The management and organization of large scale software development projects*

by RONALD H. KAY

IBM Research Laboratory
San Jose, California

INTRODUCTION

Two consecutive papers on the subject, "Managing the Economics of Computer Programming" presented at the 1968 National Conference of the Association of Computing Machinery conclude respectively:

> - "First, one must understand computer programming well enough to know what is possible, what is probable, and what is impossible or unlikely.
> - Second, one must make commitments based on the technology used, not on the needs of the world—and not on the unreasonable hopes of the starry-eyed experts.
> - Third, one must insist upon schedules based on physical events, and on numerical descriptions of the products that are being produced, to the greatest extent that ingenuity will permit.
> - Fourth, one must objectively assess the status of the project against a well-developed plan.
> - Finally, of course, one must do something about the trouble one finds.
>
> Thus, given these prerequisites, I conclude that computer programming can in many respects be managed just like any other process."[1]

> "We do not really know how to select programmers, and we tend to select those with some undesirable characteristics. . . . Typically, they work for a manager who is ineffective because he has been given neither proper management training nor basic tools and disciplines with which to work; whose functions have not been defined, and whose process of communication with the system analyst or user is generally confused. Finally, all this takes place within a technology which changes so rapidly that it is almost impossible to get a fix on the functions and the method by which the work is supposed to take place, before it changes."[2]

An equally wide range of views emerged from a series of seminars organized by Professor Maison Hare of the Sloan School of Management and Professor Malcom Jones, Assistant Director of Project MAC during the fall of 1968 at MIT.

Insofar as the seminar speakers represent independent organizations with widely varying objectives, divergent views on many issues are hardly surprising. Management objectives will be viewed differently by a software firm working on a contract, a computer manufacturer developing an operating system for his hardware, or a university attempting to develop new concepts in time sharing systems.

Thus, the basic contention that the management of large programming efforts does or does not present a unique problem, suggested in the above quotations, may reflect the relevant management experience which an organization has been able to bring to bear upon a problem, rather than conflicting conclusions drawn from a given set of premises. Each of the invited speakers* had extensive experience in the management of large programming efforts and is, or had been, associated with an organization presently involved in such efforts. Most of the organizations represented have won recognition both, for leadership in technical innovation as well as success in the large scale applica-

---

* Based on a seminar series attended by the author while a Fellow at the MIT Center for Advanced Engineering Studies, Fall, 1968.

* The names and affiliations of the participating speakers are appended.

tion of new technology. Unfortunately, some of the organizations active in the advancement of this field of management and whose work was referred to by several speakers, were not represented.[3]

It must also be pointed out that most of the speakers were executives with direct responsibility for large programming efforts or for all programming activity in their organization rather than specialists in management science concerned with the advancement of this field.

It would be presumptuous to attempt a condensation of these seminars representing the experience and wisdom of the 21 speakers and the reactions of the other participants. Rather, this paper endeavors to find common threads and possibly resolve conflicting views on specific issues. It is based entirely upon the content of the seminars, and represents the author's interpretation for which he assumes full responsibility.

This interpretation reflects the relative weight given by the speakers to various aspects of the subject, with some additional emphasis where there was significant disagreement among the contributors.

Since the more visible sources of management difficulties are often the derivatives of unclear or changing objectives, this issue is identified at the outset. Next, the need for accurate assessment of the state of the art, the difficulties of making such an assessment and its relevance to the project organization are considered. Arguments are advanced for two kinds of organizations based upon the objectives and the required degree of innovation.

The scope of this paper permits only the most general observations regarding techniques for project planning. The significance of the project plan as the principal means for rendering a programming effort visible is pointed out. The discussion of specifying and evaluating a complex programming system and measuring the performance of the people responsible for its development serves to illustrate the magnitude of the problem faced by management at this point in time. The lack of a common set of tools and standards are pointed out as some of the specific causes of difficulty.

The issue of higher level languages is still an attractive topic for debate; those in favor appear to be pulling ahead. This analysis of presently held views concludes with the observation that recent advances in the shared use of computers through interactive terminals promises to provide programming development management with more effective means for the control of the implementation phase by putting documentation online.

The basic assumption underlying the seminar series, i.e., the existence of significant unresolved problems was confirmed. There is little contention that frequent underestimation of the risk, possibly more than in other areas of innovative development, is due to the disproportionate responsibilities placed upon programming management relative to their experience and the maturity of the field.

The term "programming" is here used in its broadest meaning to include all phases of software development. In emphasizing "Large Scale Software Development Projects" the seminars focussed principally on complex operating systems, command and control systems and large simulation efforts, in each case involving groups large enough to require several levels of management. The systems referred to typically involved 100,000 to several million instructions.

Objectives

The statement of definitive and time-invariant functional objectives at the outset of a large programming effort has proven to be one of the requirements most difficult to satisfy. For example, the compatibility of an operating system relative to a family of hardware or the extendability of a storage/management approach to a multi-processor system are objectives difficult to formulate at the outset of an effort intended to break new ground in these areas. In spite of this, management has frequently accepted vague or unrealistic objectives in the expectation that the project itself will produce the required clarification or advances.

Inadequate objectives at the outset, more than any other single factor, are held responsible for subsequent modification and consequent overrun of initial schedules and budgets. It is felt that this situation will continue as long as management or the sponsoring agency feel the risk to be commensurate with the potential benefit.

Why are objectives held to be more vague and subject to change in the case of large programming efforts than in the case of hardware development? Certainly, the relative experience accumulated in the two areas favors definition of hardware. Even when objectives are dictated by non-technical considerations, those charged with the responsibility for setting these objectives are more familiar with hardware development. They may avail themselves of advice from programming experts but their judgment of possible alternatives is still influenced by their experience.

From the views expressed by several of the seminar speakers, one is led to conclude that as long as the economic motivation associated with hardware is greater or thought to be greater than that associated with software, the latter will be adapted and modified.

Although only a few of the speakers explicitly referred to the influence of the organization's long term objectives such as profitability, personnel policy and the organization's image, it is evident that they can have a significant bearing upon the management approach. The shortage of experienced programming development managers who can effectively implement such objectives and in some cases, the apparent reluctance to face or discuss the issue of objectives tends to intensify potential difficulties.

The matter of objectives has been emphasized since the more visible sources of management difficulties are often the derivatives of unclear or changing objectives.

Assessment of the possible

While objectives are elusive, the outcome is predetermined by the state of the art. A particularly useful definition of the state of the art differentiates between three levels: (1) what is possible for the experts; (2) what is generally known; and, (3) what has been done by a development organization. This definition implies that expert knowledge must be relied upon to assess the degree to which the objectives depend upon contributions by expert personnel.

The first thing insisted upon by the experts is that there is no substitute for a basic conceptual framework which stands the test of time. That is to say, if the basic concepts and philosophy of implementation do not stand up, no amount of administrative technique can be successfully brought to the rescue. The lack of scientific discipline in the field, i.e., the absence of first principles upon which to base first principle calculation, so successfully applied in the physical sciences and engineering, produces the dilemma of conflicting expert opinions. The field has as yet not produced an adequate number of people who can match their managerial skills with this sort of expertise (or vice versa) to be able to resolve such conflicts with confidence.

Experts can be relied upon more readily for an assessment of what is generally known, or more precisely, of the advances required in what is generally known in order to meet the objectives. Frequently, the need for this implied precision is inadequately appreciated in the assessment by expert and manager alike.

It is only natural that the current literature on the management of programming efforts is more fruitful in the realm of what has been done by a development organization. There is persuasive evidence that many organizations have the competence to manage even complex tasks which require only minor advances in the state of the art.

Organization

The assessment of the possible in terms of three levels of the state of the art suggests that the organization of the project should reflect the objectives as well as the required degree of innovation. Management problems and compromised objectives have been traced repeatedly to situations where the resulting programming system reflects the a priori organization chosen for its development.[4]

Accepting the need for tailoring the organization to the objectives, the case has been made for two very different approaches.

1. A relatively small group of experts and selected support personnel charged with carrying the project from inception to completion.
2. A small group of experts in an advisory and monitoring function to the management of a large development organization where distinct groups of people are responsible for various phases of the effort such as analysis, design, implementation, integration, testing and maintenance.

Although these two approaches are frequently argued on their absolute merits, more often than not they are born of necessity.

A one-time commitment on the part of a university to a large programming system effort is often justified on the basis of an available small group of experts and their ability to muster a temporary support group with better than average qualifications.

An industrial organization committed to continuing development activities requiring varying numbers of people for a variety of tasks on a continuing basis, finds it necessary and effective to develop specialized centers of competence so that a number of projects can draw upon this resource.

It appears that the larger the relative need for innovation at the expert level, the stronger the preference for the small group. Given these two types of organization, the management techniques which find favor differ greatly. The large contract software organization may respond to the first sign of a problem by getting machinery in motion to hire or transfer an additional 100 programmers to the project. The rationale for this approach is that it may take six months to really understand the nature of the problem. At that time you probably have between 10 and 100 people with up to six months hands-on experience from whom you can select a few who can now be identified as being able to correct the problem.

The small group of experts, developing a complex system would react differently. Having less of a communication problem, they would identify the cause and come up with a potential solution in a shorter period of time. But by definition they would expect to make use of the same small number of people to correct the situation accepting the unavoidable postponement of other planned activity. Depending on the magnitude of the fix required, this could be an appreciable fraction of the original estimate. It can also be concluded that the small group is less able to absorb some of the influences beyond its control, such as, turnover of personnel, limited machine access, etc.

Although some of the speakers attempted to address the organizational issues in reference to the classical distinction between project and functional organization, it became evident that the definition of the various functions of software development in the represented organizations are not sufficiently precise or uniform to allow for meaningful generalization.

Depending upon the priorities of management objectives, such as minimum cost, fixed deadline or optimum performance, there is some basis for choosing an appropriate organization: ability to muster resources, degree of innovation required and long term objectives such as developing skills vs. hiring experienced people.

The wide divergence of views on this subject reflected the difference in management objectives and philosophy of the organizations represented.

The project plan

A significant amount of attention was given by a number of seminar speakers to the subject of a project plan. Such a plan identifies various phases of the project such as Analysis, Specification, Design, Implementation, Integration, Testing, Publication and Maintenance.

We shall here confine ourselves to some general observations regarding such a plan.

- The plan is not an end in itself, but a management tool which helps define responsibilities and checkpoints. It is the principal means for achieving visibility of the project.
- There is considerable overlapping of the various phases of the plan in the case of programming development; e.g., testing is initiated with the specification phase, where component-, integration-and acceptance-tests must be defined.
- Such plans point up a basic difference between hardware and software development. Hardware development, culminating in a tested prototype, leads to the subsequent manufacturing phase which generally requires much greater resources and thus becomes a major factor in the ultimate success of the project.
- Programming development culminates in the end-product and in this sense resembles the development of one-of-a-kind hardware.
- The use of PERT charts is generally held to be ineffective as a means of planning and control of large programming development tasks.
- Project control against a plan is not unique to programming projects.

Problems frequently are not due to a poor plan but to the fact that the plan is not being carried out for a variety of reasons, some of which are being considered in this paper. Above all, a good plan is no substitute for poorly defined objectives.

Specifications

One of the most difficult aspects of programming development is the process by which the results of analysis are translated into a set of specifications. What is wanted is a set of blueprints which uniquely specify what is to be implemented. The first problem arises with the decision as to what should go onto which blueprint. The need to break the job down into separate modules forces early decisions regarding the interaction between the modules. An attempt to expalin the concept of "Functional Modularity" may help to clarify this issue. In the case of hardware, modularity is derived from considerations such as physical dimension (what one can get through a door, tolerable delay, etc.) components which can be shared (e.g., power supplies) ease of access and replacement, standardization of modules, etc. Once a hardware module is defined, its relation to other modules is fixed by virtue of a finite number of interconnections. Every physically accessible connection is a potential test point permitting isolation of modules. In a complex programming system modularity is sought in terms of frequently used sections of the program and elements which are common to several functions. There is a desire to minimize interaction between modules and to achieve clean separation of function to facilitate division of the development effort and module testing. This concept of modularity as yet does not take account of the fact that not all parts of a complex program can be equally accessible at all times, i.e., sections of the program must be moved to provide the desired access. It is fairly obvious that the larger the "module" which is moved the fewer the required moves. Yet, a large module occupies more prime space and takes longer to move.

This suggests that the various considerations which influence modularity in a large programming system are functionally interrelated in a much more complex way than the parameters which influence hardware modularity. Thus, to achieve "Functional Modularity" at the specification stage, implies the need for past experience which proves relevant to the problem at hand or a methodical approach to reduce the inherent complexity. Neither appears to exist in the case of large programming development tasks, particularly where new ground is to be broken. This most abstract aspect of programming development remains the intellectually most challenging.

Another problem relating to programming specifications involves the degree of detail necessary. It is argued that to assure equivalent results from two programmers given the same specification, a level of detail (and effort) nearly equal to the program itself is required.

Reliance upon experience which may or may not prove relevant and the lack of a methodical approach hardly ease the problem of evaluating the end-product against a set of specifications.

Evaluation of the end-product

The end product of the development is a programming system, i.e., a collection of programs designed to perform a specified function in conjunction with specified hardware.

For example, a large systems program must, among other things, provide the scheduling and allocation of system resources as called for by a particular set of instructions, i.e., by the application program. What constitutes a "typical" application program or set of programs which would provide a realistic measure of performance?

Another aspect of evaluation is the data dependence of the program. To illustrate this point, consider the logical combination of two data elements whose combined value exceeds the capacity of some hardware facility. The programming system must be designed to cope with this problem in terms of all possible logical combinations of all possible data elements with regard to all possible combinations of affected hardware components. How can this be done? Only the most careful design of test programs and the most extensive test cases can hope to provide a satisfactory approximation to "all possible combinations."

Given this level of evaluation, what can we say about the quality of the programming system in its ability to cope with a given job stream which has a unique sequence of programs? How representative is this given job stream of the types of applications a variety of users are likely to encounter? How do the answers to these questions relate to a multi-processing or multi-user environment? What is more, how can we evaluate such objectives as compatibility and useful generality which are related to past and future hardware and applications?

To date, there simply are no generally satisfactory answers to these questions. Even when satisfactory answers are obtainable for a specific subset of the desired range of parameters, the evaluations upon which these answers are based can only be attempted *after* successful integration, i.e., when the job is presumably done.

While techniques have been developed which provide a basis for predicting the mean time between failures of hardware components, the asymptotic nature of program debugging and the effect of transient causes of error upon program behavior introduce elements of uncertainty which are difficult to quantify.

Evaluation of a programming system, probably better than any other aspect of the development cycle, illustrates the level of complexity and the relative lack of proven techniques in this field.

Evaluation of the programmer

The difference in approach to organization and the difficulty of evaluating the end-product highlights one of the unique problems of programming management: The measurement or evaluation of a programmer's effort.* There is conclusive evidence that there are order of magnitude differences in individual performance by almost every criteria, such as time required to complete an assignment, tightness of code, quality of documentation, running time, storage requirements, and computer time required for debugging. These criteria serve as indicators, but realistic measures of performance based upon these indicators are qualitative, not quantitative. These indicators would not necessarily point up interfacing problems before integration, or provide a measure of ultimate performance but would serve only as a warning in the case of extreme departure from the norm.

Two principal reasons for the large variation in performance among programmers and the difficulties of measuring this performance are considered in some detail:

- The craft-like nature of programming.
- The personality traits of programmers.

---

* "Programmer" is here defined to include all personnel engaged in the analysis, design, implementation and testing of computer programs.

Programming—Its craft-like nature

Non-programmers find it difficult to understand how a task, at once requiring the utmost in logical consistency, at the same time can provide so much choice in the approach to a given problem.

Hardware engineering experienced the same problems in its evolution from the skills of the craftsman to the mature technologies based on various branches of science. Today, two hardware engineers given a task to perform generally can agree on what constitutes a precise definition of the problem and what constitutes an adequately tested and documented solution. Even though their end products may look different, there will be considerable resemblance in their approach. They will go through clearly-identifiable prodecures such as analysis based on a set of equations, circuit diagrams, breadboard models, tested models, prototypes, etc.

In programming, on the other hand, the situation is different. There is generally no way to relate the work of two programmers or even the same programmer's work on two different jobs. Unlike engineering, the road from gross program design to a detailed design is a function of a set of highly unpredictable human events. As the job progresses, its nature tends to be redefined as the programmer becomes more familiar with the problem. The way he reacts to this increasing awareness and translates his reaction to the program is highly idiosyncratic to the individual and to the individual project.

Programmers are different

In listing the personality traits of programmers as a source of difficulty in the measurement of their performance, it must be understood that the difficulties referred to are those perceived by management. Many programmers probably accept the fact that they are not easily measured.

What is the basis for the assertion that programmers are different?

First of all, most managers who are led to this conclusion compare programmers to engineers.

Second, many programmers are recruited from the ranks of liberal arts graduates, while the managers were trained in engineering or business administration.

Third, in many organizations the programmers are the most homogeneous age group. They are thought to represent a readily-identifiable group of young people, bringing the new look from high school or college into certain segments of industry and government. This new look is sometimes equated with appearance and attitudes designed to set themselves apart from the existing majority.

Finally, many aspects of programming require a high degree of concentration over an extended period of time which tends to make programming a solitary occupation; those drawn and devoted to it may be or become more introverted than the majority of their nonprogramming colleagues.

Although management training in the liberal arts has long been advocated, few present day managers are prepared to cope with the generation gap or the culture gap. Until enough managers can be drawn from the ranks of programmers, the problem is likely to persist.

The tools of the trade

Most craft-like processes which have been carried on for some time evolve a set of tools which are available to the community of craftsmen. Their skill level may vary, but their tool kit is generally the same. In the case of large programming systems, and particularly when major advances in the state of the art are to be incorporated into the system, adequate tools may not exist. In fact, the evaluation of the adequacy of available tools or the creation of such tools can constitute a significant aspect of the project. The adequacy of the computer and the operating system available to the development effort is a case in point. Higher level languages and the adequacy of the available implementation (compilers, etc.) fall into this category also.

Another unique problem of programming development is pointed up by the lack of a programmer's equivalent to an oscilloscope, this most useful of electrical engineering tools. The core dump which reflects the contents of storage locations at a given program step is the nearest equivalent. It roughly corresponds to a simultaneous presentation of the wave form of every possible test point in a circuit ordered according to the geometric location of solder joints rather than points on a diagram.

It takes a great deal of experience for a programmer to make effective use of more sophisticated techniques such as the "snapshot" or its equivalent. He must learn to structure his program to permit meaningful tests to be performed without affecting the desired operation of the program.

Higher level languages

An issue certain to provoke heated discussion among the experts seems to be resolving itself in favor of the use of higher level languages for system programming.

The arguments for higher level languages include:

- Better communication where interaction of programmers is essential.
- More compact documentation—a significant factor in systems of several hundred thousand instructions.
- Facilitation of debugging.
- Closer relation to the conception of the algorithm.
- The ease of transferring the resulting programs, i.e., less machine dependence.
- The potential savings in programming cost resulting from the above.

Meaningful arguments against higher level languages can be made in some special cases

- Real time systems where running time efficiency is paramount.
- Inner loops in large scale computation where computer capability is taxed..
- An adequate compiler for the proposed higher level language is not available.
- Retraining time of experienced assembly language programmers is not commensurate with the schedule.

Only the most sophisticated specialists will claim an advantage for a special system programming language. In time, they may be proven right. (They usually are.)

It should be mentioned that there is considerable exploration of special higher level languages aimed at providing better production tools: specification languages, simulation and modelling languages as well as systems languages.

Never trust the computer manufacturer

This cry is heard with sufficient frequency that it cannot be ignored. At its most vehement, it comes from inexperienced academicians, semi-annually reassured of their omniscience by a sea of bewildered undergraduate faces. Trained in an atmosphere of distrust of their own institutions' administration, they assume contractual responsibility for computer related projects and point with pride to the efficacy of informal arrangements, "cutting through the red tape."

The manufacturer of hardware does not help this situation by responding with a salesman admonished to preserve good relations, but with little or no pertinent technical experience. Properly impressed by the technical knowledge of the young professor who has assured him that only a minor modification is needed, he will make commitments based on his conviction that if it can be done, his company surely will do it. These commitments all too frequently lack the degree of precision required to assess the magnitude of the requested modification and are often made with little or no knowledge of the available resources which would permit a timely response.

Numerous businesses, educational and government organizations succeed in consummating contractual arrangements involving computer software. They do so by availing themselves of the services of personnel experienced in the negotiation of contracts who will ascertain the required detail and level of authorization needed to make a commitment. This suggests that recognition of this problem on the part of responsible members on both sides should be sufficient to remedy this situation.

Documentation—Asset or liability

The importance of documentation in the management of large programming developments is generally accepted. A number of groups have found a formal system of documentation the most effective management tool at their disposal. In its most advanced implementation, such a system of documentation is on-line to a time sharing system available to all participating members of the system programming project.

Difficulties often are related to the programmer's resistance to documentation which may be due to several reasons:

- Lack of tangible evidence of benefit to his own activity.
- The inaccessibility of his colleagues' documentation because of sheer quantity, lack of organization and common format and out of date status.
- Rejection of standards, imposed for reasons he does not appreciate.
- Belief (often confirmed) that he can get along without, and in fact feel at his creative "best" when free to improvise.

Putting a documentation system on-line appears to have overcome this resistance in a manner acceptable to the programmer.

- The system itself can help him by rejecting certain types of inconsistencies.
- He has instant access to the latest version of his colleagues' work.
- Standards have been translated into formatting conventions with which he is familiar.
- He understands that the system must safeguard itself and his programs from unauthorized change. Thus, he more readily accepts the need for authorization to change and implement.

Given such a documentation system, management can institute necessary controls such as a senior programmer's or analyst's approval of a proposed approach prior to implementation. Communication, which is universally identified as a major problem in the development of large programming systems, is facilitated and documentation becomes incidental and concurrent to the development effort. It is conceivable that the management of large programming efforts in the future will be structured in keeping with a well proven system of documentation.

SUMMARY

An attempt has been made to define some of the problems of large scale software developments as seen by managers experienced in this field. This definition has taken the form of identifying generally agreed upon solutions, where they exist, providing the rationale for opposing points of view, and by exploring issues which are largely unresolved.

Among the unresolved issues one finds:

- There is a relative lack of experience at the level of management responsible for setting objectives.
- This is aggravated by the shortage of experts capable of assessing the relevant state of the art.
- In more mature fields of endeavor, managers have been drawn largely from the ranks. Today's programming managers often have a different educational background from the programmers, and are not trained to overcome this difference.
- Complexity, rather than the size of large programming systems has introduced a level of uncertainty by forcing the evaluation of success potential well beyond the design or implementation phase.
- The craft-like nature of programming, i.e., the lack of scientific discipline has proven a real source of problems, such as the difficulty of evolving standards which in turn has made it difficult to specify the task to be performed or to evaluate the end-product.

Issues which have been resolved satisfactorily by at least some organization include:

- Where the advice of experts is available at the outset, it is possible to identify the objectives which should dictate the project organization.
- A project plan can be structured to provide the management tools which allow the measurement of progress against a plan, i.e., means of rendering the development of programming systems visible can be provided.
- Methods of documentation can be developed as an integral part of the effort which aid management in both evaluation and control of the project.

CONCLUSIONS

By committing the time of key executives to this seminar series, organizations large and small have shown their interest in, and support of cooperative efforts to better our understanding of the issues and to share experience.

Some of the organizations represented which have developed effective management techniques in areas other than programming and whose activities span the range from research to production have been able to apply much of their management know-how to programming efforts. The success based upon these techniques has not been unqualified. One reason is the difficulty of relating the visibility of "progress" during a programming effort to ultimate performance. The fact that schedules are being met does not insure success during integration or anticipated performance.

Where management techniques have not evolved and their lack is first felt in the pursuit of a large programming effort, the problems tend to be thought of as unique to programming. Extrapolation and scaling up from past programming experience has proven hazardous. Complexity turns out to be a non-linear attribute.

The management of large programming systems presents some unique challenges. Those most intimately involved recognize the problems. To date, they have had limited success in conveying the full significance of the problem to policy-making management.

To the extent that unresolved problems in the management of large scale software development have been recognized, one can now turn to examining the appropriateness of efforts proposed or under way, as to their potential of providing desired solutions.

REFERENCES

1 C H REYNOLDS
Proc ACM National Conf 1968 334–337
2 D H BRANDON
ibid 332–334
3 E A NELSON
ibid 346–349
4 M E CONWAY
Datamation Vol 14 No 4 April 1968 28–32

APPENDIX I

Participating Seminar Speakers

Mr. Joel Aron
Manager, Boston Programming

IBM
Cambridge, Mass.

Mr. Thomas E. Cheatham
Computer Associates, Inc.
Wakefield, Mass.

Mr. Ted Climis
Director of Programming
IBM
Armonk, New York

Mr. Larry Constantine
Information and Systems Institute, Inc.
Cambridge, Mass.

Professor Fernando J. Corbato
Project MAC
MIT

Mr. Ted Crowley
Bell Telephone Laboratories
Whippany, New Jersey

Dr. Ruth Davis
National Library of Medicine
Bethesda, Maryland

Mr. A. Dean
Manager, Information Laboratory
General Electric
Cambridge, Mass.

Dr. Donald L. Durkey, Vice President
Computing and Software, Inc.
Panorama City, California

Mr. Robert Everett
The MITRE Corporation
Bedford, Mass. and
Baileys Crossroads, Virginia

Professor Jay W. Forrester
Sloan School of Management
MIT

Professor Edward L. Glaser
Director of the Computing Center
Case Western Reserve University
Cleveland, Ohio

Mr. Neil Gorchow
Vice President, Systems Programming
UNIVAC
Philadelphia, Pennsylvania

Mr. William O. Harden
Manager, Data Processing
Union Carbide Corporation
New York, New York

Mr. Alexander S. Lett
Time Sharing Systems Development
IBM
Yorktown Heights, New York

Professor Donald Marquis
Sloan School of Management
MIT

Mr. George A. Mealy
Computer Consultant
Boston, Mass.

Mr. Donald Ream
U.S. Naval Ship Engineering Center
Washington, D.C.

Mr. Carl H. Reynolds, President
Computer Usage Development Corporation
Mount Kisco, New York

Professor Daniel Roos
Director, Systems Lab
Department of Civil Engineering
MIT

Mr. Charles Zraket
The MITRE Corporation
Bedford, Mass. and
Baileys Crossroads, Virginia