The testing problems in a complex systems environment are described with categorization by purpose of the tests and objectives to be achieved. The approach to testing and the methodology required to adequately test in the various testing categories are presented.

The methodology described, together with appropriate testing tools, can aid users in the migration to new operating systems and new versions of subsystems, as well as to accelerate their own application development.

Testing in a complex systems environment

by M. O. Duke

Testing in a complex systems environment includes, but is not limited to, the planning, preparation, control, execution, analysis, reporting, and correction of tests and procedures. As far as practical, the testing is accomplished through the use of automated test tools, including a test driver.

A complex systems environment as discussed herein is one in which a telecommunications network is attached to a computing complex. The computing complex is under control of an operating system and has one or more subsystems, such as IMS/VS (Information Management System/Virtual Storage), executing and communicating with remote terminals via the telecommunications network. User-written application programs, under control of the subsystem, execute upon demand from the remote terminals. Batch processing may also be performed concurrently. For testing, this environment is replicated in a laboratory environment, so that all physical portions of the environment need not be present.

Testing in this complex systems environment requires both a definition of the purpose of the testing and a knowledge of the methodology which can be used to accomplish the purpose that has been defined.

The purpose of testing is really two-fold. Testing is conducted to verify that programs accurately perform the functions they are intended to perform. Testing is also conducted to determine the

performance characteristics of the programs. The result of appropriate testing is the assurance that the programs will execute both correctly and speedily. This determination in a test environment prevents the chaotic situation of having programs fail in a production environment where errors can be catastrophic.

Testing in a complex systems environment consists of creating specific collections of tests that are stored on direct-access storage devices and accessed by a test driver. These tests duplicate the actions of humans at remote terminal hardware to exercise the programs under test. Information is obtained for analysis to ensure that the programs have performed correctly and well.

Because different approaches to testing are required, depending upon the purpose of the test, an understanding of the varieties of tests is necessary.

Test categories

Two major categories of testing exist, together with subcategories that more precisely define the testing. The two major categories are: (1) functional testing and (2) performance testing.

One major distinction between the two types of testing is that functional testing depends upon an exhaustive one-for-one test of each cause that can produce an effect or a test for each and every condition that can occur. Performance testing, however, depends upon the repetitious use of specific tests in an attempt to recreate or to simulate a specific production environment. The specific tests are derived from a "user profile" which must be obtained if meaningful tests are to be conducted. This "user profile" is discussed later.

functional testing

Functional testing can be divided into the following subcategories: new function testing, functional regression testing, and functional stress testing.

New function testing refers to the testing that must be done to a new program or to a new function that has been added to an existing program. For each new function added, a test case, or series of test cases, must be prepared to exercise that new function and ensure that it operates correctly. In addition, all of the possible error conditions that could occur should be tested to ensure that the error conditions give the appropriate error messages and that those messages correspond with the documentation associated with the program.

Functional regression testing is a term that applies to testing programs, when a part of the total environment has been modi-

fied, to see that the functions provided by the programs remain intact and that there has been no change in the functions that had previously been available. In this context, programs can be thought of as being small application programs, major application systems, a subsystem such as IMS/VS, or the Operating System itself.

In order to accomplish complete functional regression testing, all of the test cases that were run against the programs in the previous environment must be rerun against the programs in the new environment. Each test case must produce exactly the same results as before, except for known compatibility and timing dependencies (e.g., time, date).

If a program is a very large one, with a great deal of function, then the number of test cases will likewise be large. There may be several hundred test cases, each of which executes some function provided by the program or some error condition that is being tested, to ensure that errors are located and corrected in a test environment and not when the program is run in a production environment. The collection of test cases is commonly stored in a test library.

The fundamental concept of regression testing is that each and every test case that was run successfully against the unmodified code must produce identical results when run against the new code, except for the differences noted. Obviously, if the library of test cases is very large, the process must be automated, since the resources required to permit each test case to be run manually is very great. Unless all of the results compare precisely (except as noted), then regression has occurred in the code, and the new code must be corrected to restore the function that was available prior to the modification.

Functional stress testing is the term that applies during functional testing when quantities of tests are executed concurrently to ensure that the required function operates properly even when the programs are heavily loaded. Often programs will execute properly under minimal loading but will fail under heavy loading due to such things as timing or resource conflicts. Functional stress testing is used to locate these failures.

performance testing

Performance testing, by its very nature, depends upon a "user profile" that depicts not only the types of messages that are to be transmitted but the frequencies with which specific message types are transmitted. The "user profile" also includes message content, transmission speed, and intermessage delay or "think-time".

The user profile can be obtained from analysis of a production environment through the use of a log such as the IMS/VS log. Information from this log can be extracted to provide a test stream that can be used for performance testing.

Performance testing can be divided into the following subcategories: performance load testing, regression performance testing, expanded network or projected environment simulation, and performance measurement testing for tuning purposes.

The objective of *performance load testing* is to determine whether or not the programs can indeed handle the planned load with an adequate throughput and acceptable response. Bottlenecks and queuing problems can be located in this testing environment, rather than in a production environment. An appropriate user profile is essential to this testing. Otherwise, the programs may appear to execute adequately but actually fail to perform in the production environment.

Regression performance testing is conducted when there is some change in the environment, either through the addition of new programs, modification of existing programs, or the installation of new hardware, operating systems, or subsystems. This testing is conducted to determine the extent of change to throughput and response when the environment is modified. If the throughput and/or response is no longer acceptable, other changes are mandated.

Expanded network or projected environment simulation is testing that is done in order to place a projected larger load on the programs to ensure that the appropriate number of terminals can be supported and that the quantity of transactions anticipated can be handled. In order to accomplish this testing, it is necessary that the anticipated number of terminals and transaction loads be applied over a sufficiently long period of time so that stabilization can occur, measurements can be made, and conclusions can be drawn. It is practically impossible to accomplish this manually.

The expanded network or projected environment simulation ensures that the planned load can be handled. It also shows areas where bottlenecks may occur. This knowledge then adds to the predictability of the effect of change.

Performance measurement testing is done to locate critical bottlenecks in the system and to assist in tuning. The environment is contrived, in that certain parameters are being pushed to their limits, to locate elbows in curves, which are associated with critical variations in parameters.

It is possible to examine the effect on response and throughput when a parameter is varied (e.g., CPU loading). In this case, a contrived environment is produced that will cause CPU loading to occur, and measurements can be made that show the effect on response and throughput.

This contrived environment is in no way representative of any production environment and should not be considered as such. It is an experimental environment where causes and effects are being studied to obtain additional knowledge of the total environment so that modifications can be made to the environment, permitting the system to operate more desirably.

Test methodology

The methodology to test in these several environments consists of planning the testing, test preparation, control of testing, test execution, post-test analysis, and corrections.

planning the testing

A thoughtfully developed and well-documented test plan is essential to successful program installation and maintenance. It is also essential to the goals of the test, and should minimize the expenditure of resources. This plan should take into account all aspects of the testing and should include:

- Testing systems.
- Test tools.
- Test scripts.
- Data bases.
- Expected results.
- Schedules.
- Checkpoints.
- Resources.
- Dependencies.
- Expected problem areas.

Comprehensive testing involves great quantities of data in the form of data bases, test cases, scripts, expected results, and statistical information. To create all of the data, to execute the test cases, and to analyze the results, requires a major expenditure of human effort and machine resources. The use of these resources must be carefully planned.

The test planning must begin very early in the development cycle for two major reasons. First, there are the design considerations of the program to be written. Testers look at the specifications from an entirely different point of view than do the developers. As such, they can frequently spot flaws in the logic of the design or incompleteness in the specification. In doing this, they help the developers to produce a superior product.

The second major reason is the scope of the effort required in the actual testing itself. A large number of test cases will have to be prepared. Data bases will have to be built. Expected results will have to be generated. The test cases will have to be consolidated into scripts. Even the planning is quite time-consuming. Without an early beginning, there is often a crash effort to conclude by a scheduled date, with quantities of overtime and the expenditure of large quantities of machine time to attempt to meet schedules.

The plan should be in sufficient detail that other persons, such as other testers, managers, and developers, will be able to determine the purpose of each portion of the testing, the magnitude of the testing, the results that are expected, and how the effectiveness of the test will be determined. Even with automated procedures and tools available every step of the way, it is still possible (probable) that a large expenditure of both human and machine resources will be required to adequately test a program or a subsystem.

Since it is usually not possible to run all varieties of tests during the same machine run, a careful selection must be made, and test runs must be performed in an intelligent manner. Good testing depends upon a careful and thoughtful selection of the tests to be run and the environment associated with running the tests. Any available automated tools should be used to accelerate testing, reduce errors, and minimize costs.

test preparation

Before any testing can occur, many pieces must be in place. The operating system must be decided upon and generated for the system. Any subsystem that is associated with the test must be generated and incorporated within the system under test. Application programs that will be used must be available. Data bases must have been created or must be available from some source.

It is highly likely that a version of an operating system will already be available. A copy of this operating system can be obtained as a starting point for use by test personnel.

It is also likely that the operating system will have to be modified in some way, or a new system generation performed to create the right environment for testing. For example, the current operating system may be generated for a certain number of terminals and specific subsystems that are required in a production environment. If a new subsystem is to be added, modifications to the system generation environment may be required for the test. It is also possible that a new operating system is about to be installed and that it must be generated and tested, along with all of the subsystems and programs that existed on the previous

operating system. The same is true for Network Control Program generation when needed for a communications controller.

If a new subsystem is to be installed, it must first of all be generated and added to the current operating system or to a new operating system. This generation process must take into consideration all of the requirements of the subsystem, such as space requirements in main storage and on disk and the number of terminals that the subsystem is expected to support.

For projected environment or expanded network simulation, it is likely that larger versions of the subsystems will have to be generated to support the planned additional terminals and lines.

preparation components

Preparation for functional testing depends to a large measure on an explicit methodology that includes:

- Identification of test conditions.
- Test case development.
- Script preparation.
- Data base preparation.
- Driver run preparation.

Preparation for performance testing substitutes tests that are based upon the user profile for test condition identification and test case development. Intelligent testing depends, to a large extent, upon the understanding of the use of this methodology.

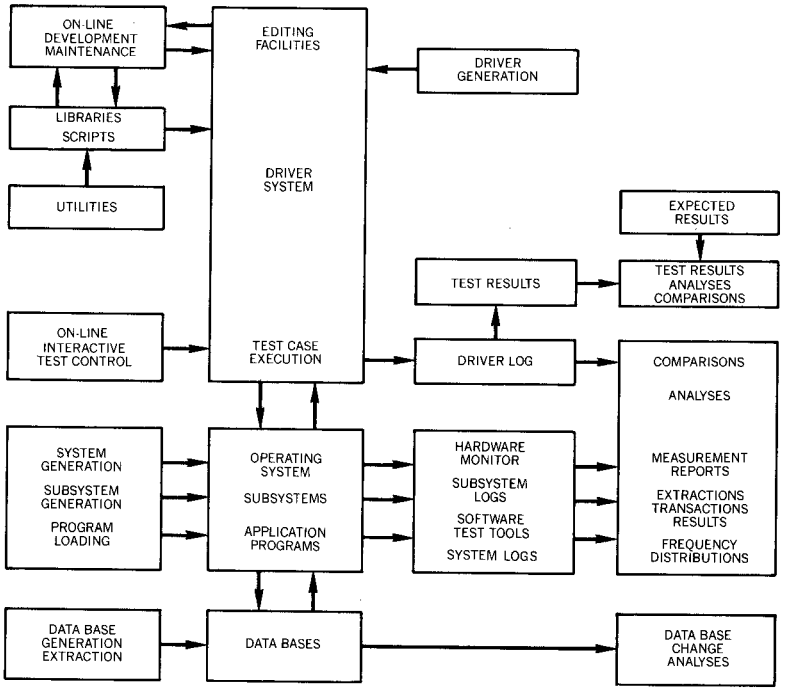
Using a test program's functional specifications as a base, every unique *test condition* is identified wherein some specific cause results in a known effect. For example, one specific condition results in a given error message. Each parameter or variable has one or more correct values and one or more incorrect values. Each value is a unique condition and can result in correct execution or in an error condition. Each test condition must be identified and documented together with the expected result that should occur when the condition is encountered.

In its purest form, one *test case* represents the test of a single test condition. In practice, however, a test case is developed to test one or more test conditions. The important concept here is that each test condition has one or more test cases that cause that condition to occur.

A test case, by itself, may not be executable in an automated testing environment. It becomes executable when it is consolidated into a script.

A *script* is a collection of one or more test cases, together with sufficient identification and control information to permit that

Figure 1 Test environment



script to be executed from a test driver. The script may contain unique tests for use in functional testing (and possibly performance testing) or it may contain repetitive tests for use in performance testing. A script is typically stored in a test library and is accessed during the execution of the test driver.

Data base preparation is necessary whether one is testing a new processing program, a new subsystem, or new features in a subsystem such as IMS/VS. A data base needs to be created prior to running any test scripts. That data base needs to be of sufficient size, content, and complexity so that the functions that the programs are attempting to perform and the test scripts that are exercising the programs can indeed perform the required function. The content of the data base needs to be known so that processing against the data base that causes changes to that data base can be identified, and those portions of the data base that are changed by the testing can be identified.

The data base can be generated manually, it can be a copy of an existing data base, or it can be generated automatically with some existing tool. In any case, the content and structure of the data base must be known.

Preparing for execution of the test driver (*driver run preparation*) consists of the identification of network components, such as terminals, clusters, and lines, of the test scripts

that will be executed, and of other resources that will be used, such as the name of a logging data set. This preparatory information may be divided so that some of it exists within the test libraries and some is entered when the test driver begins execution. Typically, the test scripts are associated with system resources and may be used once for functional testing or repetitively for performance testing.

test control

Figure 1 depicts the test environment. Many components are illustrated that are data base-oriented. Many are program-oriented. Each component must be identified and controlled in order to minimize testing costs and to ensure that test schedules are met.

Test execution

Test execution using a test driver, or "driving" as the term is commonly used, is the process whereby a test driver program executes in one region of a computer, simulating the action of terminals and communication hardware and causing another program or subsystem to react to the simulated terminal input. The other program or subsystem is being "driven" in that all input/output is identical to that which would occur if persons at terminals were causing transmission to occur, rather than a program causing the transmission.

If the "driven" program or subsystem resides in the same computer as the test driver, then simplex driving is occurring. If, however, the "driven" program resides in another computer, then duplex driving is occurring. In both cases, the "driver" and the "driven" program are interconnected through a telecommunications network.

simplex driving

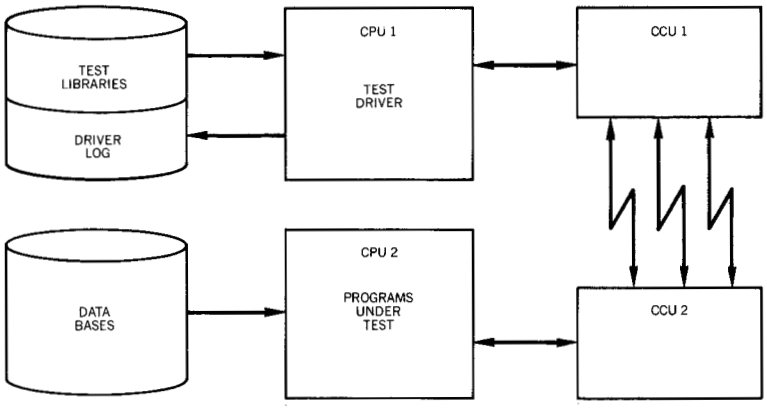
Simplex driving is the term attributable to the environment where both the driver and that which is being driven reside in the same computer. The driver resides in one region and the program or subsystem being tested resides in one or more other regions in the same computer (Figure 2).

Simplex driving is desirable for functional testing since another computer is not required, therefore reducing costs. It is less desirable for performance testing since the effects of the driver must be adjusted for in the analysis of the performance data acquired.

duplex driving

Duplex driving is the term attributable to the condition where the driver resides in one machine and that which is being driven resides in another machine. This is the situation that usually exists when performance tests are to be made and accuracy is necessary (Figure 3).

Figure 2 Simplex driving



dynamic modification

When drivers are used to do functional testing, the test cases are accessed serially. During normal testing, it is often true that certain test cases will fail and may bring the test to a halt unless they are corrected or bypassed.

One of the useful features of a driver is the ability to permit the tester to modify both test cases and scripts in order to correct or bypass certain test cases without completely aborting the driver run. Otherwise, each test case failure causes the driver run to cease. The test would have to be corrected off-line and run at a later time. The ability to modify both test cases and scripts dynamically can therefore save precious days or weeks of testing when functional testing is being performed.

logging

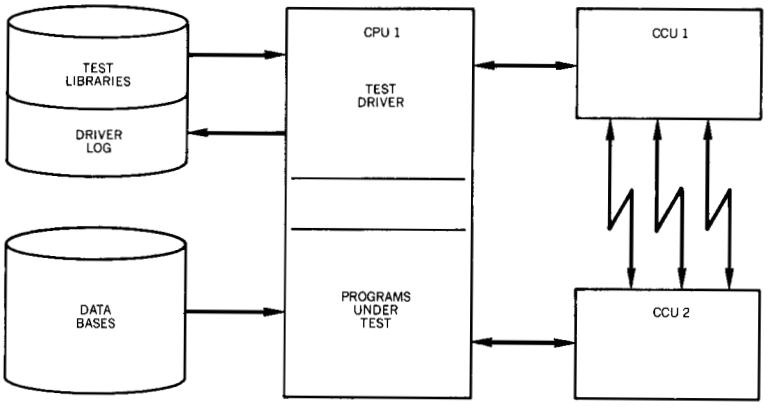
The test driver log records and time stamps all messages sent from the driver to the "driven" programs and likewise records and time stamps all messages received in reply. In addition, it records the test environment and other indicative information.

The driver log permits post-test analysis for both functional and performance tests to provide verification of functional correctness, and to provide performance information as well.

Post-test analysis

Post-processing is the term applied to the analysis of test runs. This analysis can be in the form of comparisons between expected results and actual results, of comparisons of data bases as they exist with what was expected, of resource usage, of frequency distributions, or of any other processing that verifies that the test ran correctly or that verifies the performance of the program and/or subsystem under test.

Figure 3 Duplex driving



actual versus expected results

At the completion of any functional test driver run, it is imperative that the results actually obtained be compared with the results that were expected from the run. The results of each and every test case must be verified to ensure that the code executed properly. If any discrepancies are noted, they must be resolved.

The most difficult part of the comparison process is ensuring that each and every test case output was identical to what was expected. If this is a manual procedure, involving hundreds or even thousands of test cases, then it is a laborious undertaking and requires many hours of human examination to ensure accuracy. If, however, it is an automated process, then comparisons can be done quickly, and discrepancies can be noted in the form of exception reports, showing what was expected, what was actually received, and how the two differ.

data base changes versus expected changes

When a test is run, both the output and the data base changes must be analyzed to determine the validity of the test. Just as the expected results must be compared with the results actually obtained, when data bases are involved it is imperative that the data base changes that occurred during the testing be compared with the changes to the data base that were expected. It is important to know that the data base was changed in the exact manner expected and that no other changes were made.

The data base change comparison can be done manually in a very time-consuming manner, or it can be automated to the point that the comparisons are done with computer programs. With an automated comparison, every discrepancy is noted, and any problems can then be pursued to determine the cause of the discrepancy.

output

When performance test runs are made, data are available from many sources. System data is available. An IMS/VS log tape is available if IMS/VS was one of the subsystems under test. Hardware monitors may have been attached, and if so, data from these monitors are available.

All of the data received from the many sources can be analyzed. The time that each event occurred must be determinable and preferably time-stamped on the recording as it is made. All of the information can be sorted into time sequences prior to examining the information. Adjustments must be made for any discrepancies in clock recordings.

Statistical information gathered during performance runs must be available upon completion of those runs to show the usage of individual resources within the computer complex. Much of this information can be available by processing subsystem log tapes, system logs, and hardware monitor information. It is normally displayed in a series of reports in order to present it in a meaningful fashion.

Part of the post-analysis procedure associated with performance testing is an analysis of the loading of the system at periodic intervals. The system may have been lightly loaded at certain times and heavily loaded at other times. It is nearly meaningless to know what the average loading was. The peak loading may or may not be of value since this peak may have been an instantaneous peak, which was accidental and therefore would not represent the normal peaking.

It is much more meaningful to be able to distribute the factors over time to show the periods of time when the system was loaded and the percentages of time that specific resources were in use. This frequency distribution, then, has the effect of permitting one to examine a complex system over a prolonged period of time.

corrections

As a result of the post-test analysis, it will be seen that some errors occurred because of deficiencies in the programs being tested. Others will have resulted from incorrect test cases or incorrect expected results. Where the error is with the test preparation, corrections must be made there. When the error is in the programs under test, those errors must be communicated to the developers so that the programs can be corrected. Adequate control dictates that procedures be in place for follow-up to ensure that the programs have been corrected and have been retested without regression.

Summary

With a thorough understanding of the various types of testing, of the objectives or purposes of specific tests, and of the test tools and test methodology, it is feasible to conduct meaningful tests in a complex systems environment. The major test tools are the test driver itself, the test libraries, the ability to edit or modify tests, and the post-test analysis programs.

By utilizing these test tools and the methodology shown, both functional and performance tests can be conducted. The combination of tools and methods can accelerate program development, aid migration, ensure that accuracy and performance goals are met, and reduce costs by the replacement of manual methods with the automated testing techniques described.

ACKNOWLEDGMENT

The author gratefully acknowledges the many creative suggestions and support contributed by D. M. Langston and S. Togasaki.

GENERAL REFERENCE

Information Management System Virtual Storage (IMS/VS) General Information Manual, Form No. GH20-1260, IBM Corporation, Data Processing Division, White Plains, New York.

Reprint Order No. G321-5021