AT&T Technical Journal
Vol. 64, No. 6, July–August 1985
Printed in U.S.A.

The 5ESS *Switching System:*

System Development Environment

By R. G. BASINGER, J. A. HERNDON, B. KASKEY, J. A. LINDNER, and J. M. MILNER*

(Manuscript received October 13, 1983)

This article describes the elements of the system development environment that support each phase of the *5ESS*™ digital switching system product development. The elements themselves are complex hardware/software systems that are specialized to meet the particular needs of the different phases of development but integrated to smooth the transition through the phases. The requirements and design phases are supported by general-purpose *UNIX*™ systems that provide a rich set of tools for document preparation. The design is transformed into executable software for the *5ESS* system, primarily in the C programming language, augmented by C-based extension languages to support the special needs of database and diagnostic programming. This software is created, compiled, combined, and packaged for the *5ESS* switch test models by large *UNIX* systems running on IBM System/370-compatible hardware. Unit testing of developed software takes place on smaller support computer systems in an environment that closely resembles the test model. The test models, actual *5ESS* switching systems, are used for the integration and system test of the product. A dedicated computer system for each test model supports testing at the C language level. A number of actual and simulated load generators are used to stress and regression test the system. All the support systems are interconnected with high-speed data transmission networks that support remote command execution as well as file transfer.

I. INTRODUCTION

Since the early 1960s AT&T Bell Laboratories has been developing and introducing large software packages that operate, maintain, and

* Authors are employees of AT&T Bell Laboratories.

administer the many types of electronic switching systems that make up the major portion of the telephone switching network in the United States. As these packages have grown in scope and complexity, there has been a parallel growth in the use of computers and computer-based aids to support each system. For the *5ESS* digital switching system, this collection of computer aids, with its development elements, is known as the System Development Environment (SDE).* The elements are complex hardware/software systems that are specialized to meet the particular needs of the different phases of development but integrated to smooth the transition through the phases. The requirements and design phases are supported by general-purpose *UNIX* systems that provide a rich set of tools for documentation preparation. The transformation of the design into executable software for the *5ESS* switching system is done primarily in the C programming language, augmented by C-based extension languages to support the special needs of database and diagnostic programming. This software is created, compiled, combined, and packaged for the *5ESS* switch test models by large-scale *UNIX* systems. A dedicated *UNIX* system for each test model supports testing at the C language level. All elements are tied together with high-speed data-transmission networks.

The *5ESS* system architecture and the development approach provide new challenges that are met by the SDE. The software in the *5ESS* system is distributed over several different types of processors, and the architectural design is such that as new processors are developed they can be substituted into the basic system. This means that the development environment must be able to assign software to the designated processors and must make it easy for existing software to be applied as technology changes both during development and after the system is released for commercial service. Software may be located in writable memory or in read-only memory. The development environment must always associate the correct software and firmware (software in read-only memory) for each load or release of the system. The development plan provides for many organizations to be generating software simultaneously for different releases of the system. This means the development environment must allow for parallel developments with overlapping content and independent schedules for actual releases.

The environment operates under a formal methodology for each step in the process and supports the generation of complete documentation for all phases of development from requirements to system release. All source and object code is controlled and tracked for each development version of the system as well as for each officially released

* Acronyms and abbreviations used in the text are defined at the back of the *Journal.*

version. The environment allows the *5ESS* system design to evolve by sequentially adding capabilities that can be independently developed and tested. The environment also allows continued maintenance of the system by applying incremental updates to the software. Overall, the SDE gives the programmers their individual, personal contact to the system under development without the programmers being hindered by the hundreds of other programmers and developers accessing the system.

II. PROGRAMMER SUPPORT SYSTEM

The Programmer Support System consists of all components needed to facilitate the work of the software developer during the requirements, design, and software-generation phases of the development life cycle. In the *5ESS* SDE the requirements and design phases are supported on medium-scale, general-purpose computing facilities, and the software-generation phases are supported on large-scale computing facilities. The medium-scale systems are either AT&T 3B20S or VAX-11/780* machines running the *UNIX* operating system and are typically referred to as general-purpose *UNIX* systems. The large-scale software-generation machines are IBM 3033AP, IBM 3081K, or Amdahl 5860 systems, also running the *UNIX* operating system. These are referred to as Large-Processor (LP) machines.

2.1 General-purpose UNIX *systems*

The *5ESS* switch methodology relies on a series of well-defined specification documents to guide the development process. These documents include architecture, requirements, capability design, development unit design, and test-plan specifications. A standard template outline has been defined for each document, and the developer uses the general-purpose *UNIX* facilities to create and maintain these documents.

Several sophisticated tools exist in the *UNIX* environment to support this document generation. The Programmer's Workbench and *Writer's Workbench*™ systems provide tools that perform such tasks as spelling verification, double-word detection, sentence structure analysis, and readability analysis. In addition, there are tools that allow the writer to transform text into well-formatted, publication-quality documents. These tools include `nroff`, a macro language for formatting text; `tbl`, a tool to create tables; and `gc`, a tool that formats "typewriter art" into polished figures. The *5ESS* switch project's general-purpose computing resources are supplied through the computer center operations that support the general AT&T Bell Laboratories user community.

* Trademark of Digital Equipment Corporation.

2.2 *Large-processor* UNIX *systems*

The majority of the Software Generation System (SGS), source control, and Change Management System (CMS) functions are performed on the LP *UNIX* systems. The LP systems run the *UNIX* operating system and, in addition to the standard rich set of tools provided with the *UNIX* operating system, support several major piece parts integrated through well-defined procedures and scenarios. The process of software development proceeds from the initial writing of the software to its official introduction under source control. At this point the source can be compiled and introduced into a product package of varying size for execution and testing in either the Execution Environment (EE) (see Section III) or test-model target machines. It can then be changed in a controlled way throughout its initial development. After being introduced into the field it can be supported and updated. The tools to support these steps in the software life cycle will now be described.

2.2.1 *Change Management System*

The CMS provides the ability to keep track of the current version of the official program source, isolate test versions of programs, and eventually introduce either new code or corrections into the official version of *5ESS* system software. The CMS provides not just isolation of changes in differing states of development but also independence of one generic from another. It can also provide dependence of one generic on another by allowing pieces of the software to be defined as common between multiple generics. The software for the *5ESS* switch is partitioned into a number of logical subsystems, each of which performs a part of the total switching task. The software for each subsystem is organized in a structure that can be cleanly represented as a subtree of the *UNIX* file system's tree-structured directory hierarchy. These structures, known as nodes, are understood by the CMS system and the load-building tools. The subsystems are spread across several LP *UNIX* machines; however, no single subsystem spans more than one machine. Each subsystem is built (compiled) independently and then collected on a single machine designated as the Program Administration machine for final building (link editing).

2.2.2 *Software generation systems*

There are several SGSs in use to support cross-compilation to a multiplicity of target machines; System/370 compatibles, AT&T 3B20, VAX-11/780, Intel* 8086, Motorola MC68000, and several smaller microprocessors.

* Registered trademark of Intel Corporation.

The SGSs consist of preprocessors, compilers, optimizers, assemblers, linkers, and utilities. Preprocessors have been developed to provide enhanced message and data definition capabilities to the developers. They provide a higher level of abstraction and better user interface for the definition of interprocess and interprocessor communications and the definition of database relations. The output of these preprocessors is C language code.

The vast majority of *5ESS* system software is written in C, and the portable C compiler provides the basis for the multiple-target-machine environment support. A tool, `lint`, is provided for syntax, portability, and interface checking of the source code before compilation. Listing production is a separate process and can be accessed on-line. Optimizers have been provided for real-time and memory optimization of code targeted for the AT&T 3B20D Administrative Module (AM) processor and the Motorola MC68000 Switching Module (SM) processor.

2.2.3 Load building and packaging

Using CMS and SGS to create a desired version of the system source and process it into object files, the load-building process brings together the work of developers to produce an official load for testing in the test models. Using an automated Initial Modification Request (IMR) system, developers record their progress during the development of a new capability or the correction of a previously detected problem. When the developer has completed the necessary software changes and tested the changes privately, the developer submits the IMR to the load-building team for inclusion in one or more public loads (a change may belong in two or more load streams because of shared development). Using information recorded in the IMR, the load-building team extracts the desired version of the affected source without interfering with other ongoing change activity. Using a series of recipes for product construction, known as `makefiles`, and a series of node structures that represent the current view of the load, the necessary source-to-object transformations are automatically performed in a selective fashion. Only those transformations that must be repeated because of changes introduced by new IMRs are performed. This process, known as load building, takes place in parallel on all the LP machines for all subsystems. Shared information necessary for all subsystems, such as global header files and common data definitions, are preprocessed and distributed to all LP machines via the interprocessor network (see Section VII), utilizing a set of shipping tools that ensure the integrity of the distributed data. Once the load building on each machine has produced a single object file for each combination of processor type and subsystem, these high-level objects are shipped to a single LP machine and combined to form the final products, which are then delivered to the test models.

A more incremental development process that only reconstructs lower-level products is used by developers to produce private products for testing code before its official introduction. Two methods, called capability build and Quick Fix, allow the developer to build a private test product on a capability basis, rebuilding only those parts of the subsystems needed for the capability and thus introducing a small change into the test model without rebinding entire subsystems.

In addition, a set of tools has been developed that allow fixes to be quickly and efficiently introduced into generics in the field. These tools extract from updated object files the functions and data that have been changed with respect to the field, package the minimal changes with associated installation instructions, and transmit the resulting package via electronic delivery to affected sites. At the field sites the package is further processed and installed in in-service offices, without disruption of service. Associated listings are produced and available for field offices.

III. EXECUTION ENVIRONMENT

The EE for the *5ESS* system uses general-purpose *UNIX* systems to host a simulated *5ESS* system environment. Using a standard library of subroutines to stub off the routines that directly control and monitor hardware in the actual switch, the users of the EE can test the logical behavior of the vast majority of the operational software from their desks. Since the software to be tested is written in C, all that is required to build a load to be tested in the EE, rather than a test model, is to compile the code for the AT&T 3B20S rather than the AT&T 3B20D or Motorola MC68000. To support this mode of testing, the official load-building process produces loads for the EE as well as the test models. Using additional software provided with the EE, the user can test code destined for either the AM or SM or both. The testing language used is identical to that which the developer will use later in the test model. Test scripts can be developed and refined in the EE and then taken directly to the test model for regression testing on the actual hardware. By using the EE for unit and small-scale integration testing, users can isolate and correct logical problems before entering the test model. The use of general-purpose processors to support the EE reduces the complexity of parallel hardware/software development by allowing most software problems to be found and fixed before the code is executed on the switch hardware.

IV. LABORATORY TEST SYSTEM

The Laboratory Test System (LTS) for the *5ESS* system gives software developers a friendly environment in which to test their code. It does this by providing a standardized command language and by

interfacing the various test products to the *5ESS* subsystems through a single Laboratory Support Processor (LSP). The software under test is resident on one or more of the processors within the distributed architecture of the *5ESS* switch.

In general, LTS test products have an LSP-resident part and a *5ESS* switch-resident part, some of which require special hardware interfaces to the switch. The LSP part controls the switch-resident part and communicates with it over data links so that the LSP can be located remotely from the switch.

The LTS is an evolving system with multiple capabilities. It is packaged and distributed to users as a single system. Test-product improvements and new test-product capabilities are scheduled and developed based on the prioritized needs of the entire *5ESS* system development community. LTS is provided in tested and scheduled releases. Each release is accompanied by installation documentation and is installed in a *5ESS* system laboratory soak site by an installation team supported by an LTS first-application team. After a suitable soak interval the release is distributed to all system laboratories by laboratory support personnel. Procedures are available to give emergency fixes or interim releases to solve problems identified by the user community. New releases are accompanied by user documentation and by tutorials presented by the LTS developers.

4.1 Laboratory support processor

The LSP, running the *UNIX* operating system, provides the facilities to interface the test products to the users in a standard format. This standardization of format reduces the training required for users to become productive and also reduces user errors.

The LSP controls test resources and may be configured to handle dozens of user terminals that are either dial-up or directly connected. A single LSP can serve several *5ESS* switch test models simultaneously.

Software to be tested is prepared on the LP systems, transferred to the LSP via the Network Systems Corporation network (see Section 7.2), and then loaded into the target *5ESS* switch processor under control of the LSP. A different symbolic debugging tool is needed for each type of *5ESS* switch target processor (AT&T 3B20D, Intel 8086, or Motorola MC68000), but the user interfaces to these tools are essentially identical.

The LSP file system contains copies of all approved *5ESS* switch object code and symbol tables relating it to the C language source code. In addition, the file system contains object modules and corresponding symbol tables for the new code to be tested. The symbol tables are used for symbolic debugging. Any of the software object-level files can be loaded into an appropriate target processor in the

5ESS switch for testing. Frequently a test session will require software to be loaded into two or more target processors, each of which is monitored with a separate user terminal connected to the LSP. Symbolic test results or memory dumps can be transferred to the LSP for detailed analysis.

The LSP supports the use of test scripts for code debugging or regression testing. It makes available test products that allow scripts to generate switching system stimuli in the form of various types of line and trunk calls, high call-traffic volumes, or simulated craft input messages. Switching system response is then monitored via standard system outputs—e.g., traffic schedules or teletypewriter messages—or via special test software loaded into the target processor by the LSP.

Figure 1 is a diagram of the combined *5ESS* switch-LTS configuration showing that the LSP communicates via data links with various hardware subsystems within the distributed architecture of the *5ESS* switch. These subsystems are the AM, the Communications Module (CM), and multiple SMs. Three types of processors are used within the current *5ESS* switch. The AM uses an AT&T 3B20D processor, the CM uses Intel 8086 microprocessors, and SMs use Motorola MC68000 microprocessors. The numbers within each box of Fig. 1 indicate the LTS products that are available for each *5ESS* subsystem.

Four LTS products that reside entirely on the LSP are not shown on Fig. 1. They are the *UNIX* operating system, the software to interface to the Network Systems Corporation high-speed data bus, a C language-level software correction capability (Quick Fix), and an object-level software correction capability (Patch Compiler).

4.2 Test subsystems

The LTS test products fall into three broad categories. The first category is load administration facilities. The second is software debugging tools, which include C language debuggers for each of the processor types, coverage analyzers to determine the percentage of code actually executed, a Communication-Link Monitor (CLM) for recording and analyzing messages passed between *5ESS* subsystems, and a system-pause capability to permit all processors to be halted and restarted simultaneously under control of the software debugging tools. The third category is test products, which provide *5ESS* system stimuli in the form of call traffic, intersubsystem message traffic, or craft input message traffic.

The Debugger for Remote Target (DART) and the Integrated Test System (ITS) are C language debugging tools that allow the user to plant breakpoints, modify memory or processor registers, and provide formatted symbolic dumps of software variables and symbolic traces of software execution. Many of the actions of these debuggers interfere

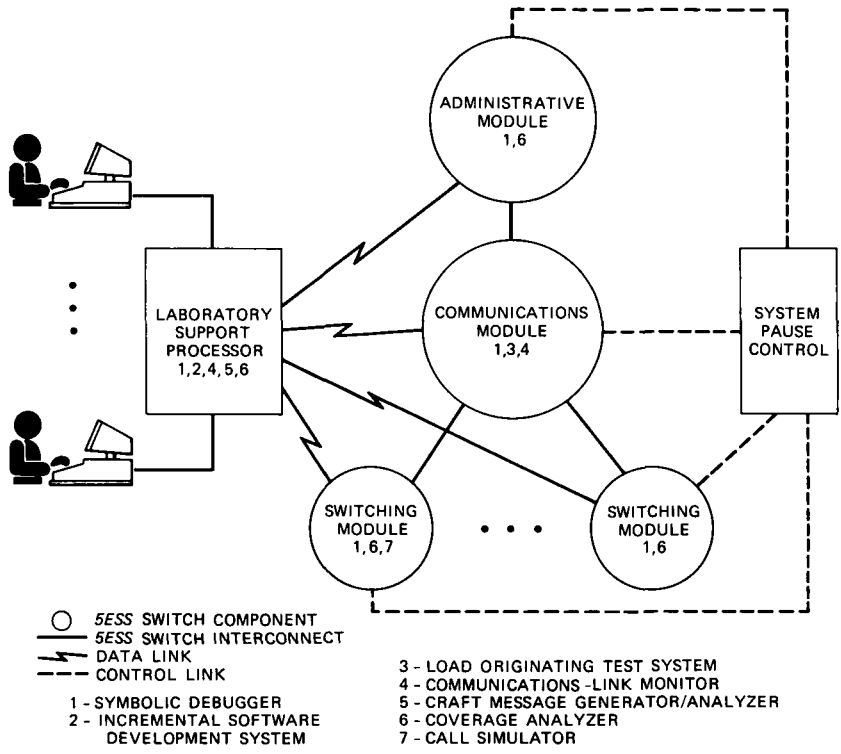

Fig. 1—LTS configuration for the *5ESS* switch.

with the normal real-time execution of *5ESS* system code and are not suitable for use in an in-service environment. ITS, however, does provide extensive debugging capabilities on a noninterfering basis.

Coverage Analyzers (CAs) are provided to determine what percentage of the code is executed. The CAs work in conjunction with compiler options that plant instructions in the object code to flag a specific memory location in a CA map when that code is executed. The LTS part of the CA function then retrieves the memory location map and produces a symbolic display of the code that has been executed. The mapping can be done either on a function-by-function basis or on a line-by-line basis.

The CLM requires a special circuit that taps into the *5ESS* switch message links at the CM and at one or more of the SMs to permit the LTS to capture selected messages or message headers for analysis. The CLM circuit also contains a display for local monitoring of the messages.

System pause is a function not presently integrated into the LSP. It consists of (1) circuits that interface with each processor in the

system and (2) an array of switches and lamps. The switches are used to select which processors will be enabled for a specific test, and the lamps display whether the associated processor is halted. A development currently under way will replace the manual switches with ones that can be remotely controlled by the LSP.

Call traffic is generated by programmable call simulators that contain (1) circuits that simulate lines and trunks and (2) circuits that can generate and detect tones and various types of telephone signaling. These line and trunk circuits interface to the *5ESS* system SMs as standard lines and trunks, and calls generated by the call simulator appear to the *5ESS* switch to be normal calls. Test scripts are written in a C-like simulation language and stored in test sequence files on the LSP. A wide variety of line and trunk calls are possible. These test scripts are extensively used for regression testing of the *5ESS* system generics.

The Load Originating Test System (LOTS) generates high-volume message traffic, which appears to the AM as coming from a number of SMs in the *5ESS* switch. In reality, the messages are generated by special software loaded into Module Message Processors (MMPs) in the CM subsystem of the *5ESS* system. The messages generated by LOTS simulate line-to-line, line-to-trunk, and trunk-to-line calls that represent either inter-SM or intra-SM traffic. The messages may represent completed calls, abandoned calls, Custom-Calling calls, or a variety of other call-sequence situations. In addition, the LOTS software generates corresponding traffic data and AMA billing data, which it sends to the AM for processing. Each LOTS MMP can generate messages representing approximately 70,000 busy-hour calls. With four MMPs, LOTS is able to load the AM with message traffic representing close to 300,000 busy-hour calls.

The Craft Message Generator/Analyzer (CMGA) is a test product that interfaces to serial I/O channels of the AM, such as the maintenance channel or service-order channel. The CMGA generates teletypewriter traffic on these serial channels and records the system responses for analysis. Test scripts can be generated and stored on the LSP, and the *5ESS* responses can be matched against the expected results.

V. TEST MODELS

5.1 General description

The fundamental purpose for the test models in the *5ESS* system development environment is to allow programmers to debug their code on the target machine. Subsequent software integration, load stress testing, stability runs, and final generic verification are also accomplished on a configuration that is representative of expected capability interactions in the field. Additionally, the test models integrate the

hardware design into the software architecture to satisfy the overall system requirements. In this manner, the test models simulate the latest state of the hardware to allow the software to execute and operate under the expected conditions that will be seen in the field.

The test model simulates an operating local telephone office with as many of the myriad combination of features and capabilities as are required to test the generic being produced. For feature debugging purposes, there is a subscriber test station, which allows manual testing of line features (e.g., dial pulse, Touch-Tone, coin) via the appropriate station set access. The subscriber test station also allows manual access to a variety of trunks for testing of transmission and signaling features of the *5ESS* switch. Automated load test equipment is also provided for simulated traffic effects to test the real-time capacity and system response of the software architecture.

In addition to the target machine and the LTS, the *5ESS* system test models are also equipped with a user area to create a friendly environment. When this space was planned, considerations ranged from furniture to a complete simulation of the field Master Control Center (MCC). Cabinets, tables, computer terminals, phones, and other appointments were considered in establishing an environment conducive to the development effort. Access to the LTS as well as the LP and general-purpose *UNIX* systems is provided via direct lines and dial-up service. Figure 2 displays a *5ESS* system test-model user area, showing the LTS user terminals and the MCC in the foreground with the *5ESS* switch itself behind the glass partition.

5.2 Lab engineering

The detailed configuration of a given test model is generated from requirements provided by the development organization in a fashion analogous to the dial administrator and equipment engineering functions in an operating company. This hardware and the associated engineerable assignments are represented in the executable software in the office-dependent data. Changes to this software abstraction of the peripheral hardware must be coordinated with changes to software tools altered by the target code, changes to the hardware configuration, or changes in assignments within a test model. This interaction of the office-dependent data with the state-of-the-generic development is one major aspect of the test-model work to ensure the integrity of the hardware as viewed by the target software.

5.3 Load procedures

The developing software takes on the character of a generic through addition and update of the various pieces into an integrated whole called a load. The test models provide the environment for maintaining

Fig. 2—*5ESS* system test-model user area.

a load of a particular vintage, termed the public load, and bringing in new loads on a schedule commensurate with developer needs. Each public load may encompass software and, possibly, hardware and firmware changes in the evolution toward a field-grade generic. Additionally, the test models allow altered states of these public loads, termed private loads, to be tested by individual programmers for specific situations.

5.4 Developer testing

The individual developer testing with private loads is the essence of the proof that the capability under development meets the intent of the requirements. The test-model environment provides the necessary hardware and software tools to allow the programmer to test, debug, alter, and retest the code under examination to the satisfaction of the documented test plan. The foundation of these private loads is the previously mentioned public load, against which the private loads are built. The programmer private-load mechanism begins in the LP environment with the public-load structure and manifests itself in the private software under test. As each capability is proven to meet the requirements in this environment, all the individual capabilities are reconstituted together to form the next public load. This synthesis of numerous capabilities into the new thesis of a public load is then regression tested to produce information relating to interactions among the various capabilities added to the developing generic.

5.5 Load testing

A subset of the test models is configured for regression and load testing of the public load. This mechanism of stress testing the software via simulated high-level telephone traffic has consistently generated heuristic information that uncovers latent problem interactions in the generic load. System testers and private-load developers take advantage of programmable call-simulation equipment to provide mixtures of ordinary and feature-activating telephone calls up to and beyond the design capacity of the system to tune system behavior at all traffic levels including overload.

5.6 Lab maintenance

As a final note, the public load is used as the vehicle to maintain the test-model environment at a satisfactory level. Such maintenance time is used to fix problems uncovered throughout a given period, apply official change notices issued against the hardware, evolve and grow the hardware according to agreed-upon development plans, and reaffirm the stability of the test model with the appropriate features in the public load. As always, this maintenance activity is balanced

against the need for the test model as a system development resource and trade-offs are constantly being made in the fast-paced development environment.

VI. SPECIAL-PURPOSE SYSTEMS

In a system the size and scope of the *5ESS* switching system, a number of functions require special facilities or benefit from novel uses of computer systems. The applications described in this section are examples of special-purpose systems in use by *5ESS* system development.

6.1 SCANS

The Software Change Administration and Notification System (SCANS) provides a gateway between the *5ESS* SDE and the AT&T Technologies field-support network. Updates to software running on in-service *5ESS* switches are produced as explained above and then sent to the SCANS machine for packaging and transmission to the field. The SCANS machine, interconnected to the other *5ESS* system development systems via an interprocessor network outlined below, focuses the necessary hardware and software for this task in a single machine but gives the whole project access to the field-support function.

6.2 Project management

The coordination and control of a large development project such as the *5ESS* switch requires considerable computer support. Dedicated systems with associated tools are used to track and report on the progress of development; maintain lists of action items and resolutions; and maintain and distribute paper and electronic copies of architecture, methodology, and schedule documents. An ever-increasing variety of graphics applications have been developed to capture and represent the plans and status of the project.

6.3 System-testing support

To ensure a quality product, extensive system-testing activity is performed on the *5ESS* switch. A dedicated computer system is used to store and track the status of the large battery of regression tests that have been developed for the switch. Electronic logging and automated analysis of output messages from *5ESS* switches also are routinely provided.

VII. NETWORKS

The *5ESS* system development involves hundreds of software developers at several AT&T Bell Laboratories locations. Support of this

number of developers requires several dozen computer systems that must be connected to both the developers and each other.

7.1 Terminal networks

To interconnect each developer with the computer systems that he or she may require in the course of a day's work is itself a major task. To meet this need the *5ESS* system development presently is using three approaches: *Dimension*® Custom 2000 PBXs, Develcom, and the *Datakit*® virtual circuit switch.

7.1.1 Dimension *Custom 2000 PBXs*

The primary site for *5ESS* system development is a complex of large buildings in a campuslike setting. Each developer is provided with dial-up access to all computers in the complex via a network of *Dimension* Custom 2000 PBXs dedicated to data switching. The most common connection is currently 1200 baud, although faster and slower rates are possible. This network is part of a larger companywide voice and data switching network that provides flexible access to computer systems at all company locations.

7.1.2 Develcom

To provide switched but nonblocking access at higher speeds to replace point-to-point connections previously used, a small data switch has been installed. These lines are used primarily by hardware developers to interconnect proprietary microprocessor development systems and programmable read-only-memory programmers to other computer systems.

7.1.3 Datakit *virtual circuit switch*

To provide widely available very-high-speed access from all potential users to the computer systems, a *Datakit* virtual circuit switch local area network is being introduced. This network will eventually supplant the *Dimension* PBXs as the primary access to all local computer systems and provide gateways to *Datakit* virtual circuit switch networks at other company locations. The bandwidth available with this approach is matched to the needs of intelligent terminals and work stations now finding their way into the project.

7.2 Interprocessor network

File transfer between the development-support computers is provided by the HYPERchannel* network and associated adaptors. The network consists of two dual-coaxial-cable backbones, one in each of the

* Trademark of Network Systems Corporation.

primary development buildings, interconnected via dual fiber-optic links. Further extension and complete interconnection of the networks is provided by a gateway system, which transships files between subnetworks. Connections are supported to AT&T 3B20S, PDP-11/70,* VAX-11/780,* and System/370-compatible machines. Data rates on the 50-Mb/s buses range from about 3000 to over 100,000 bytes/s, depending on the combination of source and destination machine types and loads. Networking software in each connected processor controls access to the buses and provides for transmission of files, execution of commands on remote systems, and notification of file delivery or command results. The file transfer mechanism is used to propagate changes in the *5ESS* system software being developed across support systems. It is also used to deliver code from the LP systems to the EE or LTS systems for testing. Updates to the *UNIX* operating system and *5ESS* system tools are distributed via the network, and performance data on the support computers and the network itself are collected via the network. Remote execution of commands allows a user on one system to perform activities on another system without logging into the second system.

7.3 I/O network

The *5ESS* switch project uses a network to access high-speed-laser page printers for publication-quality documentation, plotters for graphical data, and specialized resources such as a Cray supercomputer. This network also provides an alternate path between *5ESS* system support machines via the remote-job-entry network. Electronic mail may be routed via this network to any computer system within the company. A form of limited remote command execution also exists for this network.

VIII. CONCLUSIONS

A system development environment that allows generation of high-quality software, ensures high productivity of programmers, and provides administration of all versions of the system has been developed. The environment has been able to evolve as required by the *5ESS* system architecture and development plan. This has been achieved by using a distributed architecture much like the *5ESS* switch itself, where growth can be accomplished by adding computing and testing elements. Contributing to the ability to gracefully evolve is the versatility of the *UNIX* operating system and the C language, which is the base for the environment as well as the *5ESS* switch. The importance of a formal

* Trademark of Digital Equipment Corporation.

methodology cannot be overlooked. The *5ESS* system project can be expected to continue to rapidly add features and capabilities for many years using the SDE.

AUTHORS

Ronald G. Basinger, B.S.M.E., 1966, Ohio Northern University; M.S.M.E., 1967, Purdue University; AT&T Bell Laboratories, 1966—. Mr. Basinger was initially involved in the physical design of electronic equipment for various switching systems. In 1976 he was promoted to Supervisor and was responsible for the design of equipment for the Traffic Service Position System (TSPS). He then supervised groups responsible for TSPS project coordination, system testing, and operational software development. In 1981 he became Head of the *5ESS* System Laboratories Department. In 1983 Mr. Basinger assumed his present position as Head of the International Systems Planning and Data Design Department, where he is responsible for development planning for international applications of the *5ESS* switch. Member, Tau Beta Pi, Pi Tau Sigma.

John A. Herndon, A.B. (Liberal Arts), 1952, University of Chicago; M.S. and Ph.D. (Physics), University of Tennessee, 1954 and 1957, respectively; AT&T Bell Laboratories, 1958—. Mr. Herndon first worked on exploratory development of data communications systems, including design of tape transports, and characterization of new semiconductor devices. From March 1960 to July 1964 he supervised the physical design and factory testing of 101 *ESS.* He was Head of the 101 *ESS* and No. 2 *ESS* Physical Design Department from July 1964 to April 1968. From April 1958 to February 1983 he headed various departments responsible for software development on 1/1A *ESS,* 2/2B *ESS,* 3*ESS,* and *5ESS.* In February 1983 he became Head of the *5ESS* Laboratory Test System Department. In 1977 he helped to organize the Software Technology Committee of the IEEE Communications Society (COMSOC). He served as Secretary of that committee from 1978 to 1981, Vice Chairman from 1981 to 1982, and Chairman from 1982 to 1983. Mr. Herndon organized and chaired two technical sessions at COMSOC sponsored conferences and has co-authored two papers presented at COMSOC conferences. Senior member, IEEE. Member, Software Technology Committee.

Baylen Kaskey, B.S.M.E., 1950, University of Pennsylvania; Kansas State University, 1950–1951; Bell Laboratories 1951–1962; Bellcomm, Inc. 1962–1967; AT&T Bell Laboratories, 1967—. At AT&T Bell Laboratories Mr. Kaskey's initial work was on guidance systems for antiaircraft and antiballistic missiles. In 1962 he joined Bellcomm, Inc., an AT&T company formed to help NASA manage the Man-in-Space program. There he headed a department responsible for developing the flight mission plan for the Apollo program. In 1967 he returned to Bell Laboratories and was involved in development of operations support systems for switching. He was appointed Director of the Special Switching Systems Development Laboratory in 1980 with responsibility for No. 1A *ESS* software development systems and international application developments for switching systems. In 1982 he became Director of the Software Development Systems Laboratory with responsibility for developing an integrated *5ESS* switching system software development environment. In 1984 Mr. Kaskey assumed his present position as Director of the Operator

Services and Special Applications Laboratory with responsibilities for special switching developments for government use and for developing an operator services system for the *5ESS* switching system. Member, American Management Association, ASME, AIAA, Tau Beta Pi, Sigma Tau.

Judith A. Lindner, B.S. (Mathematics), 1968, Northern Illinois University; M.S. (Computer Science), 1976, Northwestern University; AT&T Bell Laboratories, 1968—. Ms. Lindner's experience at AT&T Bell Laboratories includes work in the fields of compilers, computer aids for design, operating system, switching software development, and software development systems. Her major projects include the *4ESS* and *5ESS* switching systems, and early operating system development leading to the Duplex Multienvironment Real Time operating system. She is currently Head of the *5ESS* Switch Software Development System Department.

J. Michael Milner, B.S. (Electrical Engineering), 1972, The Massachusetts Institute of Technology; M.S. (Computer Science), 1975, Ph.D. (Computer Science), 1976, University of Illinois; AT&T Bell Laboratories, 1976—. At AT&T Bell Laboratories Mr. Milner first worked on the initial software architecture of the *5ESS* switching system. Since 1979 he has been involved in the design and implementation of the software development environment for the *5ESS* switching system, with emphasis on software generation systems for microprocessors. Since his promotion to Supervisor in 1981, he has managed *5ESS* development environment planning and *5ESS* development computing capacity and performance provisioning. He presently supervises the Advanced Development Environment Studies Group. His interests are distributed computing for large-scale software development, and architecture of software development environments. Member, Association for Computing Machinery.