Copyright © 1982 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 61, No. 6, July–August 1982
Printed in U.S.A.

Automated Repair Service Bureau:

Cable Repair Administrative System

By P. S. BOGGS and J. R. MASHEY

(Manuscript received July 13, 1981)

The Cable Repair Administrative System (CRAS) is a recent addition to the Automated Repair Service Bureau (ARSB) that helps extend ARSB usage beyond Repair Service Bureaus into the Outside Plant Maintenance organization. The CRAS system helps this organization to schedule repair forces efficiently and to locate the trouble-prone outside plant facilities that are most in need of rehabilitation. This paper includes an analysis of the unusual features of CRAS, and comments on its development under rapidly changing circumstances.

I. INTRODUCTION

The CRAS system helps Outside Plant Maintenance (OSPM) managers to identify people and plant items that need attention on a priority basis. For example, CRAS helps managers schedule repair forces efficiently and locate organizational trouble spots in the complex process of OSP repair. The CRAS system also helps locate those cables or terminals that, over periods of time, cause an excessive number of customer troubles or especially high repair costs. Thus, repair money can be directed where it is most effective.

This article gives the background for CRAS and describes its goals, usage, and architecture. It also comments on the design approach used, unusual features included, and lessons learned from its development.

II. HISTORY AND STATUS

For several years, both AT&T and the Bell Operating Companies (BOCS) wanted a system that combined and improved on the cable trouble analysis features of the Computerized Cable Upkeep Analysis Program (CCUAP) and the manual Cable Repair Force Management

Plan (CRFMP). The CCUAP system was designed to help determine the nature and location of cable troubles, but lacked flexibility. Also, it could not accurately account for the hours worked on troubles because self-reporting was used, rather than payroll records. Trouble counts were also provided through self-reporting, rather than by obtaining the counts from LMOS. The manual CRFMP was used to forecast trouble loads and the repair force needed to handle those loads. It required a large manual effort to collect and analyze data, and had problems like those of CCUAP in providing accuracy. These major limitations are overcome in CRAS through the CRAS/LMOS/Mechanized Time Reporting interfaces described later. The goal of CRAS was to replace the "pieces" with a complete, integrated system. Thus, CRAS provides a complete cable trouble and expense data collection system that has enough detailed data to associate "hours spent" with specific type of trouble, specific part of plant, and specific work forces.

After initial functional requirements were issued in mid-1978, the New England Telephone and Telegraph Company was chosen to field test the system. Some work on CRAS architecture was done during the second half of 1978. In January 1979, software development started, with six software developers, a systems engineer, and a human performance engineer. Given strong pressure to build CRAS quickly, an ambitious development schedule was used to allow the field trial to begin July, 1979; writing and testing much of the software was scheduled for later phases of the trial. The trial was completed successfully in July, 1980, with users who were quite happy at that point. The system's economics appeared sound, and it was well documented and well packaged. The first standard version of CRAS was installed at Southwestern Bell Telephone Company in February, 1981.

This brief history shows that it was possible to build a usable system in a short time span, i.e., two years from starting to write code until a standard installation. Following a description of CRAS and its environment, later sections offer a retrospective on the development process, for it did not happen as smoothly as the speed of implementation might indicate.

III. BACKGROUND

3.1 Customer troubles and cable trouble tickets

For a Repair Service Bureau (RSB), the customer trouble report is the entity that is tracked (by LMOS) and later analyzed (by TREAT). For an OSP maintenance organization, the corresponding entity is the Cable Trouble Ticket (CTT), which is generally a more complex entity than a customer trouble. For example, suppose that a problem causes two customers' cable pairs to be crossed. The repair of this problem is considered to be one case of work, even though it involves several customers, and may even involve work at several locations. In addition, more organizations are likely to be involved in the repair of an OSP problem. Not only must an RSB handle the problem in the first place, but sometimes station repair craft persons may be dispatched, and then determine that OSP repair craft must become involved. As an extreme, but real example, if a contractor cuts a cable, tens or hundreds of customers may be out of service, and many people may work on its repair. The repair work would still be described in one CTT.

The RSB is primarily organized to handle customer troubles, which exist because they affect customer service. The OSPM organization handles this type of work when cable is involved, tracking each piece of work as a Service Affecting (SA) CTT. Unlike the RSB, the OSPM organization handles an additional type of work, termed Nonservice Affecting (NSA) or routine. The NSA work should be done sometime, even though it does not immediately affect service. Sometimes a craft person may do enough work to restore service, create a temporary closure that will need later work, then go on to the next SA problem. In other cases, people notice problems in cables or terminals that have not been reported as customer troubles, but are likely to cause problems later. In any case, the OSPM organization maintains a list of such "programmable work" that can be used to fill slack periods. As shown in Fig. 1, a complete SA CTT needs the following data:

(*i*) Data from each associated customer trouble, such as the number of trouble reports, circuit number, cable, pair, etc., all of which CRAS obtains automatically from LMOS. Before CRAS, these data were gathered manually from LMOS reports.

(*ii*) Force data associated with the CTT, such as hours worked, account charged (such as Aerial Cable or Underground), craft persons who did the work, what day(s) they worked, etc. The CRAS system obtains these data automatically from the local Mechanized Time Reporting (MTR) system. Before CRAS, these data were gathered manually via discussions with the craft person.

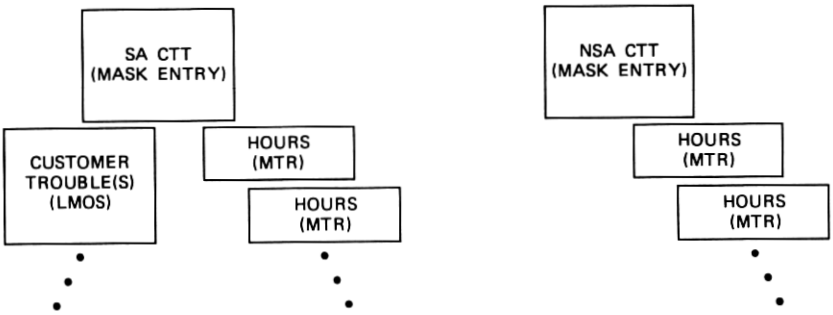

Fig. 1—Components of SA and NSA CTTs.

(*iii*) Data manually entered to complete the CTT, including trouble location, source of the trouble report, type of work, repair performed, etc. These data are obtained from the craft person.

An NSA CTT needs only items (*ii*) and (*iii*), since it has no customer troubles. The CRAS combines these pieces by a key that consists of the Cable Trouble Ticket Number (CTTN) and a Wire Center (WC) identification. Much of the CRAS design exists to assure that these different pieces of data are matched together, and that errors cause as few problems as possible.

3.2 Relationship to LCAMOS

The CRAS system is one module of three that will together comprise the Loop Cable Administration and Maintenance Operations System (LCAMOS). In the future, the LCAMOS tracker module will be used to (*i*) track individual troubles (CTTS) that are OSP-related, (*ii*) correlate multiple individual troubles into a related trouble and track this item (as a CTT), and (*iii*) transmit data on completed CTTS to CRAS for further analysis.

In addition, the predictor module of LCAMOS will be used to analyze switching machine messages and predict likely cable problems before they are reported, i.e., it will help identify NSA troubles before they turn into SA troubles.

Although CRAS is the "back end" of the LCAMOS system, it is being introduced first, for several reasons:

(*i*) It improves, integrates, and replaces several existing systems or manual plans, i.e., CCUAP and CRFMP.

(*ii*) Its financial benefits are high and quickly identifiable, because it eliminates a great deal of clerical effort. Benefits from improved management of OSP repair are also expected, although they are less easy to quantify.

(*iii*) The LCAMOS tracker will be built on LMOS front end (FE) computers, whose software has been evolving rapidly over the last few years [see Ref. 1, part VI]. The software is now flexible enough to support the effective construction of the tracker.

IV. WORK FLOW AND INPUT FOR CRAS

4.1 Workflow with LMOS and CRAS

In the LMOS environment, a trouble report is received at a central location by a Repair Service Attendant (RSA) who inputs the report to LMOS through a cathode ray tube (CRT) terminal. The trouble report becomes part of the Basic Output Report (BOR) that is transmitted to the RSB that is responsible for coordinating maintenance on the affected customer line. The BOR also contains information on past troubles on the affected line, commitment time, a number at which the customer can be reached, and the results of an automatic verification test on the line. The trouble is screened at the RSB and routed to the work center that should take responsibility for correcting the problem. The OSP repair craft are managed from a center that may be co-located with an RSB or may be a separate center that serves several RSBs. This center is currently called a Maintenance Center (MC). If the trouble should be routed to the MC, the RSB sends a BOR to the MC, using the LMOS Request Basic Output Report (RBOR) transaction, and a Cable Trouble Ticket Number (CTTN) is assigned to it. After service has been restored by the MC, the trouble must be removed from LMOS and the CTT data supplied to CRAS.

In the LMOS environment, a customer trouble report is closed out by a Final Status Transaction (FST) at the CRT terminal. In the LMOS/CRAS environment, if an OSP trouble is being closed, the person who closes the trouble is automatically reminded that CTT information is required. A Service-Affecting (SCTT) mask may be requested and displayed on the CRT to receive information on the location and cause of the trouble.

The close-out procedure described has the advantage of encouraging individuals in work groups other than OSP maintenance to enter the necessary CTT information in the CRAS/LMOS database. This is important because not all OSP repair work is actually done by OSPM forces, and yet, OSPM management needs to know where such repair work is being done to permit effective analysis.

Information on routine or NSA problems is also supplied to CRAS. The data entry procedure is similar to that for SA troubles, except an NSA (NCTT) mask is requested and displayed in place of the SCTT mask.

4.2 Interaction of CRAS and MTR

Each BOC has built a computerized time reporting system that collects all data on time charged to various work activities, then supplies that data for payroll computation and for other systems. Mechanized Time Reporting (MTR) is the generic name for these time reporting systems, which differ from company to company. Mechanized Time Reporting supplies the hours expended on various OSP activities to CRAS, to ensure accurate records with minimal input.

All craft technicians, including members of installation, business, coin and residence repair, construction, and other groups charging time to OSP repair ("R") accounts, must have a CTTN for each trouble case. All such hours are accumulated from craft persons' time reports and supplied to the MTR system.

The information derived from MTR provides data on the number of trouble cases and on the number of hours expended per trouble case.

Several LMOS/MTR/CRAS edit procedures provide users with discrepancy reports that identify most input errors. These reports enable the user to correct errors before they are propagated through the system and to prevent similar input errors from being repeated.

V. CRAS ARCHITECTURE AND DATA FLOWS

5.1 The LMOS front end and host computers

Figures 2 and 3 display the hardware architecture of CRAS and major data flows. Both MCs and RSBs use CRTs connected to an LMOS FE computer, which handles some transactions directly (such as the FST) and passes others through to the LMOS host. The CRAS SA and NSA CTT entry transactions are of the latter type. Each adds one record to the corresponding CRAS host database (SACTT and NSACTT). Host batch programs obtain customer trouble data from the LMOS Trouble History (TH) database,[2] and through several steps, attach their data to the corresponding SA CTT records. The MTR data are read and processed on the host also.

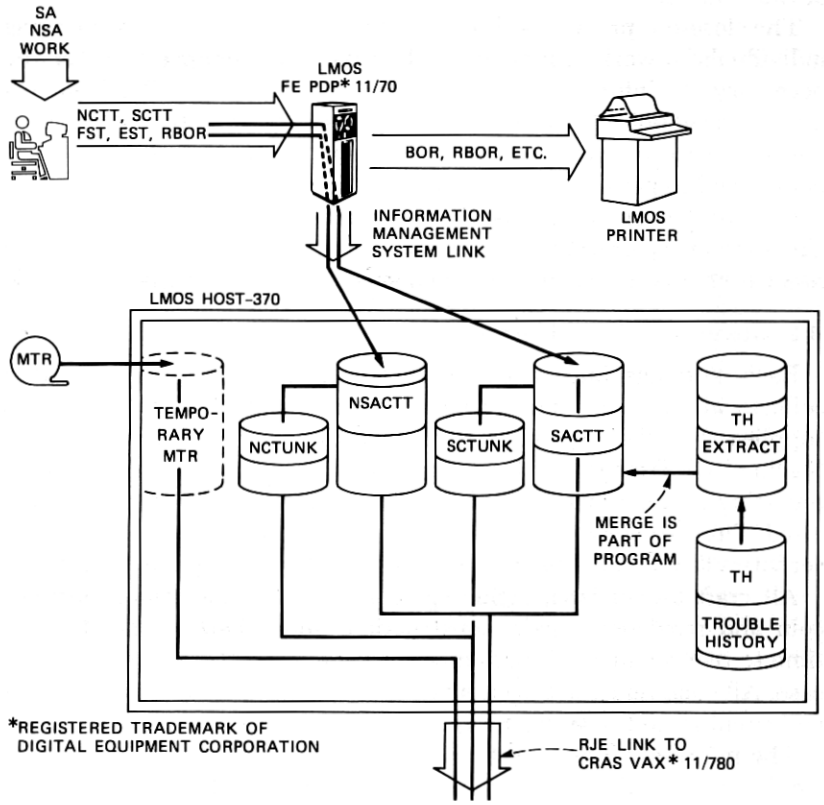

Fig. 2—The CRAS use of FE and host computers.

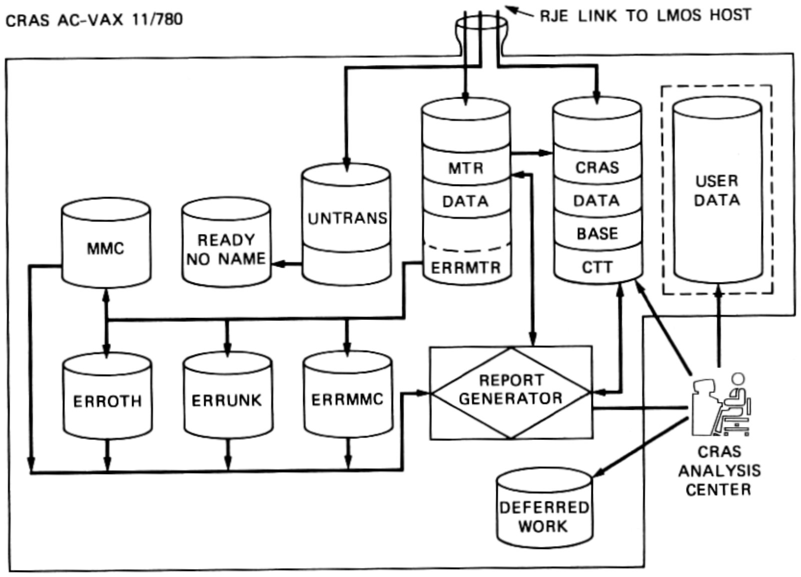

Fig. 3—The CRAS administrative computer.

5.2 The CRAS administrative computer

All the data from the previous stage is passed across a Remote Job Entry (RJE) link to another system, called the CRAS Administrative Computer (AC). See Fig. 3. The AC is a VAX-11/780 that runs under the *UNIX** operating system. Most of the CRAS capabilities are provided by the AC. These include matching of MTR data to CTT data, production of all reports, control of all CRAS host jobs, on-line documentation support, and all system administration. The AC is accessed by dial-up terminals, most frequently by MC analysts, but also by other managers and clerks.

Figure 3 shows the databases used on the AC. The CTT database has "good" CTTs, both SA and NSA, from which most trouble analysis reports are generated. Here, a "good" CTT at least possesses a legal wire center number.

The UNTRANS database includes CTTs that are "stuck" on the host, either because they contain illegal wire center numbers or because they are SA CTTs waiting to accumulate LMOS customer troubles. It also includes LMOS customer troubles not yet matched to CTTs. If many unmatched LMOS troubles are shown, CTTs are being entered incorrectly, very late, or not at all. The UNTRANS database contains images of two virtual databases (SCTUNK and NCTUNK) and a real one (TH

* Trademark of Bell Telephone Laboratories.

EXTRACT), all of whose primary copies reside on the host. The data are copied to the AC to aid the error reporting and correction process.

The MTR database contains hours data not yet matched into the CTT database. This data recirculates until it is matched correctly or ages enough that it is unlikely to ever match. The CRAS system aids matching by assuming that a CTTN is correct, but that someone specified a wire center found within the organizational boundary, but not the correct wire center. For some reason, this error turned out to be the most common one found, not the more expectable ones, such as CTTN digit transposition or omission.

The MMC database contains OSPM employees' payroll data that cannot be attached to CTTs. These include items like vacation, sick time, nonpaid absence, and any other data that are necessary to have a complete picture of OSPM employee's use of time.

Several databases (ERROTH, ERRUNK, and ERRMMC) hold MTR data known to be in error, categorized in ways to identify the sources of error. For example, CRAS maintains lists of organizational responsibility codes whose time records should be examined. If a time record appears that contains such a code, it is saved, even if it contains an unknown wire center or an improper CTTN. This assures the OSPM manager that even incorrect data entry by any relevant organization can be detected.

The CRAS system uses lists of employees, organizations, and wire center numbers to scan MTR data and save relevant records. Anything that does not match one of these lists is discarded immediately. Some spurious matches occur from data entered by parts of a BOC that do not yet have CRAS. These are kept long enough to make sure that the tables are correct, then they are discarded also.

5.3 Design commentary

The CRAS architecture is more complex than one might wish. Some of the complex matching procedures are only temporary, because they will be eliminated with the arrival of the LCAMOS tracker (Section 3.2). The AC database complexity arises from trying to make sense of data that arrives in no guaranteed order, and that may have errors that cannot be mechanically corrected. The CRAS system did not originally support the error databases. These databases seldom contain much data, so that people often do not bother using fix-up procedures to empty them. However, they must exist, if only to permit detection of consistent sources of errors, and to provide confidence that time records are being saved properly.

A host-only architecture was kept as an alternative until late in the design process. The separate AC design was chosen for several reasons:

(*i*) It permitted a faster schedule, since the *UNIX* system's tool kit and environment fit the problem well. Database requirements fit the *UNIX* system's hierarchical file system well. The low ratio of updates-to-accesses permitted simple approaches to reliability that fit into a standard *UNIX* environment. For example, in any given day, only 2–4 percent of the database files are likely to be updated.

(*ii*) It offered the flexibility of the *UNIX* system's tools, which was necessary to survive expected changes in requirements.

(*iii*) Rough economic estimates did not clearly favor either approach over the other.

VI. OBJECTIVES AND OUTPUTS OF CRAS

6.1 Objectives

Many systems exist mainly to keep an accurate, up-to-date database from which any record can rapidly be retrieved. On the other hand, CRAS is a decision support system, whose value lies not in supplying records of data for immediate use, but in extracting relevant patterns from masses of data to support effective decisions, e.g., by ranking areas of plant by maintenance costs. It does a better job when it provides the least output to isolate problems. The objective of CRAS is to have managers use this information to improve service results, ensure good performance, and reduce cost.

6.2 The CRAS reports

The CRAS system provides about 40 reports, split into the five categories described below.

6.2.1 Outside plant trouble analysis reports

The CRAS OSP trouble analysis reports are directed at the following:

(*i*) Identification of locations with high customer report rates.

(*ii*) Identification of areas with high maintenance costs.

(*iii*) Identification of all SA and NSA OSP troubles by geographic location, type of trouble, type of work, and average clearance rate.

(*iv*) Identification, by work group, of average time-to-restore service.

6.2.2 Force management analysis reports

The CRAS force management reports are designed to provide measurements of such things as:

(*i*) Individual craft performance for the OSP maintenance organization in terms of hours expended per cable trouble case, percentage of trouble found, and types of work accomplished.

(*ii*) Performance results in terms of hours per trouble report by all groups—OSP maintenance, construction, business, coin and residence repair, installation, or others charging time to OSP "R" accounts.

(*iii*) Identification of hours, including overtime, and cases where two or more people worked on the same trouble assignment.

(*iv*) Identification of hours expended and resultant performance by repair category and report source.

6.2.3 Deferred work reports

Deferred work is a listing of NSA tasks that have been identified or started but have not yet been completed. These tasks can be the result of partially completed assignments, such as temporary closures, terminal replacements, etc., or potential problems that have been noticed by BOC employees. The CRAS deferred work reports are designed to provide the following:

(*i*) A temporary closure log that summarizes temporary closure activities for a given period of time, usually one month.

(*ii*) A listing of NSA maintenance tasks that can be completed on a scheduled basis.

6.2.4 Budget reports

This category of reports assists management in budgeting and scheduling tasks by providing (*i*) a budget report that provides an analysis of the hours expended, by type of work, and (*ii*) a report that allows managers to determine work schedules by analyzing work loads in terms of days of the week and times of day.

6.2.5 Administrative reports

The CRAS administrative reports are used to monitor and correct databases. They display the following types of information:

(*i*) Database completion summary, which shows the level of database completeness, i.e., what fraction of the CTTs have acquired MTR hours and LMOS trouble records.

(*ii*) Missing geographic identifiers (low-level identifiers, which must be added to CTTs by map lookups, and sometimes must be added much later than original CTT entry time).

(*iii*) Incomplete CTTs and unmatched LMOS troubles, used for database fixup.

(*iv*) Invalid tickets, i.e., those whose data are correct according to LMOS or MTR, but that contain errors noticeable to CRAS, which has tighter error checking.

(*v*) Miscellaneous administrative databases, such as employee lists, organizational hierarchies, etc.

(*vi*) The LMOS host run summary, which shows data flows between LMOS host and the AC.

6.2.6 Miscellaneous features

The reports are all implemented as shell procedures, which use both existing tools (especially *sort*, *awk* for report computation, and *nroff* for formating) and those newly written for CRAS. A powerful report compiler exists to select and display CTTS according to complex criteria.

Many reports include statistics on the completeness of their own input data. This is a necessity in an environment where it is too expensive to get perfect data, but where people want to know just how complete the data are. Thus, CRAS is able to provide good decision support from partial data.

VII. FEATURES OF CRAS AND ITS DEVELOPMENT

7.1 Team approach with computer assistance

The CRAS development has always attempted to include an integrated team of systems engineers, human performance engineers, software developers, managers, and end users. With rare exceptions, everyone (including many who had not used the *UNIX* system before) has used the machine-enhanced communication facilities provided by the *UNIX* system. Strong efforts have been made to use the system to provide leverage for human effort, not only in generation and control of code and documentation, but in multisite communication, exploratory requirements analysis, and product distribution.

The structure of the software was strongly influenced by the nature of the personnel involved. No single person had all the knowledge necessary to do the project. The software development supervisor was both newly-promoted and new to the ARSB. Only one member of the original development group had any *UNIX* system experience; several members had neither host nor *UNIX* system experience; and a few had no software implementation experience at all. Much of CRAS was built by managers who were on sabbaticals to learn software development. Some new person always had to be brought on board. Thus, the software structure had to consist of small, easily understood, relatively independent modules. In support of the educational function, people were taught to read each other's code, to steal it when possible, and to trade it back and forth as appropriate. Thus, while there was always individual responsibility for code, there was seldom individual ownership in any restrictive sense. *UNIX* system tools were used to keep track of the activity and automate drudgery so that people could get on with the job.

7.2 Tools

When attempting to build a project quickly, under conditions of change and uncertainty, use of tools and other existing code is a necessity, if only because any prudent project manager avoids unnecessary risk. The nature of the CRAS AC permitted the use of the standard version of the *UNIX* system, so that expensive operating system changes could be avoided. Existing *UNIX* system tools were used heavily, to the point that most of the system's visible function is provided by the *UNIX* system's command language programs (shell procedures). Code sizes in the first issue of CRAS were as follows:

Lines	Type
16K	PL/I
10K	C
15K	Shell
6K	Miscellaneous
33K	Documentation

Some of these figures are misleading, since heavy use is made of program generators that operate on the source code stored above to produce the compilable source code. For example, CRAS includes a package of generators for PL/I access routines that are used to simplify the use of the host's Information Management System database system. Some product code and most development procedures were adapted from previous projects.

Tool benefits included cheapness of construction, speed of implementation, ability to demonstrate function quickly, and the chance to cope with surprises.

7.3 Data

From the beginning, CRAS development used a simple data dictionary system, which consisted of a text file of data item descriptions, plus several shell procedures for its manipulation. The dictionary was used to generate PL/I, C, and deliverable documentation. Use of a complex data dictionary system appeared unnecessary; use of some data dictionary from the beginning has proved invaluable.

The CRAS AC stores data in "packet" format, which has also been used in the new LMOS FE software. A packet is an abstract data type, which represents a collection of self-identifying data items. For CRAS, the benefits of packet use were as follows:

(*i*) It was easy to add fields, because programs that do not need the new fields need not be recompiled.

(*ii*) Storage was saved when fields varied greatly in length, or were missing entirely; CRAS has numerous such fields.

(*iii*) It was easy to write general tools to process packets.

When stored on disk, each CRAS packet is represented as a line of text ended by a new line. All but the largest CRAS packets can be manipulated directly by many *UNIX* system tools (such as *sort*, for example). This was especially helpful in the early stages of development, since occasionally used functions could rely on the existing tool kit, rather than requiring expensive development. For example, the *UNIX* text editor was often used as a database patching utility.

7.4 Documentation and planning

The CRAS system provides on-line documentation that includes an unusual degree of automation from development through distribution to end usage. It also provides an advance planning system whereby BOC users may dial a *UNIX* system, look at preliminary documentation, try out report retrieval on a test database, and use hardware sizing programs. This work is reported in detail elsewhere in this issue.[3]

VIII. DESIGN ISSUES AND PHILOSOPHY

The following discussion highlights some of the design issues faced during the implementation of CRAS.

8.1 Relationships with other systems

The CRAS system was required to interface with both the LMOS host and MTR. Previous systems have used self-reporting for both customer trouble counts and payroll hours. For the sake of accuracy, CRAS was required to obtain these data items from their true sources. In addition to the new CRAS software, CRAS implementation required a few changes to the LMOS FE, new input edits, and other software in MTR. The CRAS system also provides data to the Loop Activity Tracking Information System (LATIS). It is well known that any system interface must be examined for potential problems.

As a system designed to improve, integrate, and replace existing systems, CRAS was subject to some constraints regarding compatibility with existing systems. Because of the sensitivity of some of the CRAS data, upon which people's performance ratings may depend, people must retain confidence in the outputs of CRAS. They assure themselves of its accuracy by comparing its outputs with those of other systems. Thus, CRAS was subject to constraints such as the following:

(*i*) When counting customer troubles, it should be consistent with TREAT.

(*ii*) When counting cable troubles, it should be consistent with CCUAP and should be able to produce all existing CCUAP reports.

(*iii*) When counting hours worked, it should be consistent with MTR internal reports and with the methods of the CRFMP.

(*iv*) All its own reports should be consistent.

These constraints represented great potential for surprise, especially where the counting rules of different systems were imprecisely specified. Even worse, some of the rules were inherently inconsistent. The CRAS system was the first to try to put these particular pieces of data together, so the task of discovering the problems fell on it.

8.2 Data asynchronism

As noted earlier, a CTT contains up to three kinds of data (CTT mask, customer troubles from LMOS, and hours from MTR). Data validation would be helped if these pieces of data arrived in a consistent order. Unfortunately, every possible arrival sequence can happen in practice. The CTTs are usually entered during the day, and customer troubles (closed by the LMOS FE's FST transaction) are transmitted to the host at night, and then can be attached to the CTT. However, FST is not supposed to be used until the customer has been notified, which implies that some customer trouble records straggle in over a period of days. When there is an overload of customer troubles, CTTs are often held and entered later, so that trouble records arrive before CTTs. Depending on local policies, MTR records arrive before or after either of the other pieces.

Even if a fixed sequence were provided by normal operations, computer failures would certainly disrupt such a sequence. Thus, CRAS must tolerate all arrival sequences, and it must be immune to machine failures. It also must handle partial CTTs, as when a CTT never acquires MTR hours because of coding errors. These issues were thoroughly addressed and handled by the design.

Also addressed was the need for an unusual type of multisource error detection, which depends on examination of groups of related data to find patterns, rather than simple errors in individual items. For example, it is difficult to know whether two identical CTTNs exist because one is an extra (and can be deleted), or because one contains a typing error (and should be edited to renumber it), without looking at both together.

8.3 Change and uncertainty

During the period when CRAS was being implemented, changes were occurring and were expected to continue in the recommended organization of Bell System repair activities.[4] Some of the change and uncertainty arose from anticipated regulatory changes; other parts of it came from attempts to improve the repair process; some arose in the process of ARSB evolution.[1]

The CRAS system would often be required to deal with organizational combinations that differed from BOC to BOC and changed over time. When CRAS was started, independent MCs were only starting to come into existence, and did not exist at the field trial company.

All the above argued strongly for an emphasis on flexibility.

8.4 Implementation speed and maintainability

For various reasons, CRAS needed to be built and deployed quickly. Thus, there was pressure to get something working quickly. However, the existence of change implied that rigid, unmaintainable software would not survive.

8.5 Philosophy

Given the complex nature of the environment into which CRAS had to fit, and the need to survive continual change, CRAS design strategy assumed that perfect requirements would never be available. The resulting philosophy included the following:

(*i*) Think small, for the complex environment will stretch even small software into growth. Figures 2 and 3 offer a good example, since several of the databases came into existence only during the field trial.

(*ii*) Build a system quickly, thus lessening personnel turnover and requirements changes caused by change in the external environment.

(*iii*) Build a system that can be changed quickly.

(*iv*) Build a prototype, get it into the field, make it evolve, and make requirements converge on reality. For CRAS, it would not be feasible to discard the prototype, so it had to evolve.

IX. LESSONS

9.1 Team approach and philosophy

The approach of using an integrated team, with strong machine assistance, has worked well. In fact, the most troublesome areas have been those where we could not integrate people or functions into the environment shared by everyone else. For example, although documentation was an integral part of the system from the beginning, it was much more difficult to merge training into the machine-supported product.

The "small is beautiful" development philosophy worked well, in that it helped produce a usable product quickly. It helped CRAS survive the unusual amount of turmoil caused by major rearrangements of every organization involved in CRAS development. The idea of getting into the field quickly worked well, although the CRAS field trial probably started about three months too early. We assumed that some functions were needed to start the trial, and that others could be developed as we continued. All of the former were ready at the start, but some of the latter were actually needed earlier.

9.2 Systems engineering and human performance engineering

In a tools-based project that is moving quickly, it is important that Systems Engineering does not "throw the requirements over the wall" and then go do something else. The CRAS software developers suffered somewhat from not having enough systems engineers to try ideas with, when building software prototypes. Eventually, more systems engineering support was assigned to report to the development supervisor, who was then acting as project manager. The resulting team synergism yielded rapid solutions to problems, and a generally satisfying working relationship.

When the software developers are using good tools, and when the product structure is a tool-oriented one, the result implies that more of the staff resources must be placed in Systems Engineering and Human Performance Engineering. When it is easier to create software, a higher percentage of effort must go into knowing what it should do, not how to build it.

9.3 Other systems as data sources

If data are received from another system, and if the correctness of that data is of only marginal importance to the other system, it should be expected to contain many errors.

9.4 Complexity of environment

Organizations that have complex operations tend to evolve complexes of computer systems to help them accomplish their work. Few organizations have been able to foresee all contingencies and provide totally integrated, comprehensive systems on a timely, cost-effective basis. If something new is implemented as a stand-alone system, it is irritating to its users because it almost invariably needs data from another system, or needs to give data to another. Any new system faces an increasing number of possible interfaces, compatibility constraints, and schedule interactions.

A particular "catch-22" exists in the area of schedules. Suppose newly-planned system A needs data from already-deployed system B, and the two are controlled by different organizations. If the developers of A request changes in system B long in advance of possible deployment, they may face indifference, if only because the developers of B probably have a long list of pending change requests anyway, and cannot be sure that system A will actually ever work, or if it does work, will arrive on schedule. On the other hand, if system A is far enough along to obtain high priority from B, it may be too late to get enough real data to use for testing A, especially where the crucial need is to know the types and frequency of errors.

Another schedule difficulty arises when long lead times are required to obtain changes in other systems or procedures. Sometimes a prototype must be built quickly to discover problem areas in other systems early enough that they can be changed in time to support a later production system. A major problem faced by CRAS was the continual discovery of inconsistencies among other systems.

Complexity problems also exist in the number of organizations who own databases or systems and must be involved in planning of new systems. It is well known that multiplicity of organizations contributes to the difficulty of getting and keeping good requirements. They have needs and perspectives that contain legitimate differences, and whose relative priorities can be difficult to compare.

Such problems are hardly the property of any given organization, but are ones to be faced by any new system. It is ironic that the earlier and faster an organization computerizes, the more difficult will be the planning and implementation of later systems.

X. CONCLUSION

We have described the design and development of a successful operations system, whose history illustrates the problems to be overcome when building software in a complex and rapidly changing environment.

XI. ACKNOWLEDGMENTS

Many people contributed in the development of CRAS. We particularly thank J. A. Bayer, T. R. Schiller, J. Burtoft, E. A. Overstreet, and D. Kurshan (Systems Engineering), R. J. Glushko, G. T. Vesonder, K. A. Wright, and J. E. Zielinski (Human Performance Engineering), and M. Baade, W. F. Bauer, M. H. Bianchi, B. J. Beare, B. L. Cruse, C. M. Franklin, S. D. Rhodes. J. H. Shoemake, and R. O. Sinclair (Software Development). We are also grateful for the assistance of AT&T (especially T. Ballen, D. Webster and W. Bassham), Western Electric Company, and New England Telephone & Telegraph Company.

REFERENCES

1. S. G. Chappell et al., "Automated Repair Service Bureau: The Front-End System," B.S.T.J., this issue.
2. C. M. Franklin and J. F. Vogler, "Automated Repair Service Bureau: Database System," B.S.T.J., this issue.
3. R. J. Glushko and M. H. Bianchi, "Automated Repair Service Bureau: On-line Documentation: Mechanizing Development, Delivery, and Use," B.S.T.J., this issue.
4. L. S. Dickert and S. P. Rhodes, "Automated Repair Service Bureau: The Trouble Report Evaluation and Analysis Tool," B.S.T.J., this issue.