An Information Workstation for Software Managers

M. A. Kret
Bell Communications Research, Inc.

ABSTRACT

In the complex and changing environment of software development, it is imperative that software managers have current and meaningful information to support decision making. This article discusses a system that draws information from all phases of the software life cycle and analyzes that data from a software manager's workstation. With the analysis tools available at the workstation and data extracted from the various phases of development, managers can begin to form a model of the software development life cycle and measure the success or failure of software projects in quantitative terms.

1. INTRODUCTION

Currently, large software development environments are typically supported by a number of systems. There are planning systems, administration systems, marketing systems, systems that track user requests, systems that manage source code, deliverables and computer resources, and systems that track and prioritize action items and responsibilities. These systems support the day to day operations of software development. In addition to this primary function, the operation support systems often double as management information systems, and inevitably produce some form of reports.

Such reports have traditionally been insufficient for the following reasons:

- Data across operation support systems are neither centralized nor integrated.
- The information addresses the general management population and does not focus on a particular manager's needs.
- When a problem is detected, reports do not allow managers to probe deeper and investigate what caused the problem.
- Data is often "stale" and inaccessible when needed.

These were the kinds of problems that led to the development of an information workstation for software development managers at Bell Communications Research. The workstation extracts information from all phases of the software life cycle, and allows a manager to query, analyze, and interpret that data from a software manager's workstation.

The major benefit of the workstation over the operation support system reporting methods is that the information, which previously existed in a variety of reports, is made available in a centralized relational database management system. The system's databases contain management data, which includes trend and summary information. Managers can use canned queries or write ad hoc queries to suit their particular needs. If a problem is detected, further analysis can be performed either by writing additional queries, or by accessing the operation support systems through a direct interface. The information is current since it is extracted automatically at regular intervals, and it is accessible since the system is on-line. In addition, the system's distributed architecture enables managers to access the system at work, at home, or while traveling on business.

2. TIRKS™ MANAGEMENT AID

The software manager's workstation was originally developed to meet the information needs of managers on the TIRKS™ System at Bell Communications Research, and was given the name TIRKS Management Aid (TMA). The TIRKS System started in 1972 and was designed to track circuit orders and provisioning inventory for interoffice telephone service.

The TIRKS System has grown into one of the world's largest software systems supporting 23 Bell Operating Companies in the provisioning, operations, engineering, and marketing of the interoffice circuit network. The system as of September, 1985 was comprised of approximately 23,000 modules and over 10 million lines of source code. Managing a project of such magnitude has led to both challenging problems and creative innovations in controlling the software life cycle.

To control software releases the TIRKS System has developed a number of operation support systems. One of the first problems a large project has to tackle is product management. The TIRKS System has been known to ship as many as two major software releases, four maintenance releases, and numerous special purpose releases in each of four project lines in a given year. Thus, a system was developed to control the building of software environments both for testing and for integrating the entire software release that is ultimately shipped to the user. With development occurring on as many as 30 releases simultaneously, a second system was developed to track and manage source code for the development environment. In addition, each user request that comes in to the TIRKS Technical Support Center is logged into a tracking system and monitored throughout the software life cycle. During the testing phase several systems exist to monitor; 1) if testing is occurring, and 2) whether problems are being addressed in a reasonable time frame. The only phases of the software life cycle that are not supported by specialized systems are the requirements phase and the design phase. During these phases manual inspection methods are used to verify that quality requirements and designs have been completed.

To manage and control the software life cycle a number of support groups have been set up in the TIRKS organization and each of these groups have purchased or developed tools which complement the systems discussed above. Examples are project management tools which help plan releases and track milestones using PERT and Gantt techniques, tools developed by the performance group to monitor our users' performance levels and project capacity planning for user hardware, and analysis tools used by the planning groups to estimate the resources required for work in the coming year. By now it should be evident that an entire assembly line of support systems and tools can be used in the large software development environment to control the software life cycle.

When software projects are small the manager can usually talk to a few developers and come away with a reasonably accurate status of the project, assuming the manager is a good judge of character. As projects grow and the organizational heirarchy within a project reaches two, three and four levels deep, this method becomes impractical, if not impossible. Inevitably at this point in a large software project's history managers encounter a situation where information is desperately needed, but not available. The organization then recognizes the wealth of information in the operation support systems discussed earlier. Subsequently, extracts are taken from support system databases and reports are written in the absence of long term information planning.

There are several problems inherent in this reactive method to developing management information systems. Resolving these issues is the main focus of the software manager's workstation. The first issue is that a management information system should not be designed as a reaction to a crisis since these systems lack the flexibility to meet future information needs. The second issue is that in some organizations all levels of management receive the same reports. Since each manager's information needs are different, generalized reports both hide relevant information and restrict the manager from focusing in on areas of responsibility. A third issue involves the representation of the data provided by management information systems. Presently, reports are often geared towards the operation support system databases. Such a representation does not illustrate trends in the software life cycle. Finally there are the more general issues surrounding the value of the data made available by information systems. 1) The data must be current, 2) the system must be available not only at the office, but wherever the manager needs to use the system, and 3) the data must address those items which management defines as critical to controlling the software life cycle. This means that the system should not provide the information which is most readily available, but the information that provides managers with greater control.

The first step towards resolving these issues is to analyze the information needs of software managers. This occurred early in the development of TIRKS Management Aid through a series of interviews with managers of various levels. The managers were asked to prioritize their information needs and these priorities were used to guide the implementation schedule. The design of the system was reviewed by both managers and the architecture design group to insure that the system met the long term information needs of the organization. No specific reports were designed. This function is delegated to each manager through the use of a relational query language. For those unfamiliar with the query language used by the workstation, a set of canned queries were provided with the system and an individual on the TMA project was made available to help managers modify the canned queries to meet their particular needs. The data extracted from the various operation support systems is geared specifically towards management. It is gathered from all phases of the software life cycle and is summarized to reflect trends in the cycle.

One unique aspect of TMA in the area of decision

support architectures is its workstation/host distributed architecture which allows managers to take the system with them as a stand alone workstation and later log into the host to refresh the data at the workstation through the use of the host database management facilities. In addition to the decision support functions, which will be discussed in the next section, the workstation/host distributed architecture also lended itself to a number of office automation functions, like sending mail, word processing, and action item tracking.

3. FUNCTIONS

3.1 Introduction

The information workstation supports four basic functions; a query facility, an analysis feature, a support system interface and an extract capability. This section will describe each function in detail. Figure A illustrates the relationship between functions.

3.2 Query Facility

The query facility is the backbone of the software manager's workstation. It allows managers to access data indicating the progress of the software life cycle and analyze that data using the functions at the workstation. Queries can be written at the workstation using a text processing system and sent to the host processor for execution. The host relational database management system executes the query and returns the result to the workstation where spreadsheet, graphics, and presentation utilities are available. Managers can construct their own ad hoc queries or take advantage of the canned queries supplied with the system.

The query facility provides the workstation with the flexibility it needs to keep up with the changing software development environment. Queries can be used both to extract data and to create and modify the structure of relational tables. Software metric tables measuring indicators in the software environment can be created quickly and easily as the information relevant to software managers changes. The security associated with a given user is also controlled through the query facility. If a user is not authorized to view or update a particular subset of data the query facility will reject the user's request.

At Bell Communications Research we started out by defining software metric tables to measure the following: user requests and performance, work item scheduling, software development, and quality assurance. Table 1 shows the layouts for the metric tables. Each table is updated nightly, thus building a trend of activities. In addition to the metric tables, an activities and milestones table is extracted from project management to correlate trend data to planned milestones.

FUNCTIONAL ARCHITECTURE

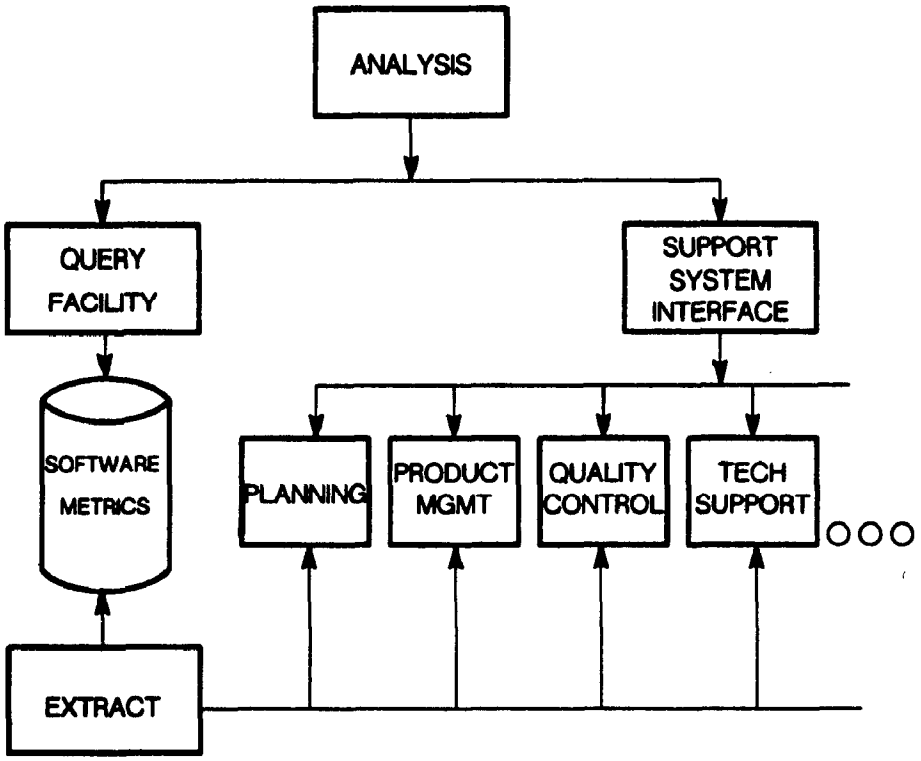

FIGURE A

The combination of an easy to use query facility and a flexible set of workstation functions led to some interesting results. We found that our users, some of whom are non-programmers, were developing their own canned reports and passing them on to other users. If the queries provided analysis useful to all managers, they would be integrated into the workstation's predefined analysis function. In other words, our users not only provided us with requirements by defining the reports, but they had also developed the software by packaging the query and the presentation of the result.

3.3 Analysis

The analysis feature is designed to detect and highlight in advance potential problem areas in the software life cycle. Input to analysis includes the following: the software metrics data measuring the development process, support system data such as planning and administration, and thresholds set by the managers themselves. The inputs are processed by a set of queries that measure those items which are critical to the project's success. Each time analysis is run these critical factors are measured and the results are presented using the workstation's graphics and spreadsheet capabilities. If a threshold is being violated, the manager is notified through the workstation's bulletin board feature. When project priorities change the queries can be modified to reflect the changes without updating the analysis programs.

Analysis is broken down into four functional areas; user analysis, organization analysis, subsystem analysis, and release analysis. User analysis tracks the status of the project as it relates to each individual client. Items such as software quality, performance, and the status of maintenance and enhancement requests are monitored. Organization analysis pertains to the quality, productivity, budget and personnel statistics of each organization in the project. Subsystem analysis concentrates on the status of each of the piece-parts of a large software system. Finally, release analysis tracks the progress from beginning to end of a new version of the system.

A typical analysis screen appears in Figure B. The analysis feature was developed at Bell Communications Research using the Framework integrated software package (Framework is a trademark of Ashton Tate). The Framework menu options appear along the top of the screen. These options control the spreadsheet, graphics, filing and word processing functions offered by the integrated package. The TMA menu is illustrated along the bottom of the screen and allows the manager to access each of the four analysis categories as well as a threshold update function. A bulletin board exists for each type of analysis; user, organization, subsystem, and release. Releases can be analyzed in combination with either users, organizations or subsystems.

Bulletin boards notify management when thresholds are not being met. Within each analysis is a set of categories. These are the factors which management has defined as critical to the success of the project. Associated with each category is an index. The index represents a measure of its corresponding category. The indices are calculated by taking the thresholds set by managers and querying the software metric tables to determine if the thresholds are being violated. A date and time stamp is placed on each bulletin board stating when the analysis process was last run.

If the manager discovers a problem in any of the analysis areas (evidenced by a growing index) he or she can use menu options to probe deeper. These options will step the manager through the calculation of the index as shown in Figure C. From here the manager can observe

SOFTWARE METRIC TABLES

USER REQUEST TABLE

DATE	-	DATE EXTRACT WAS TAKEN
USER	-	USER IDENTIFICATION
RELEASE	-	USER'S SOFTWARE RELEASE LEVEL
TYPE	-	TYPE OF REQUESTS, SOFTWARE, DOCUMENTATION, ENHANCEMENT
UNDER_INVEST	-	NUMBER OF REQUESTS UNDER INVESTIGATION
UI_AGE	-	AVERAGE AGE OF REQUESTS UNDER INVESTIGATION
BEING_FIXED	-	NUMBER OF REQUESTS BEING FIXED
BF_AGE	-	AVERAGE AGE OF REQUESTS BEING FIXED
NO_CHANGE	-	NUMBER OF REQUESTS REQUIRING NO CHANGE, i.e. NO DEVELOPMENT EFFORT
COMPLETED	-	NUMBER OF REQUESTS COMPLETED

USER PERFORMANCE

DATE	-	DATE EXTRACT WAS TAKEN
USER	-	USER IDENTIFICATION
AVG_CPU	-	AVERAGE PROCESSOR UTILIZATION
BH_CPU	-	BUSY HOUR PROCESSOR UTILIZATION
AVG_RESPONSE	-	AVERAGE USER RESPONSE TIME
BH_RESPONSE	-	BUSY HOUR USER RESPONSE TIME
TRANSACTIONS	-	DAILY TRANSACTION VOLUME
BH_TRANS	-	BUSY HOUR TRANSACTION VOLUME
ABENDS	-	NUMBER OF ABNORMAL PROGRAM TERMINATIONS

WORK ITEM SCHEDULING

DATE	-	DATE EXTRACT WAS TAKEN
RELEASE	-	SCHEDULED SOFTWARE RELEASE
ORG	-	RESPONSIBLE ORGANIZATION
TYPE	-	TYPE OF WORK ITEMS, SOFTWARE, DOCUMENTATION, ENHANCEMENT
DEV	-	NUMBER OF WORK ITEMS IN DEVELOPMENT
ST	-	NUMBER OF WORK ITEMS IN SYSTEM TEST
COMPLETED	-	NUMBER OF WORK ITEMS COMPLETED

SOFTWARE DEVELOPMENT

DATE	-	DATE EXTRACT WAS TAKEN
VERSION	-	VERSION CONTROL IDENTIFICATION
ORG	-	RESPONSIBLE ORGANIZATION
EDIT_MODS	-	NUMBER OF MODULES BEING EDITED
EDIT_CODE	-	LINES OF SOURCE CODE IN MODULES BEING EDITED
DELIV_MODS	-	NUMBER OF MODULES DELIVERED TO TEST ENVIRONMENT
DELIV_CODE	-	LINES OF SOURCE CODE IN MODULES DELIVERED TO TEST ENVIRONMENT

QUALITY ASSURANCE

DATE	-	DATE EXTRACT WAS TAKEN
RELEASE	-	SOFTWARE RELEASE BEING TESTED
ORG	-	RESPONSIBLE ORGANIZATION
TESTS	-	NUMBER OF TESTS EXECUTED
OPEN	-	NUMBER OF OPEN TEST PROBLEMS
OPEN_AGE	-	AVERAGE AGE OF OPEN PROBLEMS
OPEN_CRIT	-	NUMBER OF CRITICAL OPEN PROBLEMS
CRIT_AGE	-	AVERAGE AGE OF CRITICAL OPEN PROBLEMS
CLOSED	-	NUMBER OF CLOSED TEST PROBLEMS

TABLE 1

if the problem has been a trend by selecting a graph of the software metric, Figure D, or the user can enter the operation support system where the metric was extracted to investigate the details of the situation.

A second look at the example will illustrate how a manager might use the analysis feature. The manager sees that the organization's maintenance index is getting high (86), see Figure B. By selecting the appropriate menu options the manager arrives at Figure C, which is the calculation of the organization MR (Maintenance Request) index. There are more MRs under investigation than the threshold predicts (40), and there are fewer requests being fixed than expected (46). These two factors account for the maintenance index (86). Finally, the manager can display the data in graphical form to determine if the problem has been a trend. Figure D shows that even though there are more MRs under investigation than the manager would like, the trend in maintenance requests under investigation is decreasing.

To define the critical success factors (CSFs) managers from the assistant vice president to the district manager level were interviewed. These managers were primarily concerned with the production side, as opposed to the administrative side, of software development. They most often cited "responding to user maintenance and enhancement requests in a reasonable time frame" as being the most critical item. As a result this item became one of the factors monitored by the analysis process. Through the course of the interviews a number of critical success factors were defined and incorporated into analysis. The critical success factors defined during the interviews included producing a high quality product, delivering software releases on time, and maintaining an acceptable level of user performance. Each of these factors were broken down into measurable items and appear in Table 2.

The next step was to create an index to measure each of the critical success factors. As an example, a maintenance index was defined based on the number of maintenance requests under investigation, and the number currently being fixed. Since all CSF indices are presented on a single analysis screen, managers can quickly monitor the status of those items that they have defined as critical. Managers can also selectively monitor and set custom thresholds for a particular user, organization, subsystem, or release, resulting in greater control in tracking a particular area of responsibility. By gathering this information and attempting to define reasonable thresholds, managers gain insight into the nature of the software life cycle.

3.4 Support System Interface

The purpose of the support system interface is

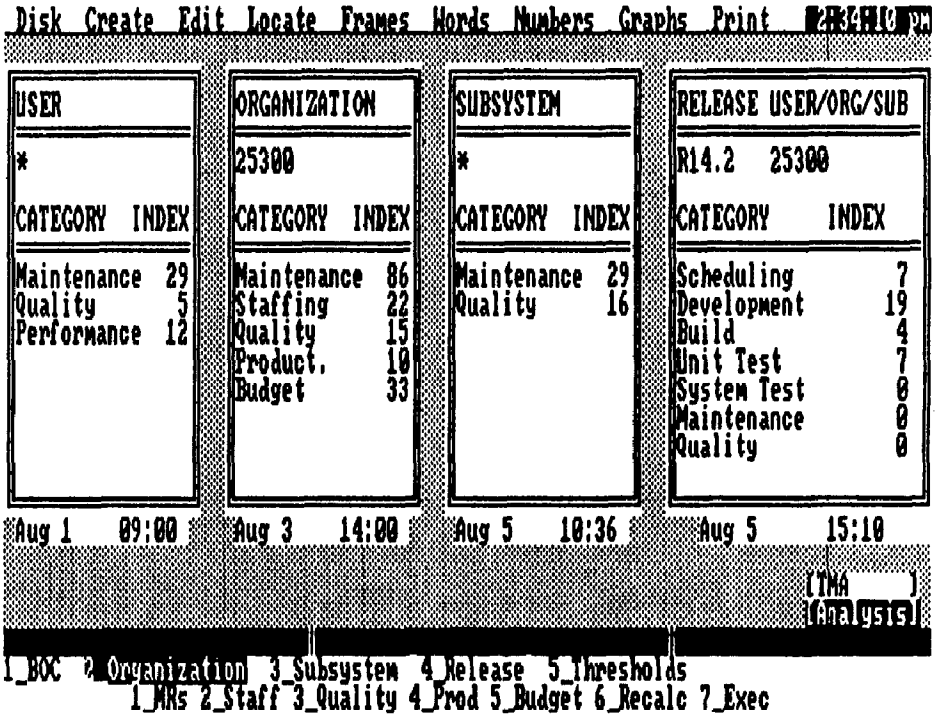

FIGURE B

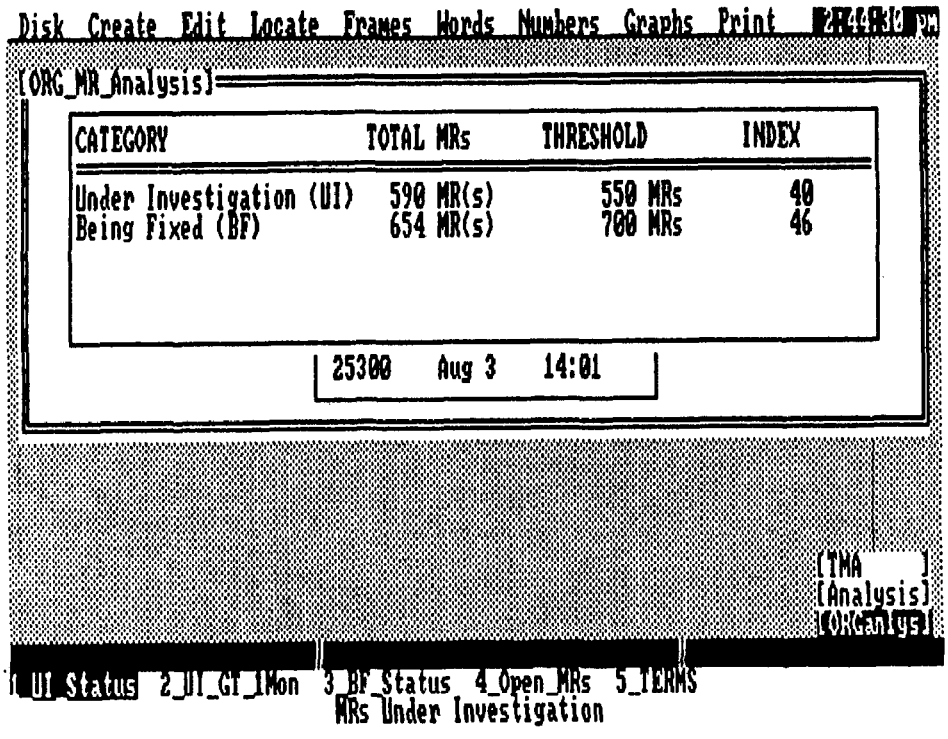

FIGURE C

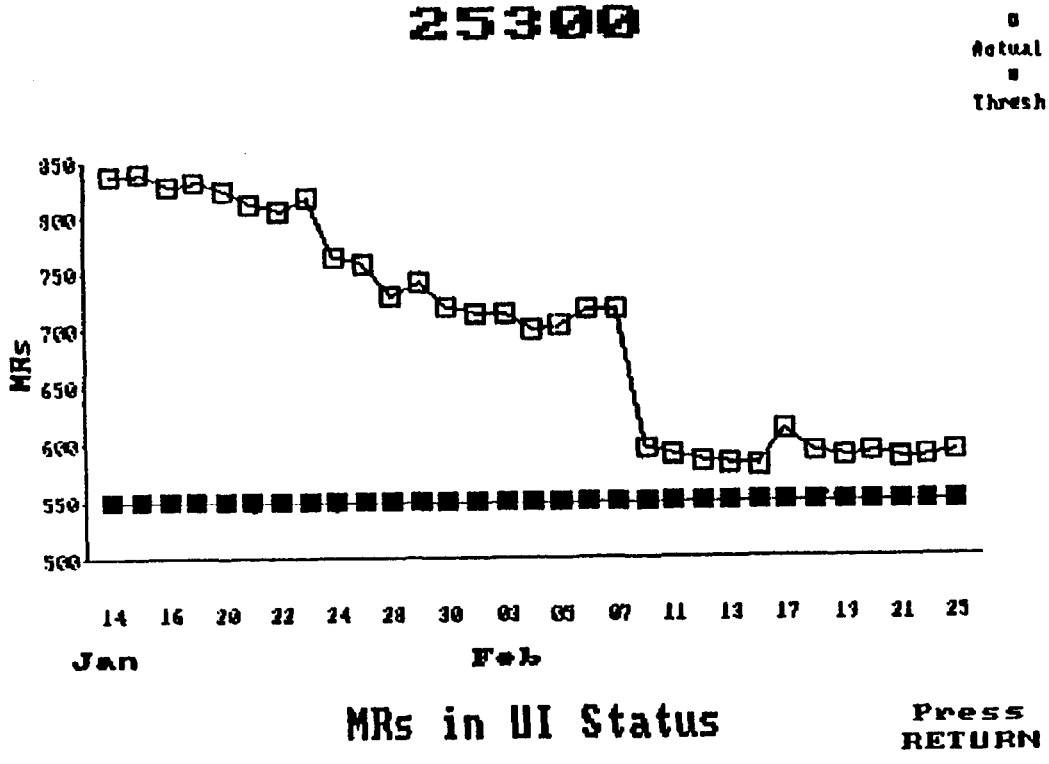

FIGURE D

CRITICAL SUCCESS FACTOR ANALYSIS

CRITICAL SUCCESS FACTOR	SOFTWARE METRIC
1. RESPONDING TO USER ENHANCEMENT AND MAINTENANCE REQUESTS IN A REASONABLE TIME FRAME	1.1 THE NUMBER OF REQUESTS CURRENTLY UNDER INVESTIGATION
	1.2 THE NUMBER OF REQUESTS CURRENTLY BEING FIXED
2. MAINTAIN AN ACCEPTABLE LEVEL OF QUALITY	2.1 THE RATIO OF USER MAINTENANCE REQUESTS TO NEW AND CHANGED LINES OF SOURCE CODE
	2.2 THE RATIO OF FAULTS FOUND DURING QUALITY ASSURANCE TO NEW AND CHANGED LINES OF SOURCE CODE
3. MEETING SCHEDULES	
3.1 COMPLETING DESIGNS ON SCHEDULE	3.1 PERCENT OF SUCCESSFULLY COMPLETED DESIGN REVIEWS
3.2 COMPLETING CODING ON SCHEDULE	3.2 TRENDS IN NEW AND CHANGED LINES OF SOURCE CODE
3.3 COMPLETING TESTING ON SCHEDULE	3.3 TRENDS IN TEST VOLUME AND TEST TO FAULT RATIO
4. MAINTAIN AN ACCEPTABLE LEVEL OF PERFORMANCE	4.1 PROCESSOR UTILIZATION 4.2 AVERAGE RESPONSE TIME 4.3 TRANSACTION VOLUME

TABLE 2

to allow the workstation to communicate with all existing operation support systems, e.g., planning, administration, etc. The following scenario illustrates why this requirement is so important. A high level manager observes from the workstation that user maintenance requests are not being responded to within the limits of the current thresholds. He or she discovers that the problem exists predominantly in a new and volatile subsystem. The manager of that subsystem is contacted and after some investigation concludes from the trend data that the requests have not been addressed due to heavy development efforts in recent months.

At this point the manager understands the problem. Yet to solve the problem the manager needs to access the system that tracks user requests. The workstation's metric tables store statistics regarding user requests, but to get the details of each request requires a link to the support system designed for that purpose. The feature is reduced to an architectural issue since what is required is that the workstation hardware must be capable of accessing all support systems in the environment.

3.5 Extract

The purpose of the extract function is to take data from the operation support environment, to summarize that data, and to make it available to the information workstation in a relational database management system. Data are extracted at regular intervals, allowing managers to observe trends in the various measurements.

It is important when defining the extract tables not to repeat what already exists in the operation support systems, but to summarize by priority, status, type etc. As an example, suppose we were extracting from a support system that tracked user maintenance requests. We would not extract data pertaining to a particular request. Instead, counts of high priority items by user group, or open status requests by development organization, would be more relevant to managers. If a manager needs the details of a particular request the support system interface can switch the user into the maintenance request support system.

The extract process consists of three steps; the raw extract from the operation support system, a summarization step, and the load of the relational database management system. The details of the raw extract will vary between support systems and will inevitably require some programming effort. Basically, items pertaining to status, priority, type and duration are extracted from the operation support system database and copied to a temporary area.

Next, the summarization programs calculate totals by user group, organization, subsystem, and software release. Finally, the load step takes the summarized data and appends it to the appropriate relational table. This entire process is automatically executed nightly without manual intervention.

4. ARCHITECTURE

The primary objective of the architecture is to support two basic functions. The first is to access and manipulate data that indicates the progress of the software life cycle. The second is to provide a flexible set of tools to query, analyze and interpret the information.

These objectives pointed to a workstation/host architecture for several reasons. 1) The data had to be centralized to integrate information from the various operation support systems. 2) Users were interested in focusing on a specific subset of data and updates would be minimal. 3) The functionality at the workstation had to be personalized, flexible, portable, and easy to use for the management community to accept the system.

Both large scale relational database management systems and personal computers were finding their way into the corporate environment at about the time we began to implement the software manager's workstation at Bell Communications Research. The two emerging technologies appeared very attractive in accomplishing the objectives stated earlier. The only drawback was their lack of integration. Most installations were implementing file transfer mechanisms to bring corporate data to PC functionality, but this approach did not isolate the manager from the PC/host communication process. Typically users had to log on to a host session, manually establish a communication link, initiate the file transfer routine, and then convert the data at the PC into a format compatible with the popular PC packages available.

The software manager's workstation architecture accomplishes integration, not through file

transfer, but by allowing the PC to directly submit queries and commands to a host system. In addition, a common area of storage is provided which can be accessed by both the host and the PC. The shared storage looks like a disk drive to the PC, but actually resides on the host. As a result, host programs can update the shared storage, making information available to the PC in a format it can recognize. This brought the power of the relational database engine on the host to the expanding functionality of Personal Computers, and at the same time isolated managers from PC/host communications. Figure E illustrates the concept. Basically, the architecture consists of two parts, the workstation architecture and the host architecture. Linking the two is the workstation/host communication process.

A combination of vendor software packages and Bellcore developed software was used to implement the system. Vendor packages were evaluated and selected to support the relational database management function on the host, the intergrated analysis functions on the PC, and portions of the communications interface. Functions developed at Bellcore included; the user interface, the support system interface, the extract process, and a facility to submit SQL queries from the PC to the host.

5. CONTROLLING SOFTWARE PROJECTS

It is no surprise that software metrics begin to get attention just when a project becomes too large to control by interviewing developers. Control is critical to the success of software. Yet software managers are often unsure about what to measure and how to use the data to better control the software life cycle. In Bellcore's experience with developing a software manager's workstation we discovered that five steps were necessary before software metrics could become useful in controlling the software life cycle.

The first step is standardization. There is little benefit in measuring the software life cycle in a project that does not develop standard design, implementation and testing practices. If each designer is using a different design methodology, it is difficult to develop an aggregate metric which accurately indicates the size or complexity of the many designs which make up a single release of software. If programmers are using different languages or have radically different coding practices, there is no sense in using lines of code as a measure of productivity. Likewise, if testers are using different testing methods, or if they are testing functions of varying complexity, then simply counting the errors found during the test phase is not a good indicator of the quality of the product.

If standards for design, implementation, and testing are developed then metrics can be defined which measure like objects, resulting in

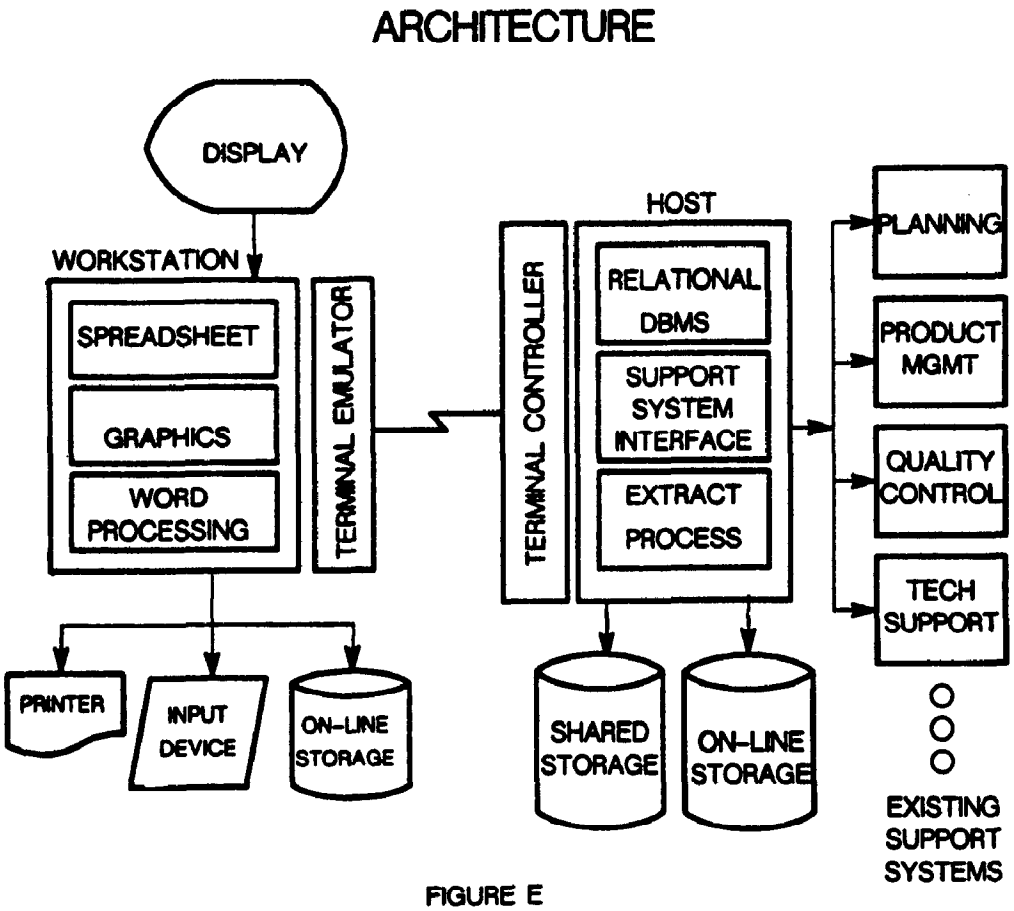

FIGURE E

more meaningful software indicators. Steps towards achieving this goal include rigorous design, code, and test plan reviews to insure that standards are being followed, defining a standard programming language and programming practices, and developing standardized manual and automated test scripts. It may seem restrictive, in light of the many hardware and software technologies available, to define a standard software development environment, but the result may mean more accurate measurements and greater control. In addition, standardization should not preclude experimenting with new technologies, but experimentation should take place within a controlled environment. Once new technologies are proven in the controlled environment they can be introduced as part of the standard development environment.

The second step is the characterization of work items. Here again, the objective is to measure like objects. By characterizing work items by size and complexity, managers can begin to predict the effect a work item will have on the software life cycle. Intuitively speaking, there is no utility in comparing a major enhancement's effect on the software life cycle with that of a minor maintenance item. Many schema for characterization are possible and a viable scheme will probably require fine tuning over time. One possible scheme might be to characterize a work item by its expected effect on software metrics in the life cycle. As an example an organization might characterize a minor maintenance work item as one that can be resolved by a single development organization, i.e., requires no interfaces, will affect a single module and will take between 10 and 25 lines of code to make the fix. In this case interfaces, modules and lines of code would be measurements kept in the software manager's workstation. The kind of information required to make this characterization is available when the work item is reviewed and can be used to predict not only the resources required to make the fix, but also the effect the fix will have on the software life cycle. This topic will be discussed further in step five, the modeling phase. Of course more complex work items will require more complicated characterization schemas, but here again, the tradeoff is loss of control since the alternative means that managers would be unaware of the nature of the work going on in the project.

The third step is measurement selection. As part of the measurement selection process the organization must first step back and take a hard look at their information needs. The organization should ask what items are most important to software managers and how might these items be quantified. At Bellcore we used a technique called Critical Success Factor(CSF) Analysis to determine which items were most important. Table 2, referenced earlier, shows some of the CSFs defined at Bellcore and the metrics used to quantify those CSFs. Often a single metric is not a good indicator of a success factor and in these instances a combination of metrics are used. The table also breaks high level CSFs into subfactors in order to reduce the item to a level that is measurable.

Once the software metrics are made available a fourth step of assimilation and experimentation begins. This is the most critical phase since it ultimately determines whether or not the managers will accept the system. One would assume that if the data defined in step three is "critical" to the success of the project that managers would be quite anxious to get their hands on the system. This is not necessarily the case. One paradox is that if the information paints a picture of the software life cycle that is drastically different from the organization's expectation, the organization may be inclined to question the validity of the data rather than their expectations. The paradox lies in the fact that this is precisely the data which is most valuable since it tells the organization something it was not previously aware of. There may also be a misunderstanding in how the metric was calculated and this too can lead to skepticism about the integrity of data. Another roadblock in the assimilation phase is that many managers have done quite well by polling their workers and peers for information. These managers do not see a pressing need to quantify information that they feel they are already aware of at an intuitive level. To compound matters, managers must usually learn to use a query language if they are to experiment with the data to form models of the software life cycle. Often there is simply not enough time to accomplish training and proceed with experimentation. Canned queries can be constructed to simplify the process, but these queries do not lend themselves to the kind of "what if" analysis required to form a working model of the software life cycle. Finally, there are the political issues involved when choosing software metrics since these metrics will be used to make judgements about the status of the organizations in the project.

There are several things that can be done to ease the introduction of a software manager's workstation. The first and most effective way to introduce the system is to open an information center and staff the center with an inquisitive individual who knows the software life cycle and how to use the query and analysis facilities at the workstation. The information center should perform two functions. First, it can supply managers with answers to specific questions, such as what is the trend in user maintenance requests for my organization. Second, the information center should publish reports on a regular basis that are of interest to the project as a whole. By performing these two functions, the information center will allow the organization to focus in on the data at the workstation rather than the mechanics of the system. Soon, the same individuals will be coming to the information center day in and day out. At this point it

makes sense to train these individuals and work with them closely to insure they get the most out of the system's capabilities. As the number of individuals in this category grows, it may make sense to form a user group to review the importance of certain metrics, form information models and suggest workstation enhancements. Certainly, if the data has value and the managers in the user group appear to be getting an "information edge" over the other managers in the project, the software manager's workstation is well on its way to wide spread use.

Once the system is accepted the next step is to form a model of the software life cycle. The purpose of the model is to better control the software life cycle, in particular, the planning, monitoring and analysis functions.

In an established software project, software development follows a fairly consistent cycle. Users make requests for new systems, or for enhancements and maintenance to existing systems. Requirements are defined, resources are allocated, and a design is prepared and reviewed. Next, software modules are coded and tested in an isolated development environment. The entire software release is then integrated and tested as a single product. Finally, a quality assurance group reviews, tests, and signs off on the software. While software development is taking place documentation and training packages are being prepared for the users.

The ultimate goal in constructing a model of the software life cycle is to evaluate the impact of work proposed by the user on each of the phases in the cycle. An accurate model can aid in planning release schedules, determining the resources required to do the work, estimating the size and quality of software releases, and analyzing bottlenecks during development. The model really begins in step four with the experimentation process. During this step managers begin to find correlations between variables and test whether or not these correlations hold true for several releases of software. As an example, an organization may discover that the number of faults found in the quality assurance phase may correlate with the number of new and changed lines of code in the implementation phase. This allows the manager to use an indicator from an earlier phase in the release cycle to predict what will occur in a later phase. If the quality assurance manager has some idea of the number of faults each of his testers can process within the testing phase, that manager can begin to estimate the impact the release will have on the testing organization. In actuality, many models may emerge since variables may correlate highly in some instances, but not in others.

In addition to correlations, a number of rules of thumb eventually become evident. As an example in the TIRKS System we look at the trend of faults found in system test very closely. By the shape of that curve we can estimate the length of time required for testing to produce a quality product. If the trend in the fault curve does not level off within a given time frame, then the time period required for testing must be increased. This can often have an effect on the ship date of the release.

Over a period of time these correlations and rules of thumb begin to form a model of the software life cycle. The emerging model can be a valuable asset to the software manager in terms of both increased awareness and greater control.

6. CONCLUSION

In summary, an information workstation designed to support software managers should accomplish the following:

- Integration of data across support systems (planning, quality assurance, administration, etc.).
- Provide flexible tools to query and analyze the information.
- Allow managers to verify the data presented by accessing the support systems which are the source of the data.
- Provide summary and trend data so managers can plan future projects based on an analysis of past software development experiences.

At Bell Communications Research we have implemented TMA and are using it to track and plan software releases. Presently, a history of curves modeling the phases of the software life cycle are being stored. Our next endeavor is to test how the indicators in the various phases interrelate. By examining the correlations between indicators from the various phases of development we are better equipped to predict and plan the software life cycle.

As a result of this effort TMA is providing insight into how software is developed in our environment. Our goal is to eventually capture this insight in an expert system that will plan and track the software development process.

7. ACKNOWLEDGEMENTS

The software manager's workstation developed at Bell Communications Research was a result of the talent and efforts of the following individuals: Bob Clair, Yu Mei Chu, Jeff Chern, George Harrison, Al Kendziora, Frank Marchese, Ray Neival, Gary Schneider, and Jeff Wallace.

I would also like to thank Mike Geary, Jerry Kaplan, Peter Vellotti and Joan Vigliotta for their support and comments.

REFERENCES

1. Alter, S. L. Decision Support Systems: Current Practice and Continuing Challenges, Addison-Wesley, Reading, Massachusetts, 1980

2. DeMarco, T. Controlling Software Projects, Yourdon Press, New York, New York, 1982

3. Fox, M. S. "The Intelligent Management System, An Overview," Process and Tools for Decision Support, H. G. Sol (editor), North-Holland Publishing Co., 1983

4. Henderson, J. C. and Schilling, D. A. "Design and Support of Decision Support Systems in the Public Sector," MIS Quarterly, Vol. 9 No. 2, June, 1985

5. Rockart, J. F. "Chief Executives Define Their Own Data Needs," Harvard Business Review, March-April, 1979

6. Shank, M. E. and Boynton, A. C. and Zmud, R. W. "Critical Success Factor Analysis as a Methodology for MIS Planning," MIS Quarterly, Vol. 9., No. 2 June, 1985

7. Sprague, R. H. and Carlson, E. D. Building Effective Decision Support Systems, Prentice Hall, Englewood Cliffs, New Jersey, 1982

8. Thierauf, J. R. Decision Support Systems for Effective Planning and Control, Prentice Hall, Englewood Cliffs, New Jersey, 1982

9. Yourdon, E. Managing the System Life Cycle, Yourdon Press, New York, New York, 1982