a contract with the United States Atomic Energy Commission. It carries out research and development programs in various areas of science and technology in response to changing national needs. For example, in recent months there has been an increased emphasis on matters relating to energy sources and energy utilization.

It was clear from the time of the founding of LLL in 1952 that its future would be closely tied to the future of the then young computer industry. The digital computer makes it possible to simulate the performance of many more experiments than can actually be performed because of limitations imposed by cost, time, technical feasibility, and political and legal considerations. Over the years LLL has striven to obtain and utilize computer equipment representing the very forefront of computer technology; the largest, the fastest, the most sophisticated. For example, LLL was the first to have the CDC 6600, the CDC 7600, the IBM 10^{12}-bit photo-digital store, and the Radiation, Inc., 30,000-line/minute printer. The first CDC Star-100 was recently delivered.

During the 1950s each LLL computer was operated independently in a batch-processing mode. It gradually became clear that this primitive approach to operation was inadequate, and about a decade ago the Octopus computer network[1] was created. Today Octopus joins all but a small fraction of LLL's computing resources into a single interconnected computing facility. It is a fully operational working system serving over 1000 users; it is not an untried design, nor is it merely an experimental project. Although few, if any, of the concepts used in the Octopus design are new or original, the network represents an integration of many advanced concepts, which is unique for its size and complexity. Therefore, the experience and insight we have gained from the Octopus experience, particularly with regard to the planning of a computer network, should be of general interest.

The planning of a computer network—or any other computer facility—should pass through four stages in proper sequence: first, identifying the needs of those who will use the facility; next, designing a system that will satisfy those needs; then, selecting the equipment required by the design; and finally, allocating funds. This sequence seems so obvious that it should not require mentioning. However, it appears to be commonplace that these stages occur out of order, even completely in reverse: an amount of money is set aside for the purpose of acquiring a computer capability; equipment is then bought in quantities determined by the funds available; decisions are made as to how the equipment will be used; and only when it is too late is it known whether or not the needs of the

*Work performed under the auspices of the U.S. Atomic Energy Commission.

user have been satisfied. Here we shall discuss the four stages in the correct sequence.

USER NEEDS

In considering the needs of the LLL employees who are the users of the Octopus network, several observations have been made. The most important is that such needs are varied and changing. Although most of the use of the network is for scientific computation, there is also use by administrative, clerical, financial, and library personnel. Even considering just the scientific activity, one finds the need not only for numerical calculation, but also for information retrieval, text editing, interactive display, and more. Furthermore, these varied needs are not static; they expand and change with time in ways that are often impossible to foresee, since the experience of using a facility frequently suggests new and more convenient patterns of activity. Therefore, the Octopus design is above all flexible, with as little as possible being assumed about the users' behavior. Octopus is a *computer utility*—a conglomerate of processing, storage, and communication resources that the user is allowed to request and then use as he sees fit (within limitations imposed by the system). In spite of the variety of LLL users' needs, certain nearly universal needs can be identified:

- Computer resources should be immediately and conveniently available. *Turn-around time*, the interval between the submission of a request for computer resources and the receipt of results, should be determined only by speed of operation of the computers; it should not be increased by inefficient operational techniques. For this reason, Octopus provides for *interactive time-sharing*. A user accesses the network primarily through a remote interactive terminal (e.g., a teletypewriter) located either in his own office or close by. If he has a card deck to input or a moderate amount of printed matter to be output, he can use a remote input/output facility consisting of a card reader and line printer located in his building, often on his floor. The operating system software of the various network computers permits operations relating to the activities of many users to be overlapped and performed in such rapid sequence that to a considerable extent each user can maintain the illusion that he has a computer to himself. This kind of organization permits the results of very small problems to be returned so quickly that to a human it appears nearly instantaneous.

- Computer resources should be universally accessible. Each remote interactive terminal should be capable of calling into play any of the network resources. It is this need that dictates that an interconnected network, rather than a number of independent computers, is required. With the network, a user at any terminal can converse with any of the large *worker computers* (e.g., CDC 7600's), which have the function of executing users' programs. The worker computers can in turn command the operation of any of the network's storage devices or input/output equipment. Without a network, there would have to be a separate class of remote terminals for each worker computer, which would no doubt require the purchase of a larger number of terminals. Also, unique and expensive equipment such as the 10^{12}-bit store and the 30,000 line/minute printer would not be directly available (*on-line*) to all computers but would have to be used indirectly (*off-line*) by manual transport of magnetic tapes or similar media, resulting in reduced efficiency and increased time delays.

- Complete programming freedom is required. The user should be able to write his program in whatever programming language he finds most suitable. At LLL, the principal language is a form of FORTRAN called LRLTRAN,[2] but a great variety of languages is available—including assembly language (for each kind of computer), COBOL, ALGOL, APL, SNOBOL, and LISP. Such freedom, coupled with security requirements (discussed below), requires that worker computer hardware provide two modes: a privileged one for the execution of the operating system and an unprivileged one for the users' programs. Programs executing in the unprivileged mode are interrupted after a predetermined time, they are limited as to the portion of the main memory that they can access, and they cannot directly access secondary storage or input/output equipment; such access is performed by making requests of the operating system.

- The computer system must retain information for extended periods, in some cases for years. That is, there must be a central data base maintained by the network that is directly accessible to all worker computers. At LLL the primary facility for storing this data base is the 10^{12}-bit store. Without a central data base, one is faced with the inefficiency and delay of maintaining multiple copies and of transporting information manually.

- Interference by one user with the activities of another cannot be tolerated (whether it arises from malicious intent or, as is usually the case, from error). No user should be able to induce a malfunction of the system. No user should be able to view or alter another user's private information. No user should be able to usurp an unfair portion of the system resources. That is, the system should provide *security*, *privacy*, and *fairness*.

- Finally, at least at LLL, users demand very extensive and very up-to-date facilities. They continually modify and improve programs, with the result that they require more processing time and other resources; therefore, LLL needs several of the largest and fastest computers available today. Users also request that every convenience of which they hear or read be made available to them; a current trend is toward the introduction of devices permitting rapid display of text and pictures. The major components of the current Octopus inventory are summarized in Table 1.

NETWORK DESIGN

In designing and implementing the Octopus network, LLL has made use of whatever good ideas could be found in the computing literature and other external sources. However, much of the effort has had to rely on trial and error, guided by the good sense of LLL's own staff, for Octopus for the most part has tread and is treading new ground. Universities, which are an excellent source of good ideas (as well as some bad ones), are hampered in their efforts to fully implement those ideas by a lack of funds. Commercial users of computers generally have been reluctant to create their own computer system designs and have relied on computer manufacturers and computer software firms. The latter two groups seem very slow to innovate, possibly because of the difficulties of accommodating radical changes to their existing customers and of convincing new customers of the effectiveness of an untried concept. It is for these reasons that Octopus is unique.

In judging the applicability of Octopus design concepts to their needs, others must first consider whether their needs are the same as the needs of LLL users (cited above). They should also note that Octopus is geographically compact and uses its own data transmission lines; the problems of dealing with the commercial telephone network are absent. Furthermore, Octopus is under a single administration, which avoids a number of problems of a political nature. Nevertheless, the LLL situation probably resembles that of many medium or large corporations or governmental bodies.

A computer system should be designed *defensively*. This is the most important principle used at LLL. Defensive design means that each part of the system will (in so far as possible) recover from the results of anomalous events. In particular, no malfunction or error in one component of the network should induce a malfunction in other components of the network. The same rule should apply to a considerable extent to different program modules within a single computer, especially when they are written by different programmers.

One aspect of the defensive design of Octopus is that each computer in the network generally has only a single function. Thus, the worker computers carry out only those activities immediately necessary to the execution of users' programs. Other computers in the network are classified as *concentrators*. Each concentrator is the center of a subnetwork that carries out a single function in support of the worker computers. The concentrator is connected to each worker computer and to whatever terminals, input/output devices, or storage media are appropriate to the function of the subnetwork, as shown in Figure 1.

The major subnetworks currently making up Octopus are summarized in Table 2. Figure 2 indicates how the entire Octopus network is formed by a superposition of its subnetworks. A failure in any concentrator can deny the network the capability provided by the corresponding subnetwork, but it will not significantly affect other capabilities. Interconnections between subnetworks (not shown in Figure 2) provide alternate routes for information so that in many cases a failure of a connecting link

Table 1. Octopus Hardware Inventory.

Function	Number	Equipment
Computer	4	CDC 7600
	1	CDC 6600
	2	DEC PDP-10
	~40	minicomputers
Storage	1	10^{12}-bit photodigital store
	1	Data Cell
	~20	disks
	~20	disk pack drives
	~20	magnetic tape transports
Interactive Terminal	~600	teletypes
	~20	30 character/sec hardcopy units
	~5	alphameric softcopy units
	2	LDS-1 high-performance displays
Output Display	128	television monitors
Output Hardcopy	1	30,000 line/minute printer
	2	FR80 microfilm recorders
	~30	300 to 1000 line/minute printers
Card Input	~30	card readers
Miscellaneous i/o	—	card punches, paper tape, DECtape, etc.

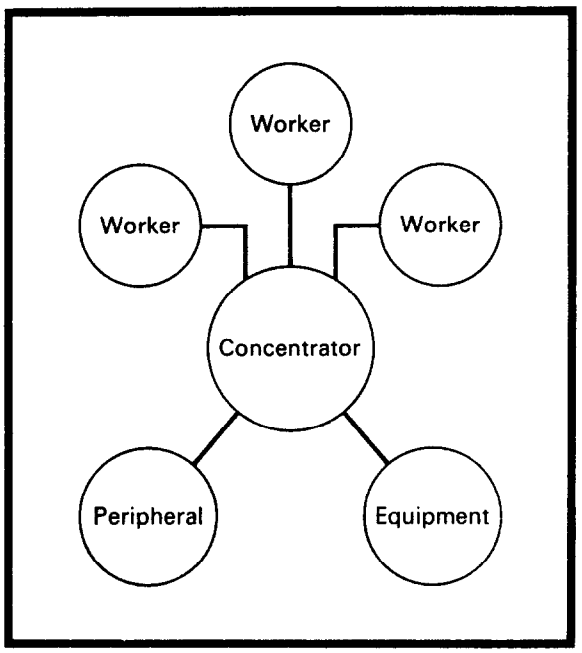

Figure 1. Typical Octopus Subnetwork.

will in no way degrade network capability. A highly centralized network (such as the name Octopus might suggest) with a single 'head' directing traffic among its 'tentacles' is highly vulnerable to failures in the head, and the frequency of failures in the head is aggravated by each addition to the network, since it requires a change in the head. The actual Octopus structure permits the graceful replacement of obsolete facilities and introduction of additional facilities; the temporary disruptions caused by such changes affect only a small part of the entire network.

Another aspect of the defensive design of Octopus is that the *system programs* in the worker computers (those that execute in the privileged mode) are kept as limited as possible; as much activity as possible is carried out in the unprivileged mode. The system programs perform those functions, and only those functions, that present a threat to the security of the system or to the privacy of the users; such programs include those for accessing the data base, routing messages, allocating and charging for resources, and performing input/output. Functions not performed by system programs include, in addition to applied computation, compiling (translating programs written in FORTRAN, COBOL, or other computer languages into machine instructions), editing textual information, searching the data base to find desired records, and altering or debugging defective programs.

Limiting the system programs has the additional advantage that the programming load may be distributed over a wide base. Only those changes and additions of a very fundamental nature affect the system programs and therefore need be made by the highly trusted and highly skilled group of system programmers. (At LLL this group numbers only about 20 persons, even though the entire network has been designed and implemented by LLL employees with no use being made of manufacturers' software.) All other changes and additions are made by programmers associated with the group desiring the change. One can imagine a hierarchical structure of non-system programmers. At the highest level are those who write the subroutines, compilers, and utilities that are used universally. Programmers at lower levels use the routines generated by those at higher levels as building blocks in producing specialized, and perhaps very sophisticated subsystems for use in particular ranges of applications. A very complex program, such as a compiler, can be written by a programmer at any level of the hierarchy, depending upon the range of users who expect to use it. (Several users at LLL, in fact, have written their own personal compilers.)

Defensive design means not only that the system will survive after a malfunction, but also that malfunctions are detected and corrected. The system must collect records of its activity, particularly of anomalous events. Very serious anomalies must be reported immediately to operating personnel. It should be possible to debug and trouble-shoot all but the grossest software and hardware failures while the computer involved continues to run.

SECURITY AND PRIVACY

The security of a computer system may be subverted by unauthorized persons who gain physical access to its components. All computers must be protected, since their programs (and even their hardware) can be modified by anyone who can physically touch them. Transmission lines must be protected against taps; in many cases the only solution is to send all messages in encrypted form. Personnel security is even more difficult. At present there is no choice but to trust the small group of persons who design and maintain the system; any completely

Table 2. Octopus Subnetworks.

1.	Controls 512 remote teletypewriter terminals.
2.	Controls 256 remote teletypewriter and other more advanced interactive terminals.
3.	Controls 128 remote television monitors for displaying computer output.
4.	Controls 24 remote card reader/line printer facilities.
5.	Controls centrally located high-speed printers and microfilm recorders.
6.	Controls data collection from remote experimental facilities.
7.	Controls the central data base and intercomputer file traffic.

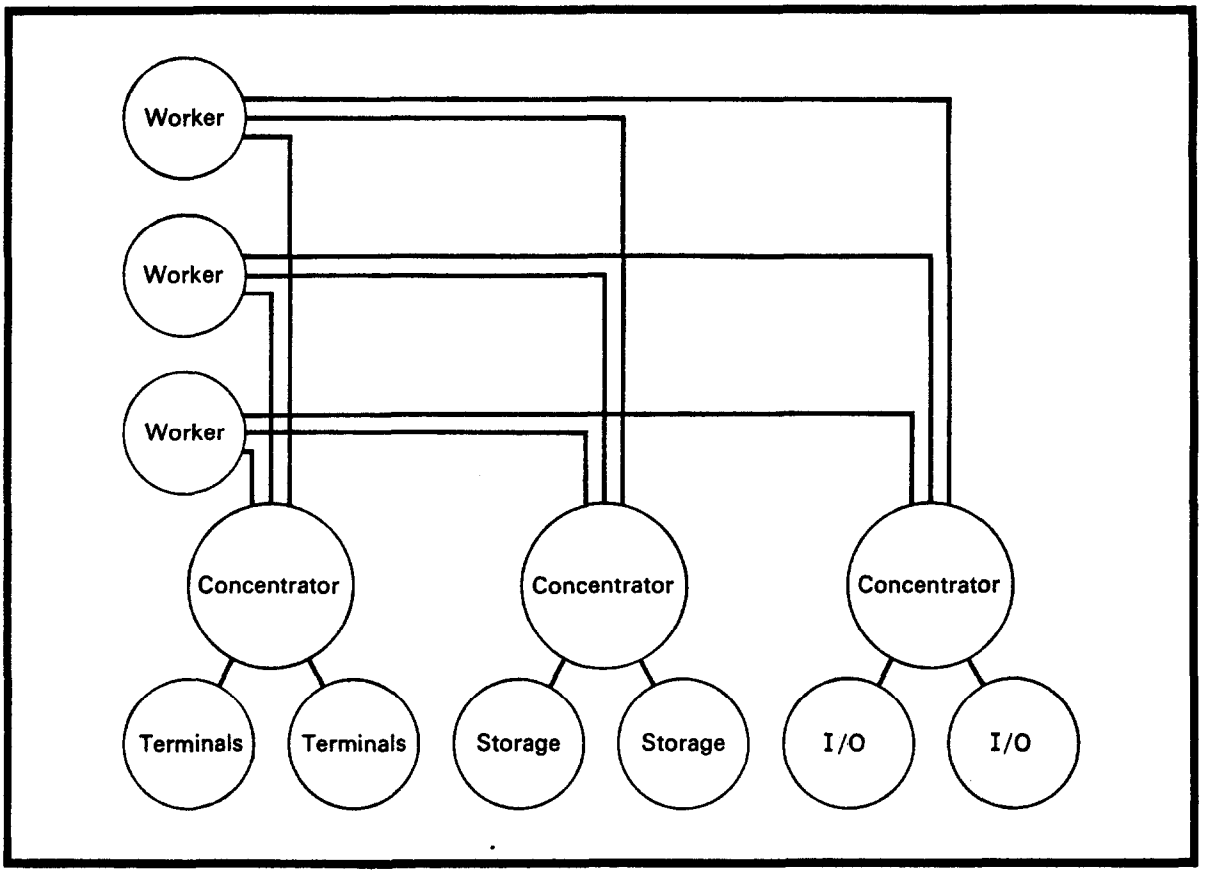

Figure 2. Octopus Network Structure.

safe scheme of program and hardware checking and verification would be prohibitively expensive. If the physical and personnel security problems are solved, then creating a secure system is not difficult. All that is required is that the designers and implementers of the system be competent and keep security constantly in mind: it is extremely difficult, or impossible, to correct an insecure system by subsequently making additions or minor alterations.

The requirement of privacy implies that the system must be able to unambiguously identify a user who has begun to use an interactive terminal. Otherwise it would not know which resources (data bases, executing programs, time allotments, etc.) should be made available to him. Octopus requires that a user type a secret *combination* (or password) at the time he begins to use a terminal. The combination, consisting of six letters, is initially generated by Octopus (using a random process), is known only to Octopus and to the user (not to any administrator), is changed periodically, and is neither printed nor displayed when typed by the user. All generation and verification of combinations is performed by a special pair of computers in the network (which have no other function); if either member of the pair fails, the other can carry on alone. Provided that users are conscientious about not writing down their combinations, this scheme seems safer than one using keys or coded cards, which can be lost or stolen. However, the future may provide (at reasonable cost) truly foolproof devices that recognize fingerprints, voiceprints, or the like.

Having identified the user, the system must next determine the resources to which he has access. Octopus utilizes an extremely flexible concept known as a *directory structure*. A directory is a body of information maintained by Octopus that points to (or lists) a number of resources; each resource is associated with a mnemonic or name by which a user refers to the resource. A directory itself is a resource; that is, one directory may point to other directories. The directories therefore form what the mathematicians call a directed graph structure: starting at any directory, one may follow numerous branching (or looping) chains of pointers that ultimately terminate at non-directory resources, such as data files (see Figure 3). For each user there is an associated, unique *root directory*; the user may access any resource that can be reached by a chain of pointers starting at his root directory. The system can access any resource whatsoever by following chains of pointers starting at a particular directory called the *master directory*. The directory structure permits each user (and the Octopus system) to create a convenient logical structure for his resources. It also permits the most general kind of resource sharing among users; each user may give every other user pointers to as many or as few of his resources as he wishes.

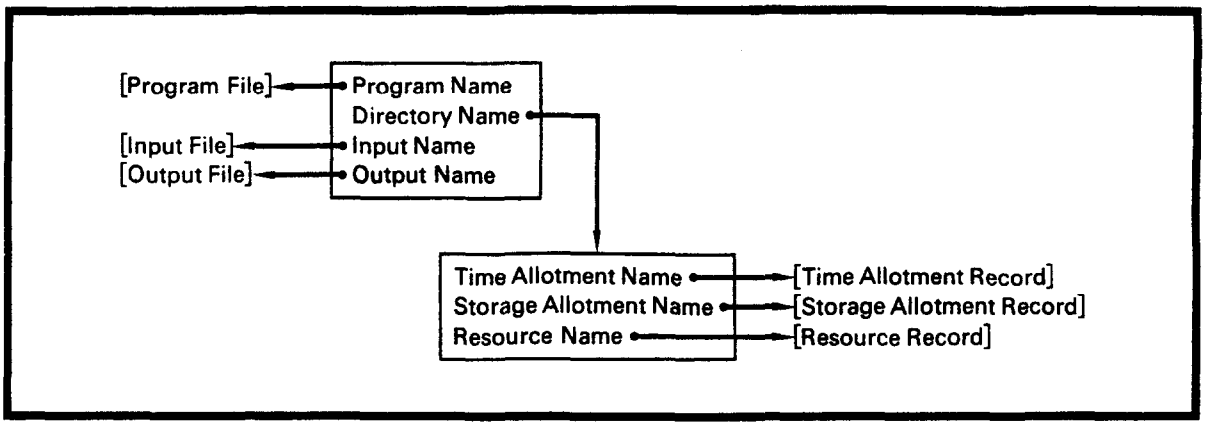

Figure 3. Example of a Portion of a Directory Structure.

LIMITED RESOURCES

Often when someone hears of the LLL computer inventory, he assumes that the users of Octopus exist in a world of effectively unlimited information processing resources. This is simply not so. Octopus resources are large because LLL computing needs are large. In fact, the majority of users' complaints about the network can be traced to the exhaustion of a resource. Octopus must be efficiently implemented because needless inefficiencies would only increase these complaints and would require additional expense to compensate for them. In achieving efficiency, a major concern of the system designers must be the effects of access delays and mismatched data transfer rates between devices that are exchanging information. In deciding what is to be done to alleviate a resource shortage, there is no simple rule to follow when balancing equipment cost, hardware and software design effort, time until availability, and the ultimate quality of service.

Since resources are always limited, some scheme must be employed to equitably limit user requests to the available supply. Quasi-economic techniques seem to be the most successful. In requesting processor time, for example, an Octopus user may bid a value in the range 0·1 to 10·0; those who bid higher receive preference. The time taken by a user's program is multiplied by his bid and deducted from his allotment of time. This, the user must balance his desire to execute his programs at popular hours of the day (when high bids are required) against his desire to execute for a long time.

Nevertheless, the entire problem of resource allocation (including processor scheduling) and charging is one of the most intractable at LLL. One difficulty is that different users exhibit different patterns of activity. For example, when considering how to include program size in the charge for processor time, one is making a decision that determines whether large or small programs are favored. The users will never agree among themselves as to what algorithm is fair and proper; each would like an algorithm favorable to himself. Another difficulty is that there is no unbiased way to determine how much time should be placed in each user's allotment. (This should be less of a problem in a commercial environment, where the allotment would be bought with real money.) Briefly, resource allocation is another area in which the system should remain flexible.

HARDWARE AND SOFTWARE

In selecting the hardware necessary to implement the Octopus design, LLL found that the most suitable equipment was not all of a single manufacture. LLL therefore employs a staff of engineers and technicians who design, install, and maintain the interfaces that join computers and other devices of differing manufacture. A connection between two computers typically consists of two interfaces; each interface adapts the input–output hardware of one computer to an Octopus standard hardware protocol, according to which the two interfaces interact (as shown in Figure 4). This means that only one kind of interface need be built for each type of computer, rather than one for each pair of interconnected types. The engineering staff also designs and installs other devices beside interfaces; this is necessary because sometimes a required kind of equipment is not available from any manufacturer.

In the case of software, LLL has found it necessary to go even further: all system programs and most other software are generated at LLL. This is necessary because of (1) the network activity peculiar to Octopus, (2) the presence of unique

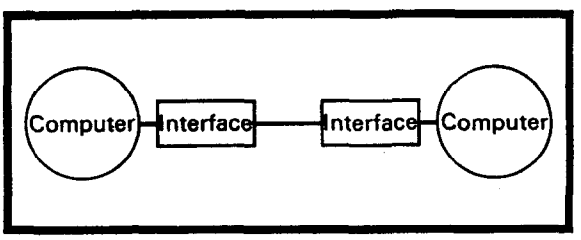

Figure 4. Typical Intercomputer Connection.

equipment, (3) the fact that manufacturer's software is often not yet available when LLL acquires a computer, and (4) the fact that commercially available software often does not adhere to the design principles outlined above, particularly with regard to security. LLL therefore employs a staff of system programmers who design, implement, and maintain Octopus software. In consequence, Octopus software is not only more advanced and more tailored to LLL needs than commercial software, but it is also more readily altered in response to changing needs. And by having hardware and software designers work together, LLL has generated designs that represent a very efficient division of function between hardware and software.

It has been found at LLL that hardware reliability is a more serious problem than software reliability: once a software error is fixed, it remains fixed, but hardware continually fatigues and wears out. Even so, it is important that software be designed and implemented with as few errors as possible. For this problem there is no panacea; there is no substitute for highly skilled and conscientious programmers who are interested in their work. At LLL, work on the computer system is kept interesting by granting programmers considerable independence. (This at least partly accounts for the fact that system programmer turnover is low.) The software designers of the network also implement their designs; there is no two-level structure of analysts and programmers. Recently this approach has been widely hailed as the significant aspect of the *chief programmer* approach to software development.

Other recently promoted techniques[3] for producing flexible and reliable software at minimal cost are in many cases irrelevant or dangerous. They are characterized by an effort to replace the good judgment of the programmer with a few simple rules regarding the programming languages or programming constructions that he should employ. The nature of the rules proposed suggests that the proposers are familiar with only a part of the computer system design and implementation problem. A complete system requires a full range of languages and techniques; the selection among them cannot be reduced to a few rules but should be left to the software expert who will do the work.

THE FINAL STEP

This discussion has followed the planning of a computer network from the determination of the users' needs, through the layout of the design, to the selection of hardware and software. The final step is obtaining funds. I can offer little assistance. The reader is on his own! ■

REFERENCES

(1) J. G. Fletcher, The Octopus Computer Network. *Datamation*, Vol. 19, No. 4, pp. 58–63 (April 1973).

(2) S. F. Mendicino, R. A. Hughes, J. T. Martin, F. H. McMahon, J. E. Ranelletti and R. G. Zwakenberg, The LRLtran Compiler, *Comm. ACM*, Vol. 11, No. 11, pp. 747–755 (November 1968).

(3) D. D. McCracken *et al.*, Revolution in Programming (a collection of five articles). *Datamation*, Vol. 19, No. 12, pp. 50–63 (December 1973).