Woman C.P.A.

Volume 40 | Issue 2 Article 8

4-1978

Electronic Data Processing: Systems Design and Application Programming

Elise G. Jancura

Electronic Data Processing

Systems Design and Application Programming

Editor, Elise G. Jancura, CPA, Ph.D.
The Cleveland State University
Cleveland, Ohio

Systems design and analysis has been defined as the analysis of present methods, applications, objectives, and all pertinent data followed by the design of improved systems using data processing equipment and techniques. The primary aim of this function is to produce the optimum solution for application needs; the most economic solution consistent with management and operating objectives, equipment capabilities and personnel resources. Programming has been defined as the translation of defined systems requirements and procedures into a logical process and a set of instructions for its operation on data processing equipment. Systems design and programming represent two separate though related functions in the data processing environment.

In some installations the same staff performs both functions, although in many of the larger installations these functions are performed by separate staffs. In some organizations the systems design group does not report to the computer installation but is attached instead to the various using departments for which the designs are being developed. This occurs because of the very strong need for the systems analyst to develop a full, detailed understanding of the information function and the needs of the user departments for which systems are being designed. Regardless of the organizational position of the systems development group — both the analyst and the programmers — it is important that those performing these functions have full access to the information available in the user departments and have the full cooperation of those departments as they attempt to study current procedures and design new procedures.

Systems and Programming Reliability

The management of an organization must clearly define and support the objectives that a system is to accomplish. It is impossible either to develop a system or subsequently to evaluate it if the objectives against which that system is to be measured are unknown. Proper control of the systems development procedure requires a formalized review and management authorization of each system before it is implemented. One of the important controls in the systems development process is appropriate authorization of the plan and evidence of regular appraisal and supervision of the progress of the systems development project.

An essential ingredient in efficient direction of the development effort and meaningful evaluation of its accomplishments is adequate documentation of the manual procedures and programmed procedures to be combined in the given data processing system or application. Evidence of close supervision such as regular progress reports, planning schedules, and management review reports can be used by the auditor in making inferences about the efficiency with which the development procedure was carried out, and more important, the accuracy and the adequacy of the resulting system. Authorization and approval of the system must be obtained not only from systems development management but also from the management of the operating or user department for which the data system is being designed.

Provision for error detection and correction procedures should be made at the time an application is designed. These detection and correction procedures may represent a mixture of manual operations and program checks, but efficiency demands that the systems analysts and the programmers incorporate those checking features as part of the original development work. These procedures should be fully documented, with specific instructions as to how errors are to be corrected and with specific provisions for reintroduction of previously erroneous data once the data has been corrected. The formalized procedures designated should also include operator instructions for both normal conditions and for error conditions such as label errors and machine errors, as well as for any special operating conditions.

Programming

Programs are a series of instructions

executed sequentially by the central processing unit of the computer. These instructions must be stored within the computer's internal memory and must be expressed in language that the computer understands. This language is frequently referred to as machine language or object language. It is possible for a programmer to write a program in machine language. However, this is a tedious, time consuming process, because computer languages are frequently complex and expressed in a coding structure different from that familiar to human beings.

To facilitate the programming process, a number of programming languages that are easier to use and are similar to language used by human beings are available. These languages are called symbolic or source languages. However, if a programmer writes in one of these symbolic languages, the programmer must first translate the language written in source code into the computer's language expressed in object code. When a computer vendor designs a language for convenient use by the programmer, the vendor also provides a program that will translate this symbolic language into object language. These translating programs are generally called compilers. In addition to performing the translation from source to object language, the compiler also produces the object program in a medium or form that is ready for loading into the computer.

There are several types of symbolic languages. Some of these languages require fairly expert knowledge of the computer system and its facilities and are rarely transferable to other systems. Others, frequently referred to as high level languages, require little knowledge of the specific computer, only general functional knowledge of computer systems in general. These high level languages are generally transferable from one computer system to another — a characteristic that is referred to as source compatibility. The language is implemented on individual computer systems through the compiler provided for each subsequent computer system. Examples of source compatible languages are FORTRAN, COBOL, PL/I, ALGO, and BASIC. A number of other generally source compatible languages have been developed for certain types of processing problems; LISP and SNOBOL are examples of list processing languages and GPSS and COGO represent examples of languages developed for simulation techniques.

An operating system is simply a collection of programs that allows control of the computer to be more fully automated. The heart of an operating system is a control program, frequently referred to as the monitor, that controls the automatic transition from one application program to another. In addition, the operating systems in a multiprogramming enviornment will control the concurrent execution of several programs within the central processing unit. An operating system usually means that the programs are stored in a machine readable library. The operating system may also collect operational statistics and keep a log of console operations.

Systems and Program Testing

The testing procedures performed in the systems development process can have a major impact on the subsequent reliability of that system, including both the manual procedures and the programmed procedures involved in the system. A definite plan should be prepared for testing the system. By testing in an organized and controlled fashion, much more accurate results can be obtained with a great deal less time and effort. It is important to make provision for testing every possible alternative processing path that exists within the system.

Test data should be prepared and run through the system for every condition that the system is designed to handle. Furthermore, test data should also include invalid data, in order to test the system's ability to recognize and segregate invalid and erroneous data. For example an accounts receivable program might expect to handle purchases and sales. Purchases might be designated by a 1 in the activity code and sales by a 2 in the code. The program should recognize that a record with a 3 in the activity code does not represent a legitimate alternative and should properly reject that transaction. A common failing of many programs is the assumption that if the first code does not exist, the record automatically represents the second code. Figure 1 illustrates alternative coding logic for program identification of the activity code. In Figure 1A the programmer erroneously assumes that if the code being tested was not a 1, that it was a 2. Mistakenly assuming that only properly recorded data reaches a program is a common error. Figure 1B shows a more desirable approach that tests the data as thoroughly as possible.

In installations handling large volumes of data, erroneous records can slip through the verification and other control procedures and reach the processing runs. Thus, in constructing the test data, the programmer or analyst testing the system should try to include examples of both valid and invalid data conditions. Further tests should be run with essential data missing (fields within records or even whole records), to test whether the programs can detect the absence of needed information. For sequential files, one test should include test data that is out of sequence. For files that need particular activity codes, tests should include data with codes that are incorrect as well as examples of all correct codes. When a program that has been properly executing for some time suddenly begins producing an erroneous result, a good probability exists that the program was incorrectly tested and that a relatively rare error condition occurred that was not recognized in the original test data. Thorough testing is such an important part of a systems development, it should receive major consideration.

Wherever possible, it is desirable to test in two phases: in the first phase, specially constructed test data is used; in the second phase, the system can be tested with actual data as it is being generated in the installation. This latter approach is called a parallel run, because this execution of the new system should be concurrent with continued execution of the old system. The purpose of a parallel run is to test the entire system in an actual operating environment. This can serve two purposes: to insure that the systems description and design accurately reflect the actual environment; and to check that the related programs function together as an cohesive system.

However, the installation should always be aware of the fact that successful parallel runs do not necessarily guarantee a complete test. When the new system differs substantially from the old, the results may not be comparable without extensive reconciliation. Also, in most environments there are relatively rare combinations of data conditions that very infrequently occur in the normal course of events. Unless the parallel run should happen by chance to occur when one of these rare combinations exist, the system will not be tested for its ability to handle such a situation. This is the reason why the

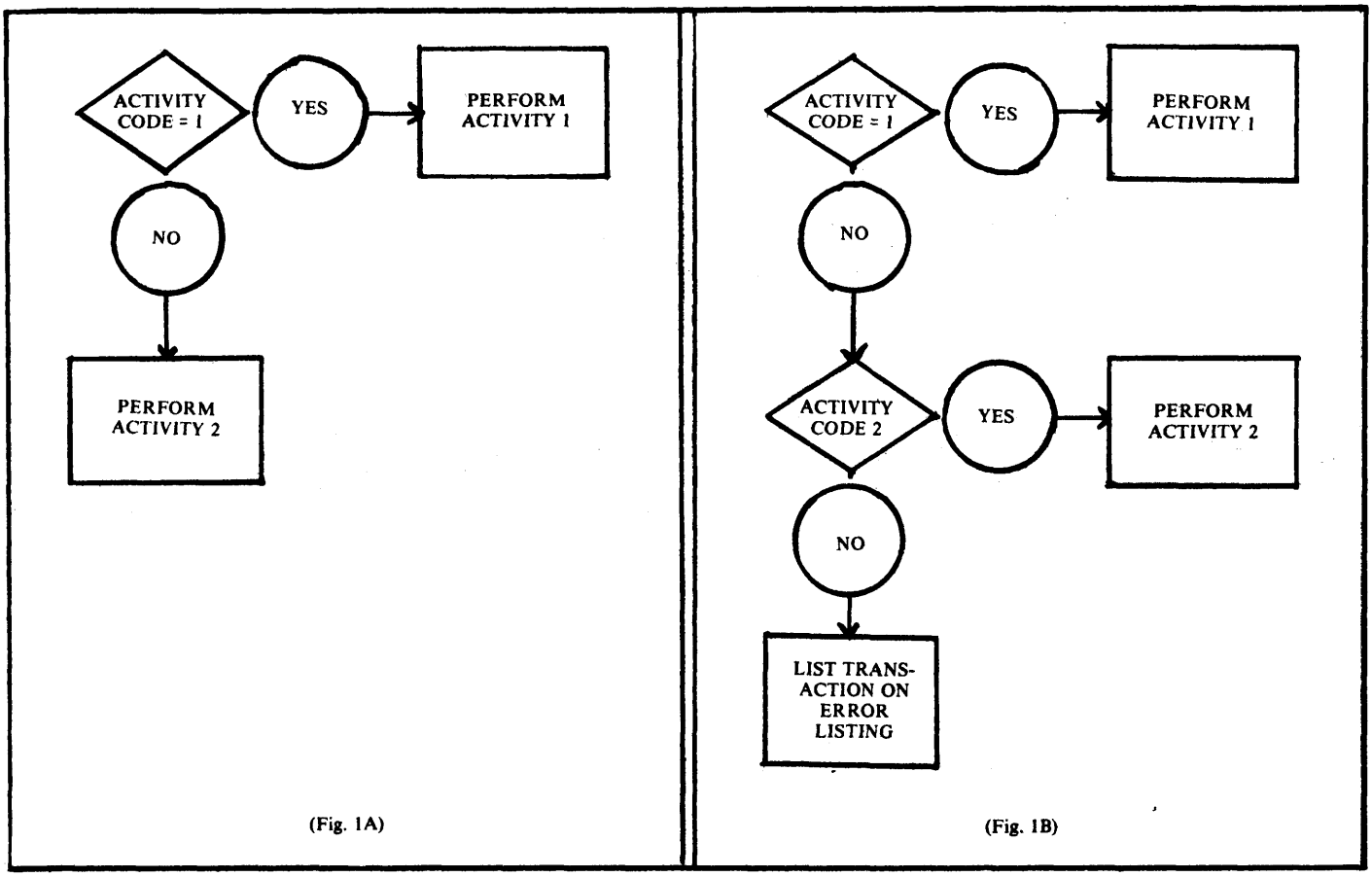

Fig. 1 — ALTERNATIVE CODING FOR ACTIVITY CODES

construction of formalized test data is so important.

Control of Programs and Program Libraries

Plans should be specified and fully documented as to the way in which individual programs are to be stored and maintained within the installation. This includes plans for control of access to the program libraries as well as specific provisions for program modifications. One test of a well controlled installation is the degree to which prior authorization is required for program changes and the degree to which any changes made to computer programs, once these programs are placed in a normal operational state, are documented.

An all too frequent error made by installations is to allow changes to the program without correction of the documentation for that program. Another serious error consists in allowing unauthorized access to an operating version of a program, thus making it possible to change the function of a program. The problem of controlling changes to programs must be addressed at several levels. First the systems development process must plan ahead and define the procedure for making authorized changes, with the attendant requirements for proper documentation. Second, appropriate operational control must be instituted to prevent unauthorized intervention from the computer console when a program is loaded for execution that could result in undocumented and therefore possibly undetected change in the program.

Depending upon the sophistication of the installation, copies of the programs can be in the form of card decks and stored in appropriate filing drawers. Most installations, however, have progressed to some sort of operating system environment in which the machine language copies of the programs are usually maintained in a machine accessible library, usually stored on a disk file. Access to these progrmaming libraries must be controlled both in terms of execution of the programs within these libraries and in terms of any changes made to the copies of the programs that exist in the working library. Periodic examination of the programs stored in the working libraries (which are libraries from which the programs are actually loaded and executed in the computer) is a necessity to insure against any unauthorized changes in the programs as they are executed from the specifications found in the installations formal documentation.

To review briefly, programming in systems reliability will be greatly enhanced if processings systems and their programs are designed to meet carefully defined objectives, are properly authorized by the organization management, have heavy user involvement in the design and testing phase, are adequately documented, make provision for control changes to the system, and are thoroughly tested.