INTRODUCTION

The requirement to construct large systems involving both men and computers to assist in the evaluational and decision making processes of command and control leads very quickly to the development of large and complex computer programs. While my own efforts have been largely devoted to the development of detailed computer simulations of strategic air warfare,1 I have had an opportunity to follow in more or less detail the progress on a few of the command and control systems. The observations which are made here relate to a broad class of computer data systems and certainly apply to the ones evolving in the command and control area.

One may think of a computer program as a progression of smaller programs or routines. Among these routines are ones which alter data presented to them, perform calculations, make new arrangements or combinations of the data, provide for employing alternate paths in the program, etc. Of considerable importance also are the control routines which bring together at the proper time the correct routine with the data on which it is to operate. In more complex programs, the sequence of use of the routines and data is subject to modification by external events.

The tougher problems in the design of computer programs begin to arise when the internal memory of the computer is too small to contain all of the routines and data involved. For the large man-machine systems the internal memory might be likened in size to a teacup used to dip a mixture of program and data from a barrel. Much larger memories will simplify some of the problems, but they will not eliminate nearly all of them.

These remarks have been made to clarify, for the present purposes, the meaning of the expression "large computer program."

The development of a large system goes through several phases. First, there is a formulation of system requirements. Then follows a period of system analysis, problem definition, and design. The computer hardware is selected, a detailed design is made and implementation is undertaken with numerous changes and delays. Experimental applications are made, redesign occurs, and modifications of the computer program are made. Further trials are made, and the cycle is repeated.

The elapsed time for the accomplishment of these phases is measured in years. The design and implementation of most of the Air Force "Big L" systems have extended for three to five years. The other services have faced similar schedules. Under very favorable circumstances, a substantial application can be brought to a productive state in about two years.
DESIGNERS' DILEMMA

Designers of large data systems begin at condition A and envision a solution B, hurdling a number of intermediate steps. Because of the anticipated elapsed time of three to five years, embarking on the task of attaining solution B without intermediate steps requires the acceptance of a fairly bold assumption. It has to be assumed that solution B, which has been arrived at in terms of present day problems, will in fact be suitable for the intended task when the system is operational. In other words, the assumption must be made that the problem and the solution will remain relatively constant. Experience has shown that this happy state does not exist. The more distant the goal, the less certain the solution becomes. Furthermore, because the problem exists at condition A, there is pressure to arrive at a useable solution as soon as practicable. Consequently, as work progresses, several compromises are made. Call these less elegant solutions in succession B₁, B₂, and B₃. As time passes, additional knowledge is acquired, and the actual requirements of the problem change to B' while the compromise solution is still B₃. A person not troubled by the esthetics of pure design could set out toward the not-crystal-clear goal B by making the step from A to A' (which may be rather close to B₃) with much greater chance of early accomplishment of the lesser goal.

Our experience with computer applications verifies the comment of one keen observer in the profession: "When we cease to change a program, we cease to use it."

If there is any conclusion to be drawn from considering these two ideas together, it is the following: Make as complete a system study as possible, but design the implementation in such a way that a portion of the system can be exercised usefully at an early date. One must attempt to build the system and its modifications so that the user can operate the system while its usefulness is being extended incrementally. It is almost inevitable that exhibiting sample products of the system will lead to changing the design of the system; and the sooner this restatement of the requirement occurs, the sooner the first round of modifications to the system can be undertaken. Changes will be required on a continuing basis as the users of the system better understand what the system can do for them.

USER PARTICIPATION

It is next to impossible for the system analysis team to discover the full extent of the system being studied or to find all the boundary conditions unless the cooperating representatives of the user are sufficiently experienced in both the operational and computer aspects of the application to exhibit all the pertinent facets of the problem. The distinction is here made between the ultimate user of the system and his representatives who are assigned to assist with the design and incremental implementation of the system. The backgrounds and perspectives of the representatives may result in the incorporation of some undesirable features in the system, if they are not able to project their thinking to the needs of the ultimate user. If the ultimate user can be involved in the early exercising of the incomplete system, clarification as to emphasis and detail should occur before the whole design is implemented in an unsatisfactory fashion.

Despite the possibility that guidance from representatives of the ultimate user may introduce some ideas into the design which will have to be modified at a later date, the participation of carefully selected representatives of the user throughout the design, implementation, and testing of the data system is vital.

Without such participation, the user's organization will be ill-prepared to understand the series of compromises which have been made to bring the system to an operational state, and will not understand that asking for too much in a hurry is likely to delay the completion of
the initial portion of the system and its incremental extensions.

The user's personnel must be able to undertake the maintenance and modification of the system after the team from outside the organization has completed its work. In order to supply this capability, the user organization must provide sufficient personnel of suitable talent to participate in the implementation of the system. These participants must be deeply involved in translating the design to language acceptable to the computer, debugging the routines created, developing test problems of proper sophistication, producing the documentation, etc. As the key personnel assigned this duty may require two years or more to become proficient in substantial portions of the system, care must be taken to assure continuity in their assignments. Until the problem of rapid turnover among computer programmers becomes less acute, it is likely that most of the more stable personnel involved will be among those in the employ of the using organization.

TESTING

For computer programs of considerable size, it is useful to distinguish three levels of testing. The programmer devises a test or tests for each subroutine he builds in order to assure that the computer reacts as anticipated to all the circumstances he considered in designing the subroutine.

As these building blocks are completed, they are ready for testing in combination with other subroutines and other aspects of their intended environment. Their input and output conventions are checked for compatibility. Larger test procedures are devised to exercise the joint functions of the program complex. Appropriate adjustments are made in the program until all anticipated results are produced.

The third stage of program testing consists of putting the program into limited operation to observe its behavior when dealing with the many variations of data which arise in practice. It is very difficult to devise test problems which employ all the possible paths in a program. Even though the program responds correctly in a limited test, it is possible after several weeks of successful operation for the program to encounter some new combination of inputs against which it was not adequately protected.

DOCUMENTATION

Documentation of the system created in one of these three-to-five-year projects is an important though time-consuming endeavor. The description of the program in the language accepted by the computer is a key part of this record and must be kept up to date as program changes are made. This level of documentation is the "court of last resort" in isolating precisely what some routine does, but the technical remarks or diary prepared by the programmer while he has all the facts in mind will reduce the amount of detective work which must be done some months later when an unanticipated change must be made. Two or three other descriptions of the system may also be required at decreasing levels of detail.

Adequate documentation is important to the person who created the computer program, as even he soon finds the maze of detail confusing without rather complete records. The person who creates one program is never free of it unless he makes it possible for others to modify his program without performing major research. The documentation must be generated so that a given programmer may be assigned to a different project. This documentation is essential, for the protection of the user, as the programmer may at some critical time no longer be available for consultation.

SIMULATION

Simple exercising of a large man-machine data system with the use of a single bank of information in a fixed pattern will bring into focus the utility of such a system to aid military decision makers only under rather limited circumstances. A simulation technique should be superimposed to alter the data flow according to various assumed alliances, postures, strategies, or the like. Such simulation exercises of the system in the operational environment by users of appropriate echelons will give...
them synthetic experience about situations which have not been encountered in sufficient detail in any other medium. Only by exercising the system with data flowing at an accelerated pace will many of its weaknesses be discovered. The simulation must, of course, incorporate uncertainty, conflicting information, degraded communications, etc., if realism is to be obtained.

Considerable effort has already been devoted to the development of detailed computer simulations of the air interactions in a nuclear exchange. It would be hoped that these simulations might be used to generate under various assumptions much of the information required to alter the behavior of the man-machine data system.

It should not be assumed that providing the simulated environment to influence the large data system is a small undertaking. Depending upon the detail introduced, the project may be even larger than the original one.

Thus far this discussion has dealt with the notions of system design, incremental implementation, inevitability of modifications, user participation throughout the project, testing at three levels, integrating documentation with the development of the program, and the usefulness of providing a simulation environment for the data system.

It also seems appropriate to record some additional remarks based on participation in or observation of several large programming efforts. These remarks will range over the topics: level of effort, estimation of time required, hardware considerations, reaction to frustration, and programming systems. While these topics are not clearly related, the comments are intended to provide at least a partial check list for those who are brave enough to undertake the development of a large computer program.

LEVEL OF EFFORT

It requires great skill and unusual circumstances for a large system of programs to arrive at a satisfactory operational state in less than three years. Even though the period since 1955 has provided several instances of this bit of truth, every new group which embarks upon a new project is dominated by persons who believe that they can avoid the mistakes made on other projects and shorten their own schedule. To some extent they do profit from the experience of others, but they usually make a few new mistakes of their own.

It is important to understand that a small team of four or five experienced analysts can accomplish about as much in the first four months as a team of twenty. Toward the end of the initial phase, the small team should be supplemented by specialists in various areas as required. There will be no advantage in gathering a team of 100 technicians at the beginning of the effort. As a matter of fact, such an action will probably delay the project.

A healthy growth in the effort applied to the project is important. A modest beginning with thorough indoctrination in the early months of the user-contractor relationship will pay off in the end.

Wide fluctuations in the support given the contractor will cause serious difficulties. If enough money is provided to acquire too large an increment of personnel, most of the old hands will be devoting too much time to training the new arrivals. If too little support is provided, progress lags and morale suffers. As new arrivals on the project will not contribute much for several months, the project is relatively insensitive, in the short run, to small personnel increases.

ESTIMATION OF TIME REQUIRED

Aside from the scarcity of experienced personnel capable of performing the scientific and engineering tasks involved in the development of a large computer data system, there are some administrative problems which lead to fixing the minimum time at two years regardless of the complexity of the system. Among the administrative problems are the following: budget justification and funding release practices, site construction schedules, and equipment delivery schedules.

It has been customary in some quarters to estimate the productivity of skilled programmers at 3000 instructions per man year. When
a great deal of on-the-job training is involved, the estimate should be set at about 1000 instructions per man year. When smaller teams are given the responsibility for well defined sub-projects, instances of 12,000 to 15,000 instructions per man year have been observed.

In estimating costs, it appears necessary to allow for each programmer two other persons classed as analysts, administrators, or engineering support personnel.

The more complex data systems fall in the 100,000 to 200,000 instruction class. Using the factors just given, one can begin to estimate the manpower required and the cost of implementing such systems. By imposing limits on the size of the team to be involved and taking into consideration the funding schedule, it is then possible to obtain a first approximation for a completion date.

After the design is relatively complete, the users and designers must agree that the design meets the requirements of the project. After the detailed programming effort begins, a major re-orientation of emphasis which requires a redesign of a major segment of the system will normally delay considerably the completion of the project.

Once agreement has been reached on the design of a specific large computer program, the user must exercise a great deal of patience and forbearance. No amount of exhortation will materially affect the progress of the project and may even cause delays. The user who schedules other events very closely on the basis of the promised completion date of a large programming effort is almost certain to have major disappointments.

Some attempts have been made to apply PERT techniques to programming efforts. Until estimating techniques in the programming field are more reliable and the addition of extra manpower on lagging subroutines has a more predictable effect in the short run, the development of a PERT network will serve only to help the managerial echelons to understand the relationship of the various pieces of the program being constructed and to make the initial assignment of effort more intelligently.

HARDWARE CONSIDERATIONS

Rapid strides are still being made in computer hardware development, and significant improvements as to speed, reliability, capacity, and flexibility are available every two or three years. It has been pointed out that the design and programming effort for large problems takes a minimum of two years.

It is hard to imagine the confusion which would arise if a computer, of not too well defined characteristics, were being constructed for a large problem which was being programmed at the same time. First, it would require most of the hardware construction period to create the sophisticated programming system required to permit the large problem to be accepted by the computer. Further, about half way through the period, there would be a requirement to debug some of the routines of the large problem.

As the problem has probably changed materially during the programming period and most certainly will change as soon as the program has been used a few times, most persons who have considered these facts have agreed that nothing is to be gained by trying to specify the characteristics of a large-scale special purpose computer. If it is built to satisfy the requirements of a problem as defined now, there is a good chance that it will not be adequate for the problem as it is understood shortly after the computer is completed three years from now.

A useful competition among the computer hardware manufacturers is in progress. For reasons which need not be considered here, different types of computers are being selected for installation at various locations on the basis of current requirements with well-documented justifications. Much of what is to be done at location A on computer X may also at some time have to be done at location B on computer Y, etc. If a programming system implemented on computers X and Y would, in fact, accept the programs first written for the other computer with only modest changes, considerable savings would result.
REACTION TO FRUSTRATION

Any person who has the tenacity to spend several weeks trying independently different approaches to the solution of a puzzle before success can appreciate in some small measure the frustrations of the programming activity and the final feeling of personal triumph when the last important programming bug is corrected. As the process of preparing a large program often extends over more than a year and the mistakes of many people are involved, the frustrations are of a higher order. In the normal course of events, nerves become frayed, fingers are pointed, neuroses appear, and other psychological and physiological reactions are not unknown. A few persons on the project who accept the ultimate technical responsibility for the completion of the project will find the pressures unabated for a period of months. It is small wonder that persons, who have experienced this trauma a time or two, have fairly firm ideas as to the circumstances under which they will accept such a responsibility again. The completion of the project will lag unless this concentrated effort does occur, and the managerial chain on the project cannot, at the beginning, predict which members of the team will punish themselves in this fashion.

One of the important contributions being made by the more powerful programming systems is the elimination of many of the frequent recording and cross-referencing errors so that the more important logical blunders do not remain concealed for long in the debugging process. This alone materially reduces the length of the effort and permits more concentration on the solution of the important problems.

PROGRAMMING SYSTEMS

Because of the general confusion in connection with the use of the words "programming system," it may be helpful to distinguish (1) programming languages, (2) compilers, and (3) operating systems. A programming language is the set of symbols and conventions which are designed for the convenience of the programmer to express his detailed solution to the logical, manipulative, and computational problems encountered. A compiler translates the expressions of the programming language into sequences of instructions in the basic language of the computer hardware.

An operating system augments the capabilities of the computer in such a fashion as to relieve the programmer of the requirement to provide in detail for input/output scheduling and assignment, memory allocation, etc., so that the programmer need not resolve these recurring problems in numerous variations and can devote his efforts to implementing the data system design.

The JOVIAL system includes all three of these components and has been implemented on several large-scale computers, but the versions of the language are not identical.

NELIAC has been widely used and consists of a language and a compiler. The data routines are incorporated during the compilation. This technique has permitted the larger computers to compile programs for computers with memories too small to accommodate a compiler.

The CL-II programming system consists only of a compiler and an operating system. The compiler is described as "syntax-directed" and can accept any of the present algebraic languages for which it is provided the necessary input tables. It has been demonstrated that careful preparation of these tables enables the compiler to generate as compact code as an experienced programmer. The sophisticated operating system of CL-II is based on the concept of the "extended machine" first expressed by Holt and Turanski.

A good programming system should

(a) Encourage modular construction of the computer programs.
(b) Provide for data descriptions which are independent of the program until compilation occurs.
(c) Relieve the programmers of the necessity of constructing substantial debugging environments.
(d) Eliminate the need for developing elaborate control programs.
There seems to be no way to avoid all the human errors which are made in the course of developing a large computer program, but a good programming system can relieve the programmer of many of the housekeeping problems so that his thoughts can be directed to the peculiarities of the data system being automated.

If there is to be reasonable progress in the production of a large computer data system, the programmer's work should be reduced as much as possible, and the initial system should be constructed with the anticipation that changes and extensions of the system will be required on a continuing basis.

SUMMARY

In closing, it seems worthwhile to summarize the main points of this discussion:

(a) The users of the data system must collaborate with the technicians throughout the design, construction, and testing of the system.

(b) A full design is desirable, but a path should be chosen through this design so that a useful sub-system is available as soon as possible.

(c) Anticipate changes and extensions to the system.

(d) Do not stint on the documentation.

(e) Exercising the system fully will probably require building a simulated environment.

(f) Putting extra manpower on a project at the wrong time to accelerate the effort will usually accomplish little except to heighten the disappointment.

(g) Large-scale special purpose computers still seem inadvisable because of changing requirements.

(h) The programmer who takes his profession seriously does not find it to be an easy life.

(i) Programming systems may not have reached perfection, but they offer the programmer some relief from many of the troublesome housekeeping details.

(j) Standardizing on a single algebraic programming language is apparently no longer required.

No claim is made that these observations form an exhaustive list of all matters to be considered by a group embarking upon the development of a large computer data system to assist in the evaluational and decision making processes of command and control. Each person who has been involved in such an effort will have a few items to add. Perhaps even this partial list will help some group to avoid a few of the mistakes others have made.

REFERENCES


