An organization for successful project management

by DON SMITH, JR.

Computer Sciences Corporation
El Segundo, California

INTRODUCTION

Successful software development requires that the product be accepted, that schedules are met, and that costs are acceptable. The number of software development efforts that do not meet one or more of these criteria is large. Conversely, there are many examples of successful development. The total number and variety of considerations that affect the probability of success is so large as to be beyond the scope of any individual paper. There is one important element of software development that appears to have been somewhat overlooked, however, and is the subject of this paper.

This paper presents the thesis that the probability of software development success is much increased by: (1) a proper separation of responsibility within the project organization, combined with; (2) extensive formal procedures; the combination of the two providing a system of checks and balances that gives continuous, accurate visibility of project status.

A general project organization is presented here that is felt to be widely applicable, and that achieves the required set of checks and balances. The formal procedures required to enforce the organizational division are not described completely or in detail, but examples are given.

It is felt that viable commercial products will only rarely be successfully developed with project organizations that do not provide some equivalent to the checks and balances approach described here. It can be argued that superior products may be developed at less cost and more quickly by small expert groups carrying the project from inception to completion. The technical contributions possible from such an approach cannot be denied, but the need still exists to place these contributions in an environment that assures their control. It should be noted that control need not be exercised as a function that unduly limits imagination. The intent of control is to assure that considerations of product function, cost and schedule remain clearly visible, and that decisions made which significantly affect such considerations are made consciously, with the effect of such decisions having received due consideration. Concerning the time at which control should be initiated, it is felt that "control begins when the first germ of an idea for a project appears in the organization—a discernible effort which will require an expenditure of organizational resources (human effort, time, money, physical resources) and lead to an organizational objective." References 2 and 3 contain related reading not totally in accord with the above view.

There are, of course, many problems of software production other than those related to project management that can prevent the end product from being successful, such as: a faulty or unknown product definition; badly estimated product life; too rapid a development speed; major design problems; poor staffing; and inappropriate marketing (see also Sections III and IV of Reference 4). Good management can in many cases, however, assist in minimizing such problems by providing early visibility of the product and its problems, and by providing an environment allowing timely remedial action.

Prior to continuing with the body of this paper, one reference is particularly recommended for project planners. This reference, by Bemer, contains an extremely comprehensive and useful "Checklist for Planning Software System Production." Reference 4, also by Bemer, is further recommended reading.

MAJOR PROBLEMS IN SOFTWARE DEVELOPMENT

This section describes the problem areas that are felt to be historically most common. The problems
described in this section are felt to be significantly reduced by the organization and procedures later described.

In the most general sense, the two major problem classes are:

Those that make the customer unhappy; and
Those that make the developer unhappy.

Problems in the above two classes might also be classified as follows:

Unsatisfactory product
Schedule delays
Excessive costs

The above problem classes clearly are interactive, as schedule problems lead to cost problems, an unsatisfactory product leads to remedial action (cost, schedule), etc. Thus the division is to some degree artificial, but is useful for descriptive purposes.

The above three classes of problems will now be discussed in more detail.

**Unsatisfactory Product**

The major causes of user dissatisfaction with the product are:

1. Too Many Bugs
2. Instability
3. Unsatisfactory Performance

1. **Too Many Bugs**

The reasons for an excessively high number of bugs are many, including: (1) over-complex initial design; (2) implementation not consistent with initial design; (3) inappropriate implementation; and (4) uncontrolled evolution. These points are elaborated upon below.

The design may be so complex that the system in fact cannot be debugged. This complexity can be introduced in a number of ways. The initial design concept may, for example, unduly require complex module interactions, extensive time-dependent characteristics, multiple and non-obvious uses of the same information by many parts of the system, etc.

Complexity can be introduced during modifications following initial design if the initial implementors, or the next generation of designers and implementors, either cannot understand the initial design, cannot make it work, or the initial design is simply unsatisfactory in some way. In such instances an approach sometimes followed is to “program around the problem”, leading to further problems in all areas of system performance, including possibly the undoing of some important initial design concept unknowingly.

A further cause of program complexity is due to the excessively “sophisticated,” “elegant,” “clever,” or “efficient” programmer. Most, if not all, systems operating today have sections that either never will work quite properly or, if they do, dare not be modified to meet new requirements, due to their initial implementation by a programmer with one of the above attributes.

Poor implementation is a very common source of bugs. The cost of finding and repairing problems caused by poor implementation is so large as to make it quite clear that proper project staffing, detail design reviews, and strictly enforced unit test requirements are of extreme importance.

A situation leading to the existence of a continually high bug rate sometimes occurs when a product undergoes an evolution process in which new features are introduced simultaneously with ongoing maintenance. In such instances, synchronization between new features and bug fixes becomes a problem. An approach sometimes encountered under these conditions is to create a “safe” local environment by minimizing dependencies upon other parts of the system, even to the extent of creating duplicate or similar local tables, functions, etc.

The problems in future maintenance and modifications are obvious.

2. **Instability**

The term “stability” will be used here to characterize the degree to which a job, once entered into the system, will terminate only due to a user error, and further, the effect of the user error is minimal (e.g., execution terminated, but files created to that time are not lost). If a user’s program frequently must be rerun due to problems not of his own making, or if the severity of the problem is not consistent with the severity and probability of the error, then the system may be said to be “unstable.”

Instability was not a problem in the days prior to multi-programming, as only one job was generally lost, and a rerun usually was successful. In multi-programming batch systems even extreme instability could be tolerated (and was) for a while, as a rerun often worked. In time, however, more capable I/O devices such as removable disks became available, with the attendant need for extensive support software. The major stability
problem then became one of file creation and retention. This problem remains in many systems.

With the advent of real-time and time-sharing systems, stability has become much more important. In such systems there are, typically, from 20 to 100 users simultaneously somewhere in the processing cycle (in core with the CPU, in core, swapped, etc.). Losing the execution state, and possibly the files, of this many programs, or of critical programs, is extremely serious.

The most significant factors affecting a system’s stability are the complexity of design and implementation, and the degree of attention paid by designers and implementors to automatic system recovery. Even good automatic recovery techniques cannot overcome problems inherent in an over-complex system.

3. Unsatisfactory Performance

The performance of a system may be unsatisfactory to the users. The performance of most batch systems is not so bad that users consider it intolerable, though it is still a major concern. The basic algorithms for obtaining satisfactory batch throughput are basically not complex, but as batch systems attempt to service an increasingly large community of hardware and users, the simplicity possible disappears. Thus, the major operating batch systems available vary widely in design, and in classes of service performed well. Batch systems with performance problems generally are troubled with excessive overhead, complexity due to the variety of user-types intended to service, difficulties in core management, and inadequate support for large data bases.

In the time-sharing area there have been notable instances where the performance was not what was required. A number of reasons exist for performance problems in time-sharing systems, for example, the basic assumptions were over-ambitious, not well thought out, or inadequately scoped (with modeling, for example). A more frequently encountered cause of unsatisfactory performance in time-sharing systems is the attempted retrofitting of a batch system to support time-sharing users.

Schedule delays

The most frequent cause of a missed schedule is, of course, the initial creation of an unrealistic schedule. Other reasons, generally related to the first at least implicitly, include ill-defined or missing requirements, changes in baseline assumptions, the customer interface relation, plus many of the causes of product and cost problems as described elsewhere in this section.

There are frequent instances in which the product was initially ill-defined and underwent a process of definition through its development. The conditions under which the initial estimates were given often change, including such changes in the available resources (money, skill, computer time). Requirements to support new hardware previously unenvisioned are often introduced.

The type of customer interface can have a major effect upon schedule. The customer interface is defined here as that person or persons who define, interpret, and control all requirements that come from the customer. The customer’s relevant technical and management experience, maturity, willingness to respond to suggested changes, understanding of problems, and readiness to assist in achieving satisfactory mechanisms of living together, are quite important. In particular, it is extremely important that the customer not require that all possible “bells and whistles” be included in his product. He should be reasonable in terms of tradeoffs, where considerable design and implementation simplicity can be gained by providing somewhat less function to the user. The customer must be convinced that simplified or limited functional capabilities will often improve the eventual performance and stability, and will increase the ability to add future enhancements to the system.

Excessive costs

A major software development problem for the developer, in addition to user-related problems, is cost. Major causes of cost over-run include:

1. Schedule Delays
2. Low Initial Estimates
3. Staffing Too Rapidly
4. Staffing With Quantity, Rather Than Quality
5. Follow-on Costs
6. Type of Contract

1. Schedule Delays

Delayed schedules generally lead to higher cost, unless the developer had the foresight to see the schedule was unrealistic and staffed according to a reasonable schedule.

2. Low Initial Estimates

Some excellent techniques for estimating project schedules and costs have been presented in the litera-
6 Persons making schedule and cost estimates must previously have been closely involved with projects of similar size and complexity. They must have accurate cost and progress summaries of the previous projects. It is useful to know the original estimates for previous projects, and eventual results.

The value of PERT-type techniques in making initial estimates is limited, due to the lack of detail concerning the project available at that time. A PERT network or the equivalent should be constructed, however, even if only on a very gross scale, as a means for forcing upon the estimators a mildly rigorous framework in which to place their estimates. Of particular importance in the PERT are, of course, those things that frequently lead to schedule problems. Foremost among these are a knowledge of: (1) dependencies outside the project; (2) the period required for initial staffing, defining the organization, and other activities required to have the people know what they are supposed to be doing, why, and when; and (3) those major activities within the project which may proceed in parallel, and which must proceed serially.

Machine time estimates are almost always underestimated, on projects of all sizes, and usually by a large factor (two to four). Even when machine time estimates are made by reasonable and experienced people, they are often made invalid by the eventual uncontrolled and wasteful manner in which the computer is used. Even today we find use of "block" or "hands-on" time on large, fast machines. Unless projects develop techniques for optimal use of computer time, including the support software required to remove the need (except in rare cases) for hands-on time, problems of cost, geographically separate computer installations, and inability to maintain schedules will generally arise.

Some activities in project development can proceed quite nicely in parallel by defining the major interfaces early. For example, a compiler can often be developed in parallel with an operating system, as its interface needs are small. Similarly, applications design and coding may often overlap development of the support system.

3. Staffing Too Rapidly

Staffing a project is an important and complex issue. There are never as many good people available as a manager would like. A manager is aware that a certain general level of manpower will be required to do a job, and then the staffing search begins, trying to identify people who are available, have relevant experience and skills, etc. Initial estimates that a project will need a certain number of programmers should not lead to the addition of "head count" simply to meet predictions. In general, projects staff too quickly with people anxious to produce code, and not quickly enough nor with enough skill with managers and supporting staffs (e.g., integration and test activities, described later). Programmers are often brought on board in order to start doing something before they really know what to do. A good generalization is to always have the minimum number of programmers possible on a project.

4. Staffing With Quantity, Rather Than Quality

No matter how insensitive, trivial, simple, or straightforward a piece of code is to write, it will cost more, often much more, if a poor programmer is given the job. Sometimes the cost is hidden, perhaps for years, but once programs "work" in some sense there is a great reluctance to replace them, which tends to extend the life of some rather poor examples of the programmer's art.

In such complex areas as systems or real time software, unless there is a very large amount of review talent available, people should not be brought onto the project unless they are extremely skilled. (It can be argued that this statement should be enlarged to include all types of programming.)

Relevant experience and resumes of people cannot be considered the most important criteria for hiring. Extensive experience cannot overcome the lack of an organized approach, insight, maturity in tradeoffs, and a willingness to make very sure that a program really works correctly.

It should not be assumed that key people will be available through hiring, or transfer from other parts of the company.

5. Follow-on Costs

Unrealistic estimates of follow-on costs after the first release of a system are often made. Follow-on costs may include maintenance, enhancements due to oversights in the initial product (frequently stability and performance problems fall into this category), user enhancements required to stay competitive, a need to support new hardware, etc. The larger a software system is the more reluctance there is to replace it. Thus the product life may be long, with a fairly continuous, and costly, development cycle.
6. Type of Contract

The basic types of contracts are cost-plus fee (percentage of cost, award, incentive, etc.) and fixed price. Virtually any type of contract is satisfactory if the contract administrators on both sides are both knowledgeable and reasonable. Cost-plus contracts are felt to be superior to fixed price contracts for all cases except the (relatively rare) instances in which the product, dependencies, etc., actually can be specified in great detail. For example, the author has observed in some detail a cost-plus contract with highly skilled technical and management interfaces, and a fixed price contract with a vague contract and inexperienced interfaces. The cost-plus contract could have been costly, but was not, due to the expertise of the contract managers. The fixed price contract in theory limited the cost, but in fact, due to extensive requirements above the scope of the contract, resulted in contract renegotiation, higher cost, schedule problems, ill will, and eventual replacement of contract managers on both sides.

There are, of course, instances of cost or product problems in cost-plus contracts, if, for example, expensive and not necessarily talented personnel are “dumped” onto the contract for profit and/or convenience. A sometimes-encountered problem in fixed price contracts is a “buy-in,” followed by either minimum possible contract interpretation or change of scope. Again, experience and reasonableness on both sides is required in order to prevent the contract type from perhaps heavily contributing to cost and other problems.

OPTIMAL PROJECT ORGANIZATION

This section describes a general project organization that assists in providing a check-and-balance relation in a project that will contribute to project visibility, control, and other factors important to meeting project goals. This organization, and the corresponding procedures of which examples are given, is felt to have wide applicability, but is not, of course, an exact model of any particular project. An organization will inevitably be shaped by the job to be done, people available, company policies, etc.

This section discusses the following topics:

General Project Organization
Project Managers
Development Activities
Integration Activities

Test Activities
Procedures
Staff and General Support Activities

General project organization

Figure 1 shows a project organization that it is felt is generally applicable for a wide range of development projects. Many of the functions shown in this figure are not, of course, required precisely as shown in particular projects, but the major functions implied, or their equivalent, are felt to be required in some form for projects on the order of five people or more.

The key element of the figure, and indeed of this paper, is the division of responsibility into separate functional groups, such as those termed here Development, Integration, and Project Test. This division is felt to be an absolute necessity. This division must occur early enough in the project life so that each functional group can adequately prepare.
The following sections describe the Development, Integration, and Project Test groups in more detail. The general responsibilities of these three groups are:

Development: Design, write, debug and unit test new code.
Integration: Integrate all code. Be the project "bookkeeper."
Project Test: Determine the status of integrated code.

It is critical that people who are developing the product not be simultaneously responsible for all the bookkeeping functions, integration of new code, and confirmation that the integrated code works. The project organization must be such that all groups have strong technical people, and receive full support from the project manager. Without an appropriate division of responsibilities, and a corresponding set of procedures, both visibility of true project status and control over development are very difficult to achieve. Among the typical problems that result are: not knowing what code is really in a particular version of the system; and a requirement to accept a programmer's statement for the status of a module ("90 percent done, only a few minor things don't work").

Project Managers

The three levels of project management discussed here are:

1. Project Manager
2. The Project Manager's Manager
3. First Level of Management Within the Project

One comment relative to all levels of management should be made before going into further details. Managers are assigned a responsibility, and must be allowed to make decisions within their assigned responsibility without undue requirements to justify their decisions. If a reasonable set of checks and balances exists (e.g., Review Boards), and if the managers chosen have appropriate experience and judgment, then their scope of decision making should not be frequently tampered with. Frequently no decision is worse than nearly any timely and moderately rational decision.

1. Project Manager

The most important single person on a project is the project manager. Therefore, choice of this person should be made with appropriate care. There are a wide variety of attributes that it is desirable to have in a project manager, and a set of attributes that if they exist in a project manager almost assuredly doom the project to failure.

The project manager must be technically strong enough to understand the important technical decisions made by those under him, to ask the right questions about those decisions, and to change them if needed. In particular, he must be able to cause the project to move toward realizable goals, and be able to say "no" to impractical ideas, and enforce it.

The project manager must have both technical and managerial maturity. All too often managers are chosen on the basis of strong technical skills plus a desire to be "promoted." Opposite extremes would be the assignment of a weak technical person to a managers role because nobody else would put up with the customer interface or personnel problems, or because the person just happened to be available.

Another characteristic that must be contained within the project manager is a desire to become involved. He must manage the project, not simply spend occasional time in reviews, believe all that is told him, and dilute his activities with marketing for new business or other company activities. It is always easier for a manager to assume that since good people are working in a certain area, and everyone is working hard, the end result will work out all right, and that he need not take the time to dig into that area in detail. On reflection, it almost always turns out that one or more areas do not develop satisfactorily, regardless of the skill of those most closely involved. Frequent reviews of design, implementation, test status, dependency status, schedule milestones, etc., by the project manager, are required.

It is not felt to be possible for a project manager to be in some sense a "pure" manager, that is, manage PERT charts, organization charts, staffing, facilities, and so forth, and allow others to make the larger technical decisions unsupervised. This would rule out software project managers whose experience is only in other types of work.

2. The Project Manager's Manager

In addition to the project manager being of appropriate technical and supervisory capability, it is also critical that his immediate supervisor have relevant experience, a set of both common and complementary skills, and not be so sure of the project manager that an independent check of status and problems is never
made. Every project manager needs someone reviewing his major decisions, a person to whom he may bring problems and receive real help, and someone to just talk to at times in order to remain objective about his project.

3. First Level of Management Within the Project

A superior project manager can, with appropriate organization and procedures, control up to 20-30 people satisfactorily with a set of managers reporting to him that are really "technical task leaders"—strong technical people with some supervisory capability, each supervising 3-5 people. Above the 20-30 level, however, support of full-time supervisory people within the project is required.

Development activities

Figure 1 shows a number of possible functions to be included within development. Many additional or different functions exist, of course, for any particular project. This is felt to be a reasonable example set, however. The particular activities to be described in this section are:

1. Major Development Function
2. Debugging Tools
3. Performance Measurement Tools
4. Dependency Consulting and Problem Diagnosis

Before beginning a discussion of the above, it should be mentioned that this paper does not attempt to discuss the merits of such very important development considerations as: software specification or implementation languages; types of debug tools; tradeoffs between speed, space, portability, flexibility, cost and schedule; the value of formal proofs; etc. The emphasis here is on functions that must somewhere be performed, not how they are best performed.

Two characteristics of the development process that frequently occur will first be briefly mentioned: the effect of design changes during implementation; and the occasional need to replace significant design or implementation efforts. A great deal of interpretation of the initial high level design takes place during detailed design and implementation. If not carefully controlled (via documentation and review) it is possible to develop a product substantially different from and/or inferior to that initially intended.

Occasionally it is realized that a significant mistake was made in design or implementation. The temptation always exists to "live with it," "fix it up later," or "program around it." Generally such approaches are a mistake—significant problems must be removed when discovered, or at least a plan for their removal created and begun. Too often the problems, if put off, remain in the system for a long period of time and cause far more difficulty than if remedied early.

1. Major Development Function

The major development function is, of course, responsibility for the overall design, detail design, debug, checkout, and internal documentation activities required for product implementation. Worth mentioning is the undesirability of fragmenting the responsibility for this effort in a manner that results in a manager with overall knowledge of the entire development not having full control over work for which he is responsible.

2. Debugging Tools

The creation of debugging tools is a distinct programming task, with substantial requirements to be defined, lead time needed for development, etc. In general there remains far too much dependence upon octal or hexadecimal dumps as the major aid to resolving difficult problems. Including appropriate (e.g., brief, symbolic) tools in a system initially is generally inexpensive; adding them later is quite difficult.

3. Performance Measurement Tools

Performance measurement tools must be included in development planning. This requirement needs planning, lead time, etc. The capture and processing of performance information is something that should go on very early in a system, but often does not. Obtaining early indication of performance problems is often critical, however, as their repair can require major surgery.

4. Dependency Consulting and Problem Diagnosis

Programming projects usually have significant dependencies on areas outside their direct control, for example, the hardware employed, support software, and communications facilities. Where such dependencies exist, it is best to assume that non-trivial problems will occur, and place in the project appropriate skills to isolate the problems and assist the appropriate support group in their removal. Since such problems often arise
due to misunderstandings resulting from inadequate documentation, or perhaps just a complex dependency, the ready availability of consulting assistance within the project will be valuable.

Integration activities

Put briefly, this group's activities are to provide a major portion of the independent control and visibility of development activities that is required. The Integration Group lies organizationally between the Development Group and the Project Test Group. This group receives new code from Development; integrates the various pieces of new code received from Development; performs the bookkeeping tasks on detailed status of all changes that have come in; assures proper approval levels; and other similar tasks, prior to passing on a new release for testing.

The control over development resulting from this sort of activity is obvious—nothing goes into the system unless properly identified and approved, and extensive documentation exists should the changes need to be altered (or, in extreme cases, removed). The visibility provided above an approach which has Development personnel alone report on system status is a bit less obvious, but critical. Development personnel tend to be optimistic about their code, tend to report a feature as fully provided even though only part of it is done, and to otherwise exhibit human traits. Integrations' job is to be more hard-headed—something is done only if the code was turned in, with appropriate documentation (e.g., updated internals), and a repeatable test sequence (without which the change is not accepted). Thus a project manager can know exactly what is completed.

It is becoming more and more common in the industry for the integration function to include very strong technical people, for such purposes as: (1) providing the ability to inspect new code submitted to the system for general technical correctness; and (2) identifying integration problems well enough to fix some, and if not to provide good information to Development people. If all the the project technical strength lies in other groups, the Integration Group must always borrow technical resources from these other groups, which they give up grudgingly. An intermediate approach is to assign people from Development to each new release, thus providing additional technical muscle for integration activities.

Another important function of the Integration Group is to perform some level of initial testing on the product in order to assure that the major new functions will work together and were integrated properly. Typically a small set of regression tests is run by Integration to assure that no major regressions have occurred prior to approving a new system for more extensive testing.

In addition to the above principal activities, a number of other activities may be convenient to include in the Integration Group. An item that may come under Integration is one that is termed here "scaffolding development." In the development of a new project it is always required that some subset of the system be generated first as an anchor on which the fuller system can be developed, that some other system be used as a vehicle for creating assemblies, load elements, etc. The term "scaffolding" is used to define the set of code and procedures that is developed to substitute for missing interfaces or missing subsystems in order that a partial system can run, or that phasing from one system to another goes smoothly.

Enforcement of such standards or requirements as linkage and naming conventions and memory utilization may also be conveniently performed by Integration. Automated tools may help such activities.

Test activities

This section describes a series of testing levels, and their place in the project organization. A fundamental corollary of this section is the now (hopefully) clearly established requirement that successful projects must include: (1) early creation of test plans, procedures, and staff; and (2) one or more groups established for testing a product that are organizationally distinct from and equal to the Development Group.

A testing approach that has proved quite successful is one in which there are a number of separate and distinct test activities, with minimal overlap, each fulfilling a needed function. A possible set of testing levels, chosen for purpose of example, is the following five level set:

1. **Unit Testing** by Development programmers. These tests show that the new function(s) operate in a private environment. Development managers must review the unit test plans in detail, as a poor job at this level will be very costly later.

2. **Integration Tests**, including rerunning of some of the above unit tests, testing the ability of the system to operate without major regression such that the next testing phase can concentrate on testing new code.
3. **Project Test.** This is the major test activity and is discussed in more detail below. This group runs extensive tests on new systems to confirm that new features work.

4. **Regression Test.** In addition to testing that new features work, one must also ensure that things that used to work still do. This may be done either within the project (e.g., by Project Test), or by an organizationally separate group. The regression testing activity differs somewhat from Project Test activities in that: (1) the volume of testing is larger; (2) the features tested are generally more stable than the new features; and (3) the regression testing group has more lead time, better documentation, etc., than the group testing new features, thus need not be quite as good “detectives” to determine what the new features really are.

5. **Field Test.** Regardless of the amount of prior test, non-trivial bugs always exist in the final release. For this reason a field test should be scheduled, in which the new system is mounted in one, or a few sites, and watched for a while, prior to full release.

The Project Test Group normally performs the following functions:

1. **Test Tool Generation**
2. **Test Plan Generation**
3. **Test Case Writing**
4. **Test Execution**

1. **Test Tool Generation**

Test tool generation requires both “hooks” in the system, and programs to process and analyze the results. To minimize the requirements for support upon the Development Group, it is advisable to develop test tools that are carefully designed to capture all information that is likely to be useful, but impose minimum space and complexity requirements upon the system. These tools should not be the last functions integrated. With early data definition, the required data processing programs may be developed in parallel with the system. Typical of the test tools that are required for development of modern systems are a means to easily and repeatedly run large series of programs. For example, a common practice is to simulate a large number of users in a time-sharing environment, with input coming from precatalogued files rather than on-line users, but seen by the system in a very similar way. These “prestored” jobs may contain test programs, a sample of real user programs, or both. Information captured from these test tools should record such information as is later needed to determine such things as response times, if or not the output compares on a line-by-line basis to previous runs, and so forth.

2. **Test Plans**

Extensive study of documentation and listings plus detailed planning is required to develop a set of test plans that may be reasonably executed and analyzed. Test plans must include enough detail so test cases are not written with dependencies on functions not yet integrated. Among the many types of tests that must exist are those for testing: functional capabilities; stability; performance; fallback (the ability for a system to be replaced by an earlier release if it fails); and the ability to support various hardware variations, including degraded mode operations in which only a subset of the hardware is available.

3. **Test Case Generation**

This is a difficult effort to do well if the test programs themselves are to: (1) not produce more bugs than they find; (2) efficiently and completely implement the test specifications; and (3) be self checking to a large degree.

4. **Test Execution**

Test execution includes the actual running of the tests, creation of files used for later analysis to determine if the tests ran properly, documentation of results, getting the information back to the Development Group and consulting on procedures used, and so forth. With automated test tools this effort can be reduced very significantly, with only a few people required to test features not easily tested automatically.

**Procedures**

The type of procedures required to assist in providing project control and visibility may be very large. This paper does not attempt to give a detailed set of procedures. The following list, provided as an example, gives information which would be expected to be
submitted by all Development programmers when their code is ready for Integration.

System level upon which the changes are based
System level intended for inclusion of the changes
Change identification
Change description
Listing of the changes
Card deck (or the equivalent)
Changed internal documentation
Changed operational documentation
Changed external documentation
Dependencies upon other changes
Modules changed
Copy of the unit test passed to confirm unit testing, and instructions for reproducing the test results in the integrated system
Approval signatures

For an example of support software to aid in implementing procedures of the above type, see the CLEAR system description of Reference 7.

A further example of a procedure usually needed is an automated planning aid of some sort. There are a number of reasons why a mechanized aid (such as PERT) is useful. These include:

1. A common basis for discussing project status is defined. Even if it is not quite everything that would be desired, everybody gets used to using it, and the shortcomings are overcome.

2. The mechanical difficulty of experimenting with changes in work required, people available, slipping dependencies, etc., is much less, as one need change merely a few data cards and rerun a program to get an initial feel for problems, tradeoffs, etc.

To meet the above objectives, the scheduling aid must have a number of attributes, including: (1) the output must be prepared by computer, not the local art department, as waiting a week to see the latest chart makes the whole thing useless; (2) the input preparation must be relatively simple; and (3) the process must be flexible enough and closely enough related to the projects' detailed activities to be a widely usable day-to-day planning tool (for example, listings should be produced that sort activities by start date, end date, person assigned, release version, etc.).

Staff and general support activities

There are a variety of activities that must exist within either the project or a supporting organization that do not always fall neatly into the Development, Integration or Project Test Groups. Those discussed here are:

1. Staffing Assistance
2. Customer Interface and Support
3. Review Board
4. Operations Support
5. Documentation
6. Status, Cost and Schedule Data Collection, Reporting

1. Staffing Assistance

In the early days of a project, when many people must be interviewed, resumes must be reviewed, and so forth, the project manager should not have to (and cannot) do all the recruiting for the project. It is also felt, however, that this job cannot be delegated completely to a personnel department. A good personnel department can be very valuable by performing such activities as obtaining candidate names and performing initial screening for quality and relevant experience. Technical men on the project must do all detailed screening, recruiting and staffing for the project, however. This is a difficult job to do well, as people range from uncommunicative geniuses to imposing charlatans. Nevertheless, detailed interviews combined with extensive checkups of references can assist greatly in minimizing the dead wood brought aboard. Realistically, it must be faced that more people must be hired than are needed, as some will not work out and must be discarded in short order.

2. Customer Interface and Support

A major element of the customer interface must, of course, be the project manager himself. This is so since he alone is responsible for the project, and therefore only he (or his manager, but hopefully not independently) can commit the project. It is often the case, however, that there is a need for frequent discussions between various people within the project and within the marketing organization, or other organizations working with the project, such that a continuous and heavy transmission of information back and forth across the project interface must exist. The project manager is generally too busy to both perform this
function and manage the project, so it is wise to set up a small group, perhaps one person, reporting to the project manager, to provide a continually available interface to the customer. This may include such customer support functions as assisting in preparation of rough external documentation and/or proposals for the customer, looking in advance at conversion problems when the system becomes operational, etc.

A further important role of the customer interface may be one of providing appropriate visibility concerning progress and problems. It is fairly easy for a project manager and a customer to both be working hard and think the other one knows what each is doing, but to discover that in fact major differences of opinion occur. A good interface can reduce these problems.

3. Review Board

A review board should be established that includes people who are generally not assigned full time to the project, but remain cognizant of technical activities going on in the project through their continuous participation on the board. This board includes senior technical people from wherever is appropriate. The purpose of this review board is to review major decisions either already made or that must be made before going on. This helps assure that all relevant experience available is brought to bear. Items reviewed should be of both technical and administrative (e.g., cost, schedule) nature.

4. Operations Support

Computer operators need someone to call on for help occasionally, efficient machine room procedures need to be developed that are generally rather different in project development from those in a typical production shop, etc.

5. Documentation

Documentation is always a problem, both where to place it organizationally as well as who should do it. There is a very wide variety of types of documentation to be done, usually far more than is obvious at budget time. Among the major types are, of course, the internal documentation for the system (usually done in Development, with the support from elsewhere). Another obvious type is the external documentation for the user. There is also the question of documentation to support people who must run the computer center, documentation for user managers, training documentation, etc.

One of the major unsolved problems of the computer industry is the creation of appropriate documentation, at a reasonable cost. It always seems to get done eventually, but is always painful to the programming staff, and not infrequently to users.

6. Status, Cost and Schedule Data Collection, Reporting

Projects sometimes begin with good procedures in place for creation of status, cost and schedule collection, etc., but as time goes on less and less use is made of such procedures. A frequent cause is the lack of staff to regularly collect and report upon the data collected. This can be a substantial job, but if done properly provides management with early indications of potential trouble areas.

SUMMARY

The proper management of programming projects requires means for visibility and control. To know what is actually going on in a project, to not have critical path activities slip, to head off such difficult-to-solve problems as memory over-use and slow performance, and with all this to meet a tight schedule and budget, is a challenge met by few. The primary attributes that must be contained in a project to provide good visibility and control are:

1. Division of responsibility in a manner that gives good checks and balances (separate groups for Development, Integration and Test).
2. A large set of procedures, rigidly enforced, that supplement the above organization by forcing an automatic flow of information.
3. Managers, supervisors, and task leaders at all levels that have good technical judgment, and will “dig” to find out the true status of all work.

REFERENCES

1 C King
   Systems analysis and project management
2 M E Conway
   Datamation Vol 14 No 4 April 1968 pp 28-32
3 R H Kay
   The management and organization of large scale software development projects
   AFIPS Conference Proceedings Volume 34 1969 SJCC pp 425-433
<table>
<thead>
<tr>
<th></th>
<th>R W BEMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Manageable software engineering</td>
</tr>
<tr>
<td></td>
<td>Software Engineering Volume 1 Computer and Information Sciences Academic Press 1970</td>
</tr>
<tr>
<td>5</td>
<td>Checklist for planning software system production</td>
</tr>
<tr>
<td>6</td>
<td>J E ARON</td>
</tr>
<tr>
<td></td>
<td>Estimating resources for large programming systems</td>
</tr>
<tr>
<td>7</td>
<td>H M BROWN</td>
</tr>
<tr>
<td></td>
<td>Support software for large systems</td>
</tr>
</tbody>
</table>