Current programming tools and techniques facilitating program development and maintenance under Operating System/360 and /370 are collected and discussed.

These aids are categorized and defined according to their function. Abstracts of some of the available programs are also presented.

A guide to programming tools and techniques

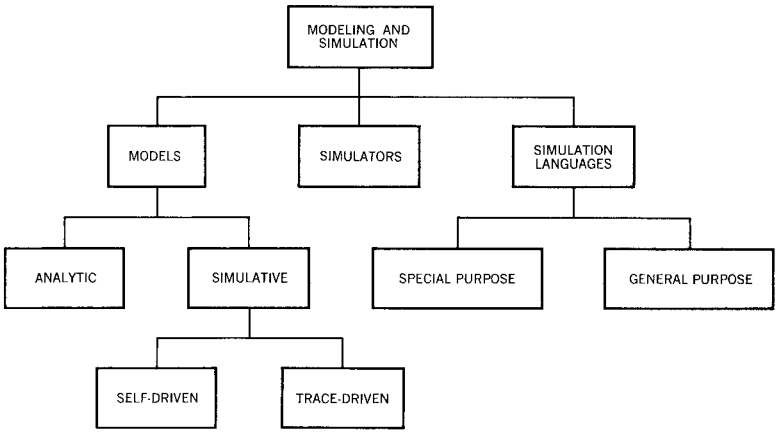
by J. W. Pomeroy

Programming has become a major concern within corporate organizations because programs and programming systems now assist in the day-to-day conduct and management of most corporate activities. Common to all programming departments or groups is the rising cost of program development and maintenance. As a result of this increasing concern, programs and techniques have been formulated to facilitate development of other programs. These programming tools may automate one or more tasks in the programming process, establish libraries of proved programs in the form of macroinstructions and subroutines, or provide assistance in a programming support area such as program management and documentation.

The purpose of this guide is to provide information to assist programmers and their management in selecting the most appropriate tools for their particular needs in the Operating System (OS) environment. This guide is divided into three parts. The objective of the first section is to acquaint the programmer and programming management with the terminology and functions of programming tools and techniques. The categories of tools and techniques discussed are the following:

- Modeling and simulation
- Measurement and evaluation
- Function testing
- Implementation
- Programming support

Figure 1 Modeling and simulation tools



These categories are further subdivided into functions. The second section is an index which organizes the program tools that follow into their appropriate categories and functions. The third section contains selected abstracts of available program tools in alphabetical order. Any references mentioned in the abstracts can be obtained from a local IBM branch office.

The concepts discussed in the first section are those with which the author is most familiar and has found to be useful to programmers. The programs presented in the following sections are only those which have been distributed by IBM. Other programming tools and techniques have been discussed elsewhere in the literature.

Modeling and simulation tools

Modeling and simulation tools, illustrated in Figure 1, support the programming development process in many ways. During the planning phase, models permit the study of new hardware configurations, work load variations, software alternatives, and operating procedures. New system designs are examined and evaluated before they are implemented. Therefore, performance problems can be addressed and resolved at an early stage in the development process. Models also provide a consistent basis for evaluating design alternatives, for estimating system processing capacity, and for planning to meet predictable growth in functional capability as systems evolve. In addition, a modeling tool tends to insure completeness of design by forcing designers to be specific and to consider details that might otherwise be overlooked. Simulators are used during testing activities when the total hardware system is not available. In measurement and

evaluation activities simulators are also used together with monitors, traces, and timers.

model

A *model* is a logical representation of a system such as a flowchart, blueprint, or mathematical formula. Models used to represent computer systems are classified as analytic or simulative.

Analytic models provide a set of mathematical equations for calculating time averaged over some mix of transactions. Because equations describing complex systems tend to become complicated and often impossible to formulate, it is usually necessary to make simplifying assumptions. If these assumptions provide reasonably accurate answers, then the analytic approach becomes preferable to a simulation model because of the faster execution time. FORTRAN, PL/1, and APL are programming languages most commonly used for constructing analytic models.

Simulative models are logical software representations of the discrete entities and events of some target system. Structured usually in two parts, one part of the model represents the hardware and control program services where the hardware may or may not be the host system. For example, a System/370 configuration can be modeled using System/360 hardware. The second part represents the application programs and processing work loads. Many models have the ability to simulate the effects of running on different configurations or CPU models and to make operating system changes without the necessity of a systems generation. *Trace-driven simulative models* derive their input from trace data that has been collected during actual execution of the target or a similar system.[4] Trace data is not possible, however, for new software that is being modeled. This type of model where the programmer or modeler must provide the inputs is called a *self-driven model.* Because of the modeler's direct involvement in input creation, a self-driven model is considerably more difficult and time-consuming to construct than a trace-driven model.

simulation languages

Most simulative models use special languages designed specifically for that purpose. Simulation languages are classified as general-purpose or special-purpose languages. A general-purpose language such as General Purpose Simulation System V (GPSS V), can be used to model any system. Special simulation languages, such as Computer System Simulator II (CSS II), have built-in features specifically aimed at computer systems performance studies. A special simulation language describes the system hardware components and the way in which they interact in terms familar to systems engineers and programmers. Much of the descriptive detail that a generalized language requires of the modeler can be provided automatically by special simulation languages.

A *simulator* is defined as a tool whose function is to provide to the target system inputs that resemble those that would have been provided by the components being simulated. Simulators, usually associated with software monitors, allow measurement and analysis when all parts of the system, such as terminals, are not present.

simulator

Measurement and evaluation tools

Program measurement and evaluation are most often used as adjuncts to system, product, and acceptance testing. In addition to establishing that performance specifications have been met, these tools are used to calibrate earlier models, provide useful information to designers of similar systems, and to tune existing systems. Tuning activities should consider the total operating environment in which the program runs and should avoid suboptimization—optimizing a part to the detriment of the whole.

System characteristics are not always quantifiable. Performance is a quantitative statement of how well a computer program does its job. Performance characteristics such as throughput, resource utilization, reliability, job turnaround and response time, availability, capacity, and effectiveness are measurable. However, usability characteristics such as human factors, serviceability, and maintainability, which express the effect a program has on its environment and the people who use it, are not measurable. They assist in establishing the ease of use of a program relative to other programs.

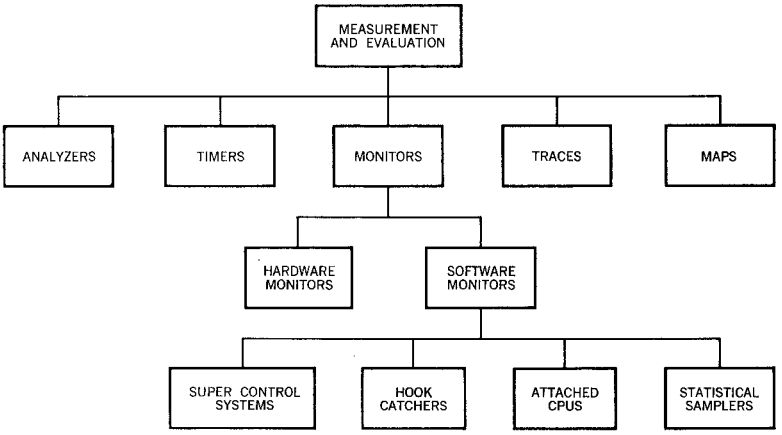
Performance characteristics are measured by monitors, analyzers, timers, maps, and traces. Also, the functions of monitors, maps, and traces can be combined with the modeling function. The organization of these measurement and evaluation tools is depicted in Figure 2.

Hardware monitors are devices that count and time the intervals between selected voltage pulses in a running computer without degrading the system. Accomplished by attaching probes to the target system, these measurements can be used to assess hardware usage such as wait and channel times. System activity is then recorded in the monitor's registers or on punched cards or magnetic tape. Most hardware monitors have reporting and/or data reduction programs supplied with them. Some of these programs provide plots as an option.

monitors

Whereas hardware monitors are used to measure the mechanical processing activities of a computer system, software monitors indicate I/O activity or the processing activity within the Central Processing Unit (CPU). A *software monitor* is a program that

Figure 2 Measurement and evaluation tools



provides detailed statistics in a production processing environment. A software monitor records usage of system software and hardware components by counting frequency of use and by recording the amount of use over some sampling period. Statistical output resulting from these measurements can be reported immediately or can be saved as a chronological trace for postprocessing. Software monitors are generally programmed in machine language to minimize degradation of normal system processing. The timing source used can be a standard computer interval timer or some other suitable high-resolution timer. Desirable features of a software monitor are the ability to add and remove the monitor without disrupting normal system processing and the ability to selectively request statistical options.[5]

The following are the four basic approaches to software monitoring:

- *Super control system* monitoring treats the operating system being measured as a problem program. The monitor operates in the supervisor state and receives control when interrupts occur. It records detailed information about the target system, but, in doing so, such a monitor may cause degradation in target system performance. Special hardware such as a high-resolution timer may be required for super control system monitoring.
- *Hook catching* monitoring relies on instructions placed at strategic points in a system being measured. These instructions cause a transfer to a catching routine that records appropriate data for later processing and then returns control to the measured program.
- *Attached-CPU* monitoring uses standard IBM hardware, including a secondary CPU, to feed data to the system being

measured and to analyze the response time of that system. A primary use of an attached CPU monitor is to analyze teleprocessing systems.

- *Statistical sampling* records events occurring at specific points in time. This technique has less impact on normal system processing than the techniques previously discussed, yet produces equally valid measurements when optimum sampling intervals are selected correctly.

analyzers

An *analyzer* provides source-language or execution-frequency statistics to assist in performance evaluation. It normally runs in problem-program state and requires no modification to the target program. Source-language analyzers are used to optimize individual programs.

timers

A *timer* time-stamps and/or computes elapsed time between target program events. It executes in supervisor or problem-program state and may require target program modification. A timer normally depends on the interval timer for timing information and may have the capability of refining the interval timer beyond its normal tolerance. Time span may be from one instruction to an entire job stream. Many tools combine the timing function with monitoring and tracing functions.

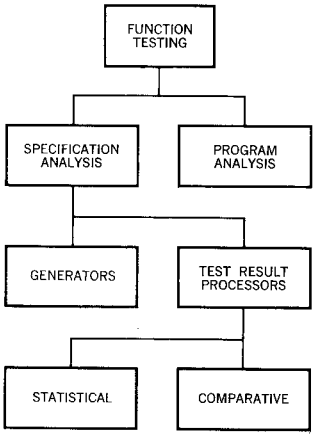
maps and traces

Maps provide location and/or size information about all or selected parts of the target system or about device-resident data. A *trace* records the chronological sequence of events taken by a target program during its execution. All or selected segments of the program may be traced. Output from a trace may then be used to drive models of the system to be measured.

Function testing tools

Function testing assumes three forms at different stages in program development. *Module testing* (debugging) is a form of function testing that is done by the programmer to ensure that basic algorithms and routines are coded correctly. *Interface testing* (component and system testing) is performed when a number of interdependent modules are linked together by higher level routines and algorithms. The purpose of interface testing is to verify that the basic modules have been combined correctly and that the composite operates according to its specifications. A third type of function testing, *regression testing*, is performed to ensure that a previously tested program correctly executes its original functions after modification or enhancement. Function testing tools are presented in Figure 3.

Figure 3 Function testing tools



The approach to function testing often used by programming groups is to extract basic functions from the specifications and

write test cases for as many of the valid combinations of functions as can be identified through a manual search. In this approach, *specification analysis*, the program is run using these test cases, and the actual results are compared with expected results. *Generators* and *test-result processors* are two kinds of tools currently available to assist programmers with this approach to function testing.[6] *Program analysis tools* describe an alternate approach to specification analysis.

generators

A generator produces test data, test cases, or job streams to exercise the target program. Other names commonly given to this type of tool are "exerciser" and "driver." A consideration of generated test cases is that they may be redundant and may not cover all possible functional variations. In this case, their use to exercise code or to verify functions is not recommended. However, when generators are used to drive the target system for the purpose of gathering information about system capacity or throughput, they save much time and effort.

test-result processors

Test-result processors perform test output data reduction, formatting and printing. Some perform statistical analysis where the original data may be the output of hardware or software monitors. Others compare expected to actual test results.

Although straightforward in concept, comparing actual with expected or previous results and then reporting on mismatches presents problems which in practice are difficult to overcome. Non-significant output data, such as date and run time, and the number of unanticipated variations possible in most software systems can cause the number of valid, but mismatching, results to be high. This obscures the significant mismatching results that the technique attempts to uncover.

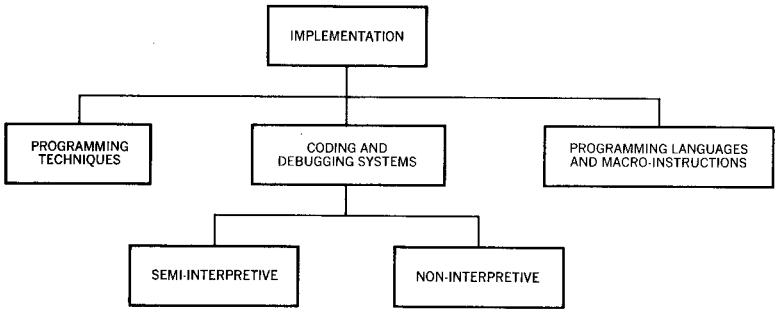
program analysis

Whereas the specification analysis approach to function testing involves a search for functions described in the specifications, program analysis involves a search for functions intentionally or unintentionally implemented in the program. This approach is effective when used during module testing by the programmer who developed the module to be tested. The programmer rigorously examines all paths that a program can take to uncover logic errors such as unexecuted code and execution-sequence-dependent bugs.

Implementation tools

Implementation tools, illustrated in Figure 4, assist programmers in coding, debugging, and module testing. Included in this category are programming languages and operating system functions that support program implementation as well as the pro-

Figure 4 Implementation tools



gramming techniques, specialized compilers, macroinstruction libraries, and coding and debugging systems. Regarding languages, our objective is not to propose a "best" language, but to suggest languages that are typically used for particular needs or applications.

programming technique

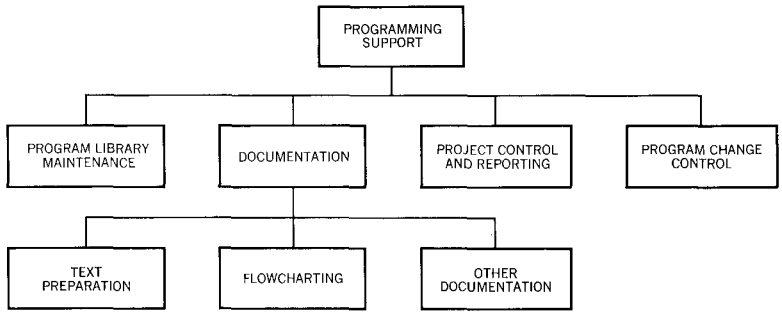
A *programming technique* may be a conceptual approach to the entire programming process, or it may also be a description of a methodology for performing specific programming functions. The chief programmer concept affects all programming activities and structured coding is a method for a specific programming activity.[7]

programming languages and macroinstructions

The coder has a rich variety of programming languages and macroinstructions from which to choose. Typical of the *general programming languages* are COBOL, FORTRAN, and PL/1. *Specialized languages* are used for particular applications such as the simulation languages previously mentioned for use as modeling and simulation tools. An experienced programmer applies the language most appropriate to the application or coding situation at hand. For example, he may wish to create a prototype of his system using APL. He may then support his design with a model written in CSS, and write his application programs in PL/1. Based on performance predictions or actual system measurements, he may wish to code certain modules in assembler language to achieve maximum execution speed.

Macroinstruction languages provide *macro libraries* that either provide tested code for frequently encountered operations or have a special objective such as debugging. Block structured programming macros, used by chief programmer teams to facilitate block structured coding, provide stylized branching within a set of standard programming figures. The need for explicit branches to different parts of a program is thus eliminated. The results are programs that can be read sequentially and that can be easily understood by programmers not involved in the original coding effort.[7]

Figure 5 Programming support tools



coding and debug systems

Interactive *coding and debug systems* are aimed at removing the programmer from the batch processing environment where a large part of his time is spent awaiting the results of the latest compile and execute run. These systems attempt to provide advantages inherent in hands-on testing without the inefficiencies in machine utilization.

There are two basic approaches to systems that provide interactive testing and debugging facilities. By one technique, instructions are executed *semi-interpretively*. Here, the environment is a virtual system in which the user loads, executes, and directs the progress of the execution of the source code. An example of this approach is the PL/1 Checkout Compiler. TSO TEST and CP/CMS use a *noninterpretive* approach, wherein problem programs are actually executed. The user can suspend execution at specified locations, display and modify storage and registers, and continue execution. The noninterpretive approach provides interaction at the object-code level and thus is less convenient to use by programmers who are accustomed to source-code notation. Noninterpretive systems, however, do provide a more realistic environment for the problem program than do semi-interpretive systems. In the semi-interpretive case, a problem program error does not cause its execution to terminate as it may be in a noninterpretive system. Also, with instruction interpretation, problem program execution is somewhat slower.

Programming support tools

Programming support tools are used to contain, control, and access the data base required during the life span of a programming project. These tools are depicted in Figure 5. The range of data classes that can be supported includes text, flowcharts, specifications, source and object code, and load modules. In addition, cost, schedule, and performance information for project control may be included. These tools may support a broad range of programming activities from architecture through

maintenance with an integrated system of support functions. However, many of the tools available provide a specific function.

Compared to tools in the previously discussed categories, a greater measure of permanence must be factored into decisions regarding the selection of support tools because a support data base is developed that tends to commit the user to a particular programming support system.

documentation

Program documentation spans a wide range of material including specifications, project workbooks, flowcharts, source listings, record layouts, set-up sheets, logic manuals, user guides, and trouble reports and replies. For the descriptive purposes of this paper, documentation tools are further subcategorized into text and flowcharting preparation.

Text preparation. Numbers and kinds of functions provided by text preparation systems vary as do their costs per document. In the selection of a text preparation tool, the document's intended audience must be considered as well as the ease of use, functions supported, and quality of the finished product. The ease of use attribute includes the following functional supports:

- Data entry
- Editing—erase, insert, change, and move words or groups of words
- Processing—spelling check, hyphenation, and index generation
- Formatting—page numbering, double and single column, page and column leveling, paragraph control, and right justification
- Document maintenance, storage, and retrieval
- Output device flexibility

Flowcharting. There are two techniques for flowchart generation. One approach produces flowcharts by analyzing the program's source code. Flowcharts produced by this method require no additional programmer efforts, but may be very detailed and low-level. A second approach uses a flowcharting language, or commands. Although programmer effort is required, the level of detail and general appearance of the flowchart can be controlled.

program library maintenance

In addition to source, object, and load module library maintenance, *program library maintenance* tools can be designed to support related functions such as program version and level control, test case library maintenance, and project reporting and control. The closer a library system can come to a centralized repository for programming data, the more opportunities exist

for cross-checking between classes of data such as plans and project reports, flowcharts and implementation code, and specifications and test plans.

A library maintenance system supporting many of these functions can be built using standard OS utilities and job control language. One such system is the Programming Production Library (PPL) used by a chief programmer team. The PPL consists of programs and procedures that aid program development where a highly visible external library is important so that individual team members can read each other's code for documentation and interface information.[7]

project control and reporting

The objective of *project control and reporting* is to gain visibility and control over the elements of cost, schedule, and performance.[8] Frequent reports do not guarantee control or visibility, and they may also cost many valuable man-hours in their preparation. Visibility is the ability to predict a variance in planned cost, schedule, or performance. Control is exercised when action is taken to correct the variation. Visibility and control are management activities. The recording of project status can be automated to produce reports comparing actual status to planned.

Project control and reporting systems vary in the amount of reporting required and in the control exercised. Many such systems are variants of some larger, more generalized system, whereas others are simply procedural with informal or highly structured review of project progress at specified checkpoints.

program change control

The *program change control* function provides assistance to programming development activities such as integration test, release, and maintenance. Analysis of the impact of a proposed change on other parts of the system, as well as the portion to be changed, and the assurance that the change will be properly implemented rely on the ability to obtain accurate and current information on interface requirements over many levels of a system.

There are a number of ways to gain this ability. One approach is to retrieve interface information from data in a specified format supplied by programmers. Another approach is to analyze source programs for interface information. A third approach is to establish office and machine procedures to limit access to operational program modules.

Concluding remarks

The terms and definitions previously discussed introduce one to the categories and functions of programming tools and tech-

niques. It is hoped that this information will assist programmers and their management in the design and implementation of present and future systems.

An index to a selection of programming tools by category and function is next presented. This selection comprises those tools available to computer users as Program Products, Field Developed Programs, or programs classified as Type I, Type II, or Type III. ABSTRACTS of these tools then follow.

INDEX TO PROGRAMMING TOOLS BY CATEGORY AND FUNCTION

Modeling and simulation tools

- Simulative languages
  - CSS II
  - GPSS V

Measurement and evaluation tools

- Software monitors
  - GTF
  - SMF
- Timers
  - GTF
- Traces
  - GTF

Function testing tools

- Generators
  - OS utility IEBDG
- Test-result processors
  - SMFSA

Implementation tools

- Programming languages and coding aids
  - APL
  - PL/1 Checkout Compiler
  - PL/1 Optimizer
- Coding and debug systems
  - APL
  - COBOL Debug
  - CP-67/CMS
  - ITF
  - PL/1 Checkout Compiler
  - TSO TEST

Programming support tools

- Documentation tools
  - ATS
  - OS Flowcharts
  - OSFLOW
  - TEXT360

- Library maintenance systems
  OS utility IEBUPDTE
- Project control and reporting
  MINIPERT
  PMS IV

ABSTRACTS OF PROGRAM TOOLS

APL *APL/360*

- Function: interactive programming language
- Availability: Program Product
- Description
  APL is a conversational, time-sharing system based on a concise mathematical programming language with simple syntax. The system has a large set of primitive operations that work directly on arrays of information.

  There are two modes of operation: (1) immediate-execution, and (2) program definition. The system is designed to provide fast response to all requests. Current work may be saved between sessions. The system also allows users to create programming packages and to exchange programs and data online. Uses of the system include mathematical and statistical calculation, symbol manipulation, computer assisted instruction, and general data processing. Concurrent processing is possible in as many additional regions or partitions as are supported by the host operating system.

  - Programming systems

Operating systems:	System/360 OS/MFT, OS/MVT, and DOS
Programming language:	assembler
Mode of operation:	terminal-oriented, time-sharing using IBM 2740, 2741, or 1050 communications terminals

  - Documentation

	OS and DOS
APL/360 Users Manual	GH20-0906
APL/360 Primer	GH20-0689
General Information Manual	GH20-0850

ATS *Administrative Terminal System under OS/360*

- Function: documentation aid
- Availability: Type II
- Description
  ATS provides data and text input, editing and formatting capabilities which are time-shared among terminals. Text and data are stored on direct-access devices and can be retrieved at a terminal or other output device on demand. Free-form and fixed-format data may be intermixed. Changes to existing data are entered from the terminals with automatic realignment of the information set. Text-formatting functions such as right justification, page numbering, heading, and footing are also included.

  Background programs may be run simultaneously with ATS since it resides entirely in one partition or region. The system is open ended in that user-written application programs may be catalogued and called.

- Programming systems

Operating systems:	System/360 and System/370 configurations of OS/MFT (Version 2), OS/MVT, and DOS
Programming language:	assembler
Mode of operation:	terminal-oriented, time-sharing using IBM 2741 communication terminals

- Documentation

	OS	DOS
Program Description Manual	GH20-0582	GH20-0508
Application Description Manual	GH20-0297	GH20-0510

Online COBOL Symbolic Debug under TSO
COBOL Interactive Debug

COBOL Debug

- Function: noninterpretive debug system
- Availability: Field Developed Program
  Program Product
- Description
  COBOL Debug, when invoked under the Time Sharing Option (TSO) of OS, can monitor, direct, and debug the execution of ANS COBOL programs by referencing data and paragraph names. The facilities of the symbolic debugging system are available to the programmer when an external interruption occurs from the console, a program interruption occurs within the COBOL program, a breakpoint set by the programmer is encountered during execution, or an unrecoverable I/O error occurs. Some of the facilities available to the programmer are a paragraph trace option, display and/or alteration of data fields, data field address display, source statement display, snapshots of main storage, and the ability to resume execution at a paragraph or statement number determined by the programmer.
- Programming systems
  Operation systems: System/360 and System/370 versions of OS/MVT with TSO
  Modes of operation: terminal-oriented and time-sharing
- Documentation

FDP Program Description and Operations Manual	SB21-0284
Program Product General Information Manual	GC28-6454

Control Program-67/Cambridge Monitor System

CP-67/CMS

- Function: coding and debugging system
- Availability: Type III
- Description
  CP-67/CMS is a time-sharing system that provides conversational use of a terminal-oriented System/360 Model 67 configuration. The system has two components that may be executed independently of each others: Control Program-67 (CP-67), which manages the resources of the Model 67 and provides time-sharing to any system; and Cambridge Monitor System (CMS), which is a conversational operating system that provides its capabilities through a terminal command language.

  CP-67 builds and maintains for each user a virtual machine (a functional simulation of a real computer and its I/O devices) from a predescribed configuration. CP-67 allocates the resources of the real machine to each virtual machine, in turn, for a slice of time. Regardless of the virtual machine configurations, each user controls his machine from his terminal (his console keyboard).

  CMS gives the user the following capabilities at his terminal: creating and managing files, compiling and executing problem programs, and debugging. Because each user has his own copy of CMS residing in his own virtual machine, he is unable to affect other users or CP-67. Also provided is a batch monitor for program compilation and execution.
- Programming systems
  Operating systems: CP-67, a hardware control program
  CMS, an operating system
  Programming language: assembler

CSS II *Computer System Simulator II*

- Function: special simulation language
- Availability: Program Product
- Description
  CSS II provides a language for constructing models of computer systems to be studied. The language and structure of CSS II closely follow those of the system under study. Equipment operation, specified by means of control cards, is automatically accounted for by CSS II. System programs, both application and control, are written by the user in flowchart fashion using the CSS II instruction set. Program execution causes CSS II to simulate actual system operation. Output statistics, which quantitatively describe system behavior, are automatically and user provided. Examples of these statistics are job throughput, channel utilization, distribution of response times, and storage usage statistics.

  Output allows the user to determine whether the system meets defined operating criteria and points to strong and weak areas of the system. Effects of changes to existing systems can also be evaluated.

  CSS II can be used to model a wide range of configurations and systems—card, tape, and/or disk, multiprocessor, multiprogrammed, real-time, and time-sharing systems. Equipment operation including System/360 and System/370 configurations with rotational position sensing and cross-channel switching, the effects of control unit blocking, and storage interference are automatically accounted for by CSS II.

  Equipment that can be modeled includes multiple processors with shared main storage; selector, multiplexer, and block multiplexer channel operation; tapes; drums; disk storage (2302, 2311, 2314, 3330); 2321 data cells; punch card data processing equipment and general I/O devices; and communication lines and terminals with provision for handling various line speeds, input rates, and polling disciplines.

  Three features incorporated especially for time-sharing systems include the ability to perform the following: priority processing, creation of interruption conditions with an internal timer, and use of terminals in a conversational mode.

  Other features include a "help" facility that permits the user to add his own coded modules to the program to perform special operations. Also, a library facility is provided whereby standard models such as operating system routines can be added and easily called by the user.

- Programming systems

Operating systems:	System/360 and System/370 configurations of OS/MFT and OS/MVT
Programming language:	assembler
Mode of operation:	batch

- Documentation

Application Description Manual	GH20-0874
Program Description and Operations Manual	SH20-0875

GPSS V *General Purpose Simulation System V*

- Functions: general purpose simulation language
- Availability: Program Product
- Description
  GPSS V is a general purpose simulation tool for modeling and examining the behavior of systems in the engineering and management science areas. The user is able to explore alternatives and identify capacity limitations. Proposed changes to existing policies, methods, and operations can be subjected to critical performance criteria and evaluated. Free-form coding of GPSS statements and the ability to interface between GPSS and user-written PL/1 routines are among its features.

- Programming systems
  Operating systems: System/360 or System/370 configurations of OS/MFT, OS/MVT, and DOS
  Programming language: assembler
  Mode of operation: batch

- Documentation

	OS	DOS
GPSS V Application Description Manual	GH20-0825	GH20-0826
Introductory User's Manual	SH20-0866	SH20-0866

Generalized Trace Facility **GTF**

- Functions: time, trace, software monitor (hook catcher)
- Availability: Type I
- Description:
  GTF is an optional serviceability feature that assists in problem determination and diagnosis. It is invoked from the master console and operates in a region under the MFT and MVT control program options. Some of the functional capabilities provided by GTF include the following:
  1. optional tracing of system events either internally in the GTF region or externally to an I/O device;
  2. optional tracing of minimal or comprehensive data for system events
  3. selective tracing of only specific system events
  4. optional recording of user trace data in additional to normal system entries on tape or direct access storage.
- Programming systems
  Operating systems: System/360 and System/370 versions of OS/MFT and OS/MVT
  Modes of operation: batch, terminal-oriented, and time-sharing
- Documentation
  OS Service Aids GC28-6719

Interactive Terminal Facility **ITF**

- Function: interactive coding and debugging system
- Availability: Program Products
- Description
  ITF is a terminal-oriented, low-entry time-sharing system providing interactive problem-solving capability. Available programming languages are ITF-BASIC and ITF-PL/1 (a subset of PL/1).

  All system and terminal functions are controlled by the user from his terminal. Some of the facilities are:
  1. Program editing on a single-line or multiple-line basis
  2. Source statement syntax analysis
  3. User interaction with executing programs
  4. Direct access file I/O statements
  5. On-line error messages
  6. Debugging of programs via execution-monitoring aids
  7. Desk calculator mode
  8. Private and common libraries for program and data storage
- Programming systems
  Operating systems: System/360 OS/MFT, OS/MVT (including TSO environment), and DOS
  Programming language: ITF-BASIC and ITF-PL/1
  Mode of operation: terminal-oriented time-sharing
- Documentation
  OS/DOS and OS(TSO) Interactive Terminal Facility:
  PL/1 and BASIC General Information GC28-6825

MINIPERT *MINIPERT*

- Function: project control
- Availability: Program Product
- Description
  An interactive Critical Path Method (CPM) program, MINIPERT, an APL application, provides a solution to project scheduling problems. A project is treated as a series of interrelated activities done in parallel or serially to form a network. A critical path is the longest time path through this network to complete the project. All other paths have slack and are called non-critical. MINIPERT schedules critical and non-critical work to take the best advantage of available resources and to make the critical path as short as possible. Features of MINIPERT include a flexible calendar capable of specifying holidays and vacation periods, manpower loading, various reporting functions including diagrams and bar charts, and the ability to accept activity time durations in days, weeks, or months.
- Programming systems
  Operating systems: APL/360-OS and APL/360-DOS
  Programming language: APL
  Mode of operation: interactive and terminal-oriented
- Documentation
  Introduction to MINIPERT GH20-0852

OS Flowcharts *OS Flowcharts*

- Function: documentation aid
- Availability: Type III
- Description
  This program is an OS version of FLOWCHART, a DOS/360 program, which enables users to produce clear, standardized, and maintainable flowcharts.
- Programming systems
  Operating system: OS/360
  Programming language: F-level assembler
  Mode of operation: batch

OSFLOW *OS System/360 Flowchart: DOS FLOWCHART under OS*

- Function: documentation aid
- Availability: Type III
- Description
  This aid, a modification of FLOWCHART, a DOS/360 program (360A-SE-22X), gives Operating System users the flowcharting capability that is available under DOS. The program enables the user to obtain clear and reproducible flowcharts and to standardize flowcharting techniques.
- Programming systems
  Operating system: OS/360
  Programming language: assembler
  Mode of operation: batch

OS utility IEBDG *OS utility IEBDG*

- Function: generator
- Availability: Type I
- Description
  One of the OS utilities, IEBDG, generates a data set from a sequentially organized input data set. Repositioning and converting of fields within records are accomplished by programmer-specified utility control statements. User exits are also available.

- Programming systems
  Operating systems: System/360 OS/MFT and OS/MVT
- Documentation
  System/360 Operating System Utilities GC28-6586

OS utility IEBUPDTE

OS utility IEBUPDTE

- Function: library maintenance
- Availability: Type I
- Description
  One of the OS utilities, IEBUPDTE can create and update symbolic libraries. It can make changes to any sequential or partitioned data set containing records of 80 bytes or less. Also, the utility can change data set organization from sequential to partitioned and vice-versa. Other features are add, copy, and replace functions and the ability to assign sequence numbers.
- Programming systems
  Operating systems: System/360 OS/MFT and OS/MVT
- Documentation
  System/360 Operating System Utilities GC28-6586

PL/1 Checkout Compiler

PL/I Checkout Compiler

- Functions: programming language, interactive debugging system
- Availability: Program Product
- Description
  The PL/1 Checkout Compiler, a semi-interpretive language processor, is a companion product for the PL/1 Optimizer and is compatible in terms of source language and object time interface. Programs consisting of mixtures of optimized code and checkout interpretive modules are supported. FORTRAN and COBOL object module interfaces are provided.

  When supported under TSO, full interactive program development which includes monitoring of text execution and source program changes during test run is supported.
- Programming systems
  Operating systems: System/360 and System/370 versions of OS/MFT and OS/MVT
  Modes of operation: batch, terminal oriented, and time-sharing
- Documentation
  General Information Manual GC33-0003

PL/1 Optimizing Compiler

PL/I Optimizer

- Function: programming language compiler
- Availability: Program Product
- Description
  The PL/1 Optimizer is an optimizing compiler for the PL/1 language with language extensions beyond level F of the language provided under OS. The optimizer uses flowtracing, constant expression recognition, statement rearrangement, register use optimization, and increased inline code. Also, the optimizer is compatible with and designed for use with the PL/1 Checkout Compiler.
- Programming systems
  Operating systems: System/360 and System/370 versions of OS/MFT, OS/MVT, and DOS
  Programming language: assembler macroinstructions
  Mode of operation: batch

- Documentation

	OS	DOS
General Information Manual	GC33-0001	GC33-0004

PMS IV *Project Management System IV*

- Function: project control
- Availability: Program Product
- Description
  This system is a collection of programs that can be combined to provide critical-path and general cost analyses. PERT and PERT COST, precedence and precedence/cost, resource allocation, and report generation capabilities are also provided. The programs comprising the system are a network, resource allocation, cost, and report processors.
- Programming systems
  Operating system: OS/360
  Programming language: assembler
  Mode of operation: batch
- Documentation
  Application Description Manual GH20-0855

SMF *System Management Facilities*

- Function: software monitor
- Availability: Type I
- Description
  These facilities are optional features of OS/360 and OS/370 selected at system generation time in conjunction with the MFT or MVT options. Examples of data elements collected include job and job-step CPU time, main storage requested, main storage used, number of SYSIN records, number of SYSOUT records, EXCP counts for user data sets by unit address, start and stop times, and Time Sharing Option (TSO) terminal statistics. This data can then be used by user-written routines to perform job accounting or system analysis. In addition to job accounting, the gathered data can be used to determine items such as channel usage, number of initiators used, and time-sharing driver performance.

  SMF also provides control program exists that are used by user-written routines to monitor processing of the job stream. Exits are provided in the breader/interpreter, the initiator/terminator, the supervisor and the timer second-level interruption handler. Additionally, SMF supports both background batch and foreground time sharing processing when TSO is used.
- Programming systems
  Operating systems: System/360 and System/370 versions of OS/MFT and OS/MVT
  Programming language: assembler
  Modes of operation: batch and terminal-oriented
- Documentation
  OS/360 Planning for System Management Facilities GC28-6712

SMFSA *System Management Facilities Selectable Analyzer*

- Function: test-result processor
- Availability: Field Developed Program
- Description
  The SMFSA processes tape containing SMF data records produced by the System Management Facilities. Reports give job, job-step, and systems usage analysis through a generalized accounting routine for user billing. The system summary report is produced for each run of the SMF selectable analyzer. All other reports are selectable under operator control.

- Programming systems
  Operating systems: System/360 and System/370 versions of OS/MFT and OS/MVT with the SMF option
  Programming language: PL/1
- Documentation
  Program Description and Operations Manual SB21-0047

TEXT360 **TEXT360**

- Function: text preparation
- Availability: Type III
- Description
  TEXT360 is a text-processing system with data entry, updating, and page formatting capabilities. Input is free-form on punched cards. Output is on a line printer. Standard text processing functions are hyphenation, justification, double column alignment, and indentation. Also, horizontal and vertical rules for tables and figures can be generated. The system consists of the following four main processing programs: file maintenance, build line, page layout, and post processor. There are also four peripheral programs—prescan, print, spelling check, and dictionary update.
- Programming system
  Operating system: OS/360
  Programming languages: PL/1 and assembler
  Mode of operation: batch

Time Sharing Option TEST command **TSO TEST**

- Function: debugging facility
- Availability: Type I
- Description
  TSO TEST is an interactive test and debugging facility for use with object level programs. The TEST command allows the programmer to debug or test his program dynamically without special provisions in the source code. The system can be used to test user or control programs, allowing the display and modification of storage, general registers, and floating-point registers. The user may suspend execution of his program at specified locations, perform an analysis or modification, and continue processing. The programmer may also alter the environment in which he is testing his program by doing such things as freeing or getting storage, loading a program, or deleting a program.

  TSO TEST differs from some debugging packages in that it allows actual execution of the problem program rather than interpretive execution. This means that problem program execution may be terminated due to error conditions produced by the problem itself. Such a testing method provides a realistic environment for the program, since it does not degrade the problem program execution speed as occurs in interpretive executions.
- Programming system
  Operating systems: System/360 and System/370 versions of OS/MVT with TSO
  Programming language: assembler
  Mode of operation: terminal-oriented time-sharing
- Documentation
  TSO Command Language GC28-6732

ACKNOWLEDGMENT

The author wishes to acknowledge the many individuals in IBM who have helped bring together the ideas and concepts presented in this guide, and especially the help of the following. G. DeSalvo and W. Newman provided assistance in compiling the

list of programming test tools. Information regarding function testing came from W. Elmendorf and P. Schlender. R. Bender and E. Pottorf advised the author on program analysis techniques. F. Schulman and R. Phillips provided information on hardware monitors. Measurement definitions were contributed by K. LaCroix. Computer systems analysis topics came from M. Lehman, R. Hill, and W. Stanley. R. Merikallio assisted with software monitors. Modeling and simulation information was contributed by G. Pirtle and W. Duerksen. Assistance on simulation languages came from D. Braddock and P. Skiko. H. Mills provided information on programming techniques.

CITED REFERENCES

1. C. D. Warner, "Monitoring: a Key to Cost Efficiency," *Datamation* **17,** No. 1, 40–42, 49 (January 1, 1971).
2. H. N. Cantrell and A. L. Ellison, "Multiprogramming system performance measurement and analysis," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **32,** 213–221 (1968).
3. K. W. Kolence, "A Software View of Measurement Tools," *Datamation* **17,** No. 1, 32–38 (January 1, 1971).
4. P. S. Cheng, "Trace-driven system modeling," *IBM Systems Journal* **8,** No. 4, 280–289 (1969).
5. W. I. Stanley, "Measurement of system operational statistics," *IBM Systems Journal* **8,** No. 4, 299–308 (1969).
6. K. V. Hanford, "Automatic generation of test cases," *IBM Systems Journal* **9,** No. 4, 242–257 (1970).
7. F. T. Baker, "Chief programmer team management of production programming," *IBM Systems Journal* **11,** No. 1, 56–73 (1972).
8. E. M. Hahne, "An hypothesis with applications for total systems management," *A Forum on Systems Management*, Bureau of Economic and Business Research, School of Business Administration, Temple University, Philadelphia, Pennsylvania (June 1967).

GENERAL REFERENCES

C. D. Allen, "The application of formal logic to programs and programming," *IBM Systems Journal* **10,** No. 1, 2–38 (1971).

A. J. Bonner, "Using system monitor output to improve performance," *IBM Systems Journal* **8,** No. 4, 290–298 (1969).

M. E. Drummond, Jr., "A perspective on system performance evaluation," *IBM Systems Journal* **8,** No. 4, 252–263 (1969).

H. G. Kolsky, "Problem formulation using APL," *IBM Systems Journal* **8,** No. 3, 204–219 (1969).

P. H. Seaman and R. C. Soucy, "Simulating operating systems," *IBM Systems Journal* **8,** No. 4, 264–279 (1969).