Technical note—
The IBM TotalStorage Enterprise Storage Server: Testing for general availability and beyond

by M. R. Dillon
P. M. Collins
P. R. Hurley
J. E. Handlin
S. A. Reed

The IBM TotalStorage™ Enterprise Storage Server® (ESS) is a powerful and versatile storage subsystem that can respond well to the explosion in demand for on-line data storage. ESS provides unique features in the areas of connectivity, capacity and performance scalability, data integrity, reliability, availability, and serviceability. This technical note discusses the stages of ESS hardware and software testing. Focus is then given to the collaboration, tools, and techniques needed for testing ESS software support provided specifically for the System/390® platform.

Testing the IBM TotalStorage* Enterprise Storage Server* (ESS)—both the hardware and the supporting software—is a daunting task. ESS supports a large number of programmable features, heterogeneous servers and their attached devices, attachment methods, and operating systems. The complexity of the hardware and software testing required is considerable, in light of a very rich function set (including, for example, Peer-to-Peer Remote Copy [PPRC], Extended Remote Copy [XRC], remote services support, and Concurrent Copy). There are also a number of performance-enhancing features for System/390*, for example, priority I/O queuing.

Additional complicating factors are the configuration and operations tools provided with ESS, which include Web-based configuration tools, service log-on tools provided for service personnel use such as ESS Net Console, a server-based command line interface tool for invocation of scripted copy operations, and a server subsystem device driver for concurrent service operation.

The ESS project presents a range of engineering testing challenges, requiring team skills in multiple disciplines, education in numerous system environments, intensive code driver delivery schedules, considerable product complexity, and the difficulty of stressing the machine in various configurations and types of I/O activity. As ESS host attachments increase in variety of host types and adapter types, the number of host attachment test scenarios is exponentially increased. The development team also has to be creative in its approach for testing ESS in multiple customer environments in order to support the "time-to-market" goals of IBM and to meet customer requirements, while providing a highly reliable and stable product.

ESS can to be attached in various ways to a variety of computer architectures. For example, System/390 uses ESCON* (Enterprise Systems Connection) and FICON* (Fiber Connection) to attach to ESS and views its ESS volumes as CKD (count-key-data) format devices. The other server types (i.e., other than System/390), referred to by IBM as "open systems," attach via SCSI (Small Computer System Interface) and view their devices as fixed block devices. As a result, ESS testing requires several different host types (not to mention hosts from many different manufacturers). Also

requiring variation are hardware adapters, device adapters, and the hard drives themselves.

In the following sections, we describe the design and execution of ESS hardware and software tests. Although much of the test implementation is tailored to the unique requirements of ESS and its applications, our experience may be instructive to testers of a wide variety of systems.

ESS hardware testing

In the following, we describe in detail the ESS hardware testing: its objective, test design, and the hardware development verification tests.

Testing objective. The objective of our hardware development testing is to provide a technical evaluation of storage products. Results of that technical evaluation are utilized to support key product checkpoints (such as announcement, early ship program [ESP], and general availability [GA]) and to support post-GA field problem resolution and the release of product enhancements.

The breadth of ESS product features and attachment types (as well as attachable computers, operating systems, and network topologies), makes testing challenging, due to the number of possible interactions and permutations. Test planning also becomes critical. Flexibility must be built into the test plans in order to adapt to potentially changing conditions. Test planning and methodologies must anticipate and work around frequent and recurring bottlenecks in device and software availability.

In addition to the formal test objective, the hardware testing for ESS is organized around the following key principles.

The process itself must be subject to analysis. Both the product and the testing processes must improve over time. Criteria for the quality of testing are defined and evaluated. Progress is monitored throughout the test (see the subsection "In-process metrics"), allowing the team to make "in-flight" corrections to test scenarios. These metrics are combined with the adoption of postprocess methods, experience reports, and postmortem evaluations.

Testing must consist of both function-oriented and task-oriented approaches. Later stages of testing—which increasingly involve the integration of subsystem elements—must be oriented toward actual customer usage, task scenarios, and solution testing, rather than individual function verification.

There must be room in the test plan for "creative testing." Such testing capitalizes on the improving skills of the test team and gives them room to modify test procedures.

There must be cooperation between the hardware and software test teams, for the benefit of both. Customers buy system solutions, not isolated elements. As illustrated by the testing of the DFSMS (Data Facility Storage Management Subsystem) Small Programming Enhancement (SPE), which is needed to support testing of ESS, this kind of cooperation is critical to the successful delivery of ESS.

Hardware test design. In this subsection, we describe our experiences in planning and implementing the ESS hardware test suite. ESS testing follows the industry-proven best practices of planning, preparation, and execution.

Test planning normally begins when a product such as ESS becomes part of the official product plan and with completion of the design documents—primarily the functional specifications for the product. Draft plans are developed and reviewed, and issues are tracked and resolved prior to test plan acceptance. At this point, detailed schedules are set in place and dependencies are cross-checked.

The preparation phase involves setting up labs, acquiring software and hardware tools, preparing status-tracking databases, and doing detailed test design. In some cases, this phase also includes "lessons learned" from the planning phase.

For each test, execution begins when all of the entrance criteria documented in the test plan for that test are met. Each hardware test ends when its exit criteria, as documented in the appropriate test plan, are met. However, some tests continue after exit criteria are met, for the purposes of fix verification or regression testing. This extended test scope covers the verification of functions added after the initial product plan, as well as boundary testing, whitebox testing, and fix verification. Boundary testing emphasizes the limits of allowable parameters and extremes in environmental conditions. Whereas blackbox testing considers only the inputs and outputs of the system under test, whitebox testing also takes into ac-

count some known internal characteristics of the system under test.

In-process metrics. The use of in-process metrics is a common practice for both hardware and software testing. Some of the metrics that are used during ESS hardware and DFSMS software testing are power curves, defect charts, and defect analysis.

Power curves track the expected versus actual progress of test case groups against time. The usual practice is to track test attempts (planned and actual) as well as test successes (planned and actual).

Product defects found in each test are recorded both by component and by type of test. The rates of detecting and resolving defects, the severity of the defects, and the time between detection and resolution of defects are all tracked. Defects are prioritized by severity and, in some cases, this may lead to reordering of the testing. Defects in the tests themselves are also recorded. Defect rates and categories are later used with the Orthogonal Defect Classification (ODC)[1] process as one measure of test effectiveness.

Several layers of defect analysis are done during and after product development. During ESS hardware and microcode development, all defects are classified by customer impact and probability of occurrence. This provides input to the development team as to the severity of the problem from a customer standpoint. This information, along with the ODC process (which helps measure test effectiveness), provides developers with tools to review the defects identified during test cycle and field life, to enhance or change the development process.

After the product cycle is completed, testing continues with escape analysis. Escape analysis is a standard engineering practice that treats problems found by customers after general product availability as potential testing escapes—problems that should have or could have been found during product testing. For ESS escape analysis, each problem encountered in the field is evaluated as follows, once the problem is resolved. The problem is evaluated to determine if it was a hardware failure or code problem. Hardware problems require a failure analysis to determine what caused the hardware failure. In some cases, the problem is so convoluted or requires so many multiple fault conditions that it cannot be found in normal/realistic testing. In the remaining cases, a determination is made as to why the problem was not found, and the testing process is then modified or expanded to eliminate this type of escape in the future. Both hardware failures and code problems may necessitate additional test scenarios for the appropriate areas of test, or a modification of an existing test scenario to perform a test in a different manner or sequence. Escape analysis is an ongoing process throughout the life of a product.

Hardware development verification. Hardware testing for ESS is conducted in a variety of modes. Some tests—for example, the device-level failure test—correspond directly to the architecture of the storage system; some tests—for example, the system-level test—represent a point in development when lower-level hardware pieces come together for the first time. Other tests—such as the system-level-serviceability test—are included as an engineering "best practice."

The following subsections describe the most significant of the test modes used in ESS hardware testing.

Hardware microcode integration test. The hardware microcode integration test provides initial validation of hardware and code integration on physical hardware. A set of scenarios is executed to confirm that a minimum level of functionality is met before delivery to engineering verification test (EVT).

Engineering verification test. The engineering verification test verifies function against the approved functional specification and design documents particular to ESS. EVT verifies that any new functions introduced late in the product cycle do not cause previously available functions to regress and do not adversely affect any other function. EVT is important for test history when doing gateway testing, allowing the team to refer back to EVT results when performing gateway tests.

Gateway test. The gateway test provides the pass-or-fail validation to each new code link or microcode level delivered to test during the test cycle. Gateway testing provides a regression test to prevent fixes and functions from interfering with further testing or reducing the efficiency of test execution. If any problems are found during this check, the defects are reviewed and fixes provided before the new microcode level is released for use in testing. Unlike some tests that have conditional exit criteria, the gateway test is a pass-or-fail test.

Protocol test. The protocol test is executed on open system fiber network connections between the tar-

get server environment and ESS. Its primary focus is to ensure correct handling according to the American National Standards Institute (ANSI) Fibre Channel specification. The protocol test also tests for unsupported usages and some exception usages.

Attachment test. The attachment test is executed on a variety of System/390 processors in both single and multihost environments and utilizes both switched and direct ESCON and FICON connections. It is also used to verify a correct response to all sequence controls from any host interfaces, in single and multihost environments, including start subchannel, cancel subchannel, clear subchannel, test subchannel, reset channel path, and suspend and resume subchannel. Test cases include those testing the copy services features PPRC, XRC, Concurrent Copy, FlashCopy*, and Parallel Access Volume (PAV, see later subsection "Going beyond software FVT" [functional verification test]). Attachment test is responsible for checking for a correct response to various invalid or perverse channel programs and the output of the correct status and sense data, where applicable. Finally, this test confirms that ESS can successfully operate at the specified maximum distances from the ESCON and FICON switch (and/or the processor, in the case of the direct channel connection).

Input/output test. The input/output test is responsible for storage area network (SAN) coverage. It is divided into the following areas.

Good path verifies that all components in the SAN interoperate while performing nonfailure operations without adversely affecting each other.

Fabric error recovery verifies that hosts and their connected storage recover as expected from temporary and permanent faults of SAN components and that the appropriate recovery actions do not adversely affect other concurrent operations in the SAN that are not involved with the recovery.

Storage device error recovery verifies that hosts and their connected storage recover as expected from temporary and permanent failures of the attached storage devices, and that the appropriate recovery actions do not adversely affect other concurrent operations in the SAN that are not involved with the recovery.

Storage device setup and service verifies that the connected storage devices can be successfully set up and serviced without disrupting other concurrent operations in the SAN that are not involved with the setup or service operations.

In addition, a Microsoft Cluster Services (MSCS) area is included to verify the Windows NT** and Windows** 2000 cluster operations in a SAN attachment.

Function-based test. The function-based test deals with specific functions one at a time and in isolation from other functions. The emphasis is not on customer tasks but rather on validating input and output and boundary conditions, with limited failure handling.

Error handling behavior test. The error handling behavior (EHB) test verifies that when a fault[2] is applied, the machine correctly responds to the fault condition. This may involve the machine repairing itself, properly isolating and reporting the problem for service action, isolating the failing hardware (fencing or quiescing the system resource), and/or routing the work through other resources to minimize the problem impact.

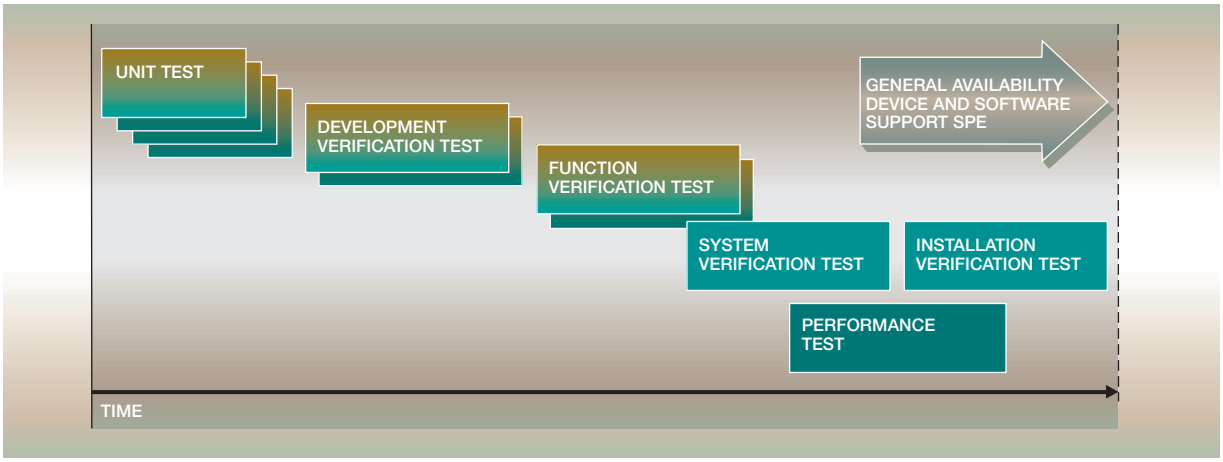
The types of EHB test are: device, channel, control unit, and IML (initial machine load).

System-level serviceability test. The system-level serviceability (SLS) test involves testing all of the human interface tools provided for service personnel as well as customer-provided tools. Testing these tools in their proper environment requires performing concurrent maintenance within the defined boundaries, configuring machines from both the service perspective as well as the customer perspective, and verifying the invocation, manipulation, and termination of various copy services operations via the appropriately provided tools. SLS test also provides usability-level testing of all interfaces that customer and support personnel use to interface with ESS.

An example of SLS testing involves the following steps:

- Configure ESS with a mix of System/390 and open system devices.
- Configure and activate copy services operations: PPRC, FlashCopy, PAV.
- Perform read/write operations from System/390 and open systems to the defined devices.
- Perform concurrent service operations on different boundary areas (cluster, power supplies, etc.) and check for any impact to the attached systems.

Figure 1 DFSMS device support SPE test phases



- Perform concurrent ESS code load operation and check for any impact to the attached systems.
- Perform all possible ESS array configurations and verify operation.
- Check all customer-capable operations and evaluate for usability and accuracy of documentation.

System-level test. The system-level test (SLT) includes open SLT (combinations of open system hosts and network switches), System/390 SLT (heavy System/390 workloads), and heterogeneous SLT (a mixture of open systems and System/390). SLT provides a focus on product stability at a level that is equal to known customer environments or an intermixture of such environments.

ESS software testing

ESS software testing involves verifying the software functions written to support the ESS logical control units and DASD (direct access storage devices). The code needed to support these devices is referred to as a Small Programming Enhancement (SPE). An SPE represents a programming development effort of between 50 and 10000 lines of code.

First, we describe the ESS DFSMS (Data Facility Storage Management Subsystem)[3] testing in general. Then, in the next section, we focus on our experience with DFSMS device FVT testing for ESS. (In our discussion, "FVT" refers to DFSMS device FVT.)

DFSMS testing. DFSMS is an element of the z/OS* operating system. DFSMS, along with z/OS BCP (base control program), is responsible for data, storage, and device management of the mainframe operating system.[4]

Because the entire current DFSMS customer base is the target group for the use of any new storage device, an SPE normally needs to be written for all releases of DFSMS currently supported. This may cover three or four supported versions or releases of DFSMS, along with any additional version or release currently in the development cycle that is to be announced at a future date.

DFSMS testing is performed by software engineers working in various areas of product test and development. Descriptions of the six phases of DFSMS testing for an SPE supporting a new device (as shown in sequence in Figure 1) follow.

Unit test. The programmers who develop the code (developers) are responsible for doing modular testing for the code changes they have made to individual modules. The purpose is to verify that every new line of code performs correctly when executed. To complete unit test for device support code requires that the new device is physically available, or that code is written to simulate the new device. Much of the device error simulation testing is also covered during this phase. DFSMS device support code normally has dependencies on z/OS base code changes, as well as other DFSMS component changes, in order to validate its function.

Development verification test. Developers perform this test as well, with the assistance of function testers (people responsible for verifying that the code functions work as documented). Development verification test (DVT) ensures that each function works as stated in the design and that device interactions work correctly. It also validates the stability of the newly integrated modules. Before DVT can begin, all of the new and changed modules need to have been unit tested and informally integrated into a driver. For a device DVT, a hardware "acceptance criteria" list is included in the DVT plan specifying the major new hardware functions of the device, which are necessary to start the software testing. The new functions listed as hardware acceptance criteria are selected with hardware planning and development personnel to support delivery of the functions that need to be working in hardware as a prerequisite for software function verification.

Function verification test. Function testers perform this test to verify that new product externals execute as documented in the programming functional specification (PFS) for the project. Product externals include functions, error handling, and product behavior. DFSMS device FVT is divided into separate FVT groups by products and devices (tape and DASD). One reason for dividing the FVT is the complexity of DFSMS and the devices supported. Another reason is to stage the verification of the functions tested. The first FVT focuses on validating the basic interaction of the device with the operating system (i.e., varying the device on line and off line) and confirming that the access methods work correctly. This FVT stage then provides the entrance criteria for the next stage, which focuses on copy services or products like DFSORT (Data Facility Sort). FVT is executed only on formally integrated code. Each FVT group writes a separate FVT plan documenting the device and software functions to be tested.

System verification test. The team performing system test (system testers) assesses the quality of the software products from our customers' perspective. The system verification test (SVT) typically follows, or sometimes overlaps, the FVT. There is one SVT plan per DFSMS device test project, covering all the functions verified in the separate FVTs. SVT is executed in a sysplex environment (i.e., with MVS* [Multiple Virtual Storage] systems working together) in order to test the interaction of many systems using the device simultaneously.

Performance evaluation. The performance evaluation team provides performance analysis of software products and I/O subsystems marketed by the IBM Systems Group. The performance team requires dedicated processors and devices to perform their measurements for DFSMS. The output from performance evaluation is the release of performance information internally and externally to our customers. This performance information is targeted to be completed and made available prior to the GA of the device and its supporting DFSMS code.

Installation verification test. The installation verification test (IVT) performed by the integration team verifies that the packaging and installability of the product deliverables meet the requirements defined for the project. In addition, they provide recommendations to the project manager on packaging and installability for external shipment to the customers.

DFSMS device FVT on ESS: Prototype to GA

In this section, we elaborate on the implementation of DFSMS device FVT for ESS, which presents a unique set of challenges and expands the boundaries of traditional FVT coverage. The testing environment, scope, and the stages of the FVT, as well as the FVT testing types, are detailed.

Testing environment. The customer-like environment we strove to achieve required the use of three systems on a large IBM processor. In 1997, when FVT for ESS was started, our testing environment was on a G5 processor. Two systems executed test cases and the third system served as the automation focal point. The two testing systems were each configured with the releases of DFSMS and OS/390* or z/OS that we were testing. The releases targeted for FVT were documented in the FVT test plan. We would like to provide coverage for all the releases for which the new code is written, but in reality we can usually only provide FVT test coverage for about four releases. Based on input from development, we may take a risk on an untested release in FVT and provide coverage for it in one of our other testing venues—SVT, performance test, or an FVT performed by another group.

The scope of DFSMS device FVT. DFSMS device FVT provides function test coverage for all DFSMS software components that interact with devices and are affected by new or upgraded devices. Devices supported are those accessed by an S/390* processor un-

der the control of OS/390 or z/OS, such as DASD, tape drives, control units and automated tape libraries.

Using an MVS-based set of IBM software products for automation gives our testers a better sense of how our customers use MVS, and it increases our skills with the products we test.

Additionally, the DFSMS device support group has the development and test responsibility for IBM device support facilities and EREP (Environmental Record Editing and Printing program). Many other DFSMS components interface with our components, thus expanding the need for testing beyond our department's component set. We provide test coverage for all access methods and utilities, as well as some BCP functions like system initialization.

DFSMS device FVT testing stages and types. The major activities performed during DFSMS device FVT for ESS can be divided into the following four stages: planning, preparation, execution, and continual improvement.

FVT planning begins when the request for a test sizing is received. The function test sizing details the number of test engineers needed and the time line for test execution. The function test sizing is created based on input from the software developers, postmortem documentation from prior tests, and past experience with prior device function tests.

The preparation stage begins when the first pass of the test plan is approved. The FVT testers build the executable test cases from the matrix tables documented in the test plan. The matrix tables show the relationship of the component functions being tested to the names of the test cases that will execute the test. Any hardware that needs to be installed and configured is also acquired during the preparation stage.

The execution stage includes running the test cases, validating and saving the output, and documenting the results. This stage is often the most volatile when determining time estimates.

A continual improvement (postmortem) meeting is held after all testing and final test plan reviews are completed. Information gathered by the team leader and presented in the postmortem review includes PTM (program trouble memorandum) information and the total time spent by each function tester on test activities. The information presented and discussed during the postmortem review includes positive and negative aspects of the test, emphasizing what worked well and what did not. Areas discussed are test strategy and execution, fix availability and verification, PTM turnaround time, development support, test support, driver support, integration support, hardware and microcode support, and communication. Suggestions for the next test are noted.

FVT of the software for device support is comprised of four main testing types: regression testing, progression testing, new function testing, and coexistence testing.

Regression testing is done to assure that the software written to support the new device does not negatively affect the functions of currently supported devices. For many of our customers, regression testing will have the greatest impact on the product quality they experience.

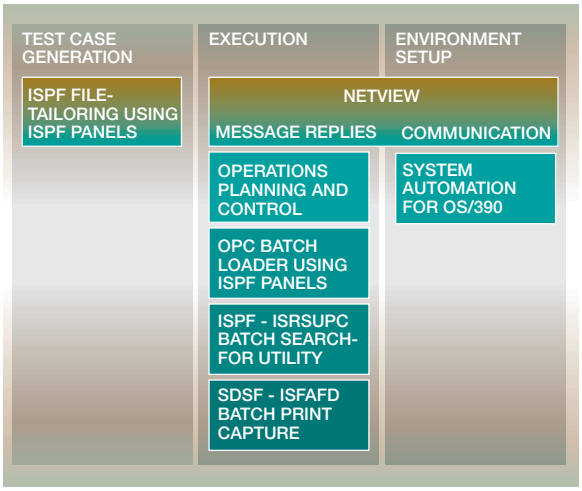
Progression testing verifies that the existing suite of test cases that work on the current devices in the field today continues to work when executed on the new device.

New function testing requires writing new or updated test cases to exercise the software that supports the new functions exploiting the new hardware capabilities.

Coexistence testing is performed to ensure that adding the ESS to an environment running multiple versions of DFSMS software does not impact the execution of older releases of DFSMS software not supporting the full functions of the ESS.

Automation strategy. After experiencing many disappointments using internally developed and supported automation tools, DFSMS device test adopted an automation strategy in 1991 based on IBM software products. This strategy had several benefits to our test team, our customers, and the support teams of the software products we chose. IBM software products provide us the stability we did not find in using internal tools. IBM software products are maintained, upgraded, and well-documented, and edu-

Figure 2 DFSMS device support automation areas



cation is available. When IBM software products reach their end of life, replacements are usually available. This is a very attractive benefit, since we have seen support for many internal tools dropped when the tool owner moved to a new job or left the company. Using an MVS-based set of IBM software products for automation gives our testers a better sense of how our customers use MVS, and it increases our skills with the products we test. On at least two occasions, we discovered incompatibilities between the IBM software products we use and new OS/390 or z/OS functions. In many instances, we were the first group in IBM to run OPC (Operations Planning and Control) or NetView* on a development level of z/OS that was not to be made generally available for a year or more. Our experience with this integration of IBM software products into our test processes has always been positive, and the product support groups have always been very quick in resolving problems.

The deployed automation utilized selected IBM software products that were targeted at the key areas of test case generation, execution, and environment setup (see Figure 2). The IBM software product set used in DFSMS testing for ESS and given below is also used by many of our customers.

- Test case generation—ISPF (Interactive System Productivity Facility) File-Tailoring services
- Test case submission, execution, validation and tracking—Operations Planning and Control
- Test case validation beyond condition code checking—ISPF Search-For Utility
- Test case system command print capture from the console—SDSF (System Display and Search Facility) executed by the batch program ISFAFD
- Test case message replies, console monitoring, and user notification—Tivoli NetView for OS/390
- Execution environment automation—System Automation for OS/390

We now describe this product set in more detail.

ISPF File-Tailoring services are used along with ISPF panels, CLISTS (command lists) and TSO/E REXX programs, which are written to easily generate the test cases. Using the panels makes passing the variables to the TSO/E REXX programs easier, and it provides the ability to maintain the library structure and naming conventions for our executable test-case libraries. (TSO/E REXX stands for Time Sharing Option Extensions Restructured Extended Executor Language.) In some instances, 20 or more executable test cases can be generated from one base test case. Each of these executable test cases exercises a uniquely targeted device function. Variable parameters are used for device and control unit characteristics in all our test cases, allowing groups of test cases to be generated for any DASD device and control unit pair in our test lab.[5]

Operations Planning and Control (now known as Tivoli Workload Scheduler [TWS] for z/OS) is the heart of the execution phase for our testing. OPC provides an excellent facility to submit and monitor successful batch job execution, and is used by some of IBM's largest customers. Extensive use of the OPC batch loader allows batch jobs to submit groups of test cases for execution. ISPF panels, along with file-tailoring, are also incorporated into creating executable batch loader jobs from base batch loader jobs. An OPC-supplied system output archiving program running in JES2 is customized to save test case output to a sequential file. Upon successful execution of a group of test cases, the tester executes a TSO/E REXX program to migrate the sequential files into a partitioned data set that serves to archive the test case execution.[6]

The *ISPF Search-For Utility* provides the power to go beyond checking the condition code of each job step. The Search-For SuperC batch program, ISRSUPC, is heavily used in our test cases to verify the existence of lines of data that are expected to be generated when the job executes successfully.[7]

The *SDSF batch program ISFAFD* print capture capability, in combination with the WRITELOG command, is used to validate system commands written to the MVS console. This batch program, combined with the ISPF Search-For Utility, allows us to verify hundreds of console messages without human intervention because checking for strings of information appearing in the captured output is automated.[8]

Tivoli NetView for OS/390 is the primary communication vehicle between systems. It provides the base of System Automation for OS/390. The console monitoring feature is used to automate replies to operator messages. Console message trapping and e-mail notification to testers is used to monitor unusual device situations.[9]

System Automation for OS/390 runs as a NetView application. System Automation is the manager for the MVS subsystems required to execute our device testing activities. System Automation sets up the execution environment by managing the startup and shutdown for all three of our systems. It also provides automated recovery and system maintenance tasks that would otherwise require a human operator.[10]

Going beyond software FVT for ESS. Adding new hardware to a software FVT requires a cross-disciplinary team effort. A small group of key engineers from software development and FVT meet weekly with hardware and microcode engineers to ensure that the progressive targets of the FVT efforts are being met. FVT team members work closely with the hardware group to establish the initial testing environment.

When the first ESS was received by DFSMS device FVT personnel, configuration was done using UNIX** scripts written in low-level ESS configuration commands. The Web-based ESS Specialist software was not available that early in the product life cycle. Changes to a configuration were extremely time consuming, so getting it right the first time was important.

The first ESS delivered to FVT in late 1997 was an engineering prototype, configured as two independent clusters. One benefit of the original design was that it allowed us to run different microcode levels on each cluster. This provided the ability to verify a new release or fix on one cluster before moving all of our testing efforts to the new code. In effect, this gave us two separate ESS subsystems.

The main focus for the DFSMS FVT was to verify the new PAV capability of ESS.[11] Parallel access volumes allow a single system to access a DASD volume with multiple concurrent requests. This is accomplished by assigning base and alias unit addresses to a volume. Base and alias unit address relationships create a single logical volume, allowing concurrent I/O operations to the volume. Before we could start testing PAV, we had to execute and verify many other components' support for this new feature. There were many interdependencies on the new functions between various components in ESS hardware and z/OS software.

DFSMS device FVT goes beyond the traditional FVT philosophy of testing only the external functions of the code. There were many instances in ESS FVT where we were asked to verify that new or changed channel control words were being correctly executed by ESS. This verification was done using the generalized trace facility (GTF), an MVS service aid. After capturing the GTF trace, a batch IPCS (Interactive Problem Control System) job was used to format it, making analysis of the trace data easier and allowing for the archiving of the successful GTF trace execution.

Having a two-system test environment allowed us to go beyond the traditional FVT methods of executing function in a single thread. We configured many initiators on our systems and, using OPC applications, we submitted several different test scenarios simultaneously. Providing this multithreaded test case execution for stress testing in FVT, we could uncover problems early in the testing cycle. This was most beneficial to the SVT and performance testing that followed. Our environment also allowed us to test shared device interaction across two systems.

Following DFSMS device FVT for ESS, other FVT groups (for example, copy services) verified their components' code. It is common for follow-on FVT and SVT to overlap, based on agreed upon entrance criteria documented in test plans. Our department works closely with the other testing departments to prioritize our testing based on the functions needed to enter their tests.

ESS software testing conclusion

The process and methodologies followed by DFSMS device FVT can be adapted to most function verifi-

cation testing. Team building, test automation, and adherence to established process and procedures are the basis of many software FVT efforts. Schedule coordination with dependent areas is crucial to FVT tracking, as is the need to verify low-level software functions beyond traditional FVT coverage.

Summary

Testing plays a major role in the introduction of new devices by IBM. The ESS hardware development verification and software testing outlined here reflects the breadth of testing coverage for a new storage device. The test planning, preparation, and execution, as well as defect tracking and in-process metrics described here, are applicable to many systems.

The testing methodology of IBM is based on controlled processes and procedures and has evolved to a high degree. Our focus on the details of the DFSMS device FVT for ESS is intended to provide a view into the complexity of a combined hardware and software testing effort.

Acknowledgments

The authors wish to acknowledge the contributions of Paik Saber, Jay Hayes, Sofia Laskowski, Gerardo Frias, Michael Houghtaling, Gale Burt, Melanie Fallis, Jim Cammarata, and Barbara White McDonald.

Cited references and notes

1. See M. Butcher, H. Munro, and T. Kratschmer, "Improving Software Testing via ODC: Three Case Studies," *IBM Systems Journal* **41**, No. 1, 31–44 (2002).
2. The term "fault" here is used in the context of the IEEE *Standard Dictionary of Measures to Produce Reliable Software*, Institute of Electrical and Electronic Engineers, IEEE Standard 982.1 (1988).
3. In this technical note DFSMS represents the DFSMSdfp™ (Data Facilities Product) base element of z/OS.
4. *z/OS DFSMS Introduction*, SC26-7397-01, IBM Corporation (March 2002).
5. *z/OS V1R2.0 Interactive System Productivity Facility (ISPF) Dialog Developer's Guide and Reference*, SC34-4821-01, IBM Corporation (October 2001).
6. *Tivoli Workload Scheduler General Information V8.1*, GH19-4539-00, Chapter 3, IBM Corporation (2001).
7. *z/OS V1R2.0 Interactive System Productivity Facility (ISPF) User's Guide Volume II*, SC34-4823-01, IBM Corporation (October 2001).
8. *z/OS V1R4.0 SDSF Operation and Customization*, SA22-7670-04, IBM Corporation (December 2002).
9. *Tivoli NetView for OS/390 Automation Guide V1R4*, SC31-8225-03, IBM Corporation (2001).
10. *System Automation for OS/390 V2R1*, GC33-7036, IBM Corporation (October 2000).
11. *IBM TotalStorage Enterprise Storage Server: Implementing the ESS in Your Environment*, SC24-5420-01, IBM Corporation (March 2002).

Accepted for publication January 16, 2003.

Michael R. Dillon *IBM Systems Group, 5600 Cottle Road, San Jose, California 95193 (mrdillon@us.ibm.com).* Mr. Dillon is a Senior Technical Staff Member in the software test area of System/390. He has a B.A. degree in mathematics from San Jose State University and is a graduate of the IBM Systems Research Institute. Additionally, he has a certificate in business administration with distinction from The Southampton University Management School (UK). He has been an IBM employee since 1968 with experience in software development, performance analysis, and testing. He specializes in System/390 Parallel Sysplex® data sharing software testing.

Paula M. Collins *IBM Systems Group, 9000 S. Rita Road, Tucson, Arizona 85744 (dbcollin@us.ibm.com).* Ms. Collins is a senior engineer in the development test organization supporting Enterprise Storage Systems. She graduated with a B.S.E.E. degree from California State University at Chico in 1982 and joined IBM at the San Jose main plant in June 1982. She has worked in multiple areas pertaining to reliability of IBM storage products. She received an Outstanding Achievement Award in 1986 for significant contributions to IBM 3380 D/E reliability, another in 1997 for RAMAC® 3 Array Storage, and an Outstanding Technical Achievement Award in 2000 for ESS SLS test. She currently enjoys testing new storage products during the development cycle.

Paul R. Hurley *IBM Systems Group, 9000 S. Rita Road, Tucson, Arizona 85744 (hurleypr@us.ibm.com).* Mr. Hurley is an advisory software engineer and is the system level serviceability (SLS) technical team leader in the Product Testing Laboratory. He joined IBM in 1978 and has worked on high-end DASD since that time. He has a degree in mathematics and has developed several software tools used in conjunction with testing DASD.

Jean E. Handlin *IBM Systems Group, 6151 Lakeside Drive, Suite 1100, Reno, Nevada 89509 (handlin@us.ibm.com).* Mrs. Handlin is an advisory software engineer in the System/390 Device Support Software department. She has been an IBM employee since 1979, working as a process technician before joining the System/390 Device Support Software department in 1985 as a DFSMS device function tester. Her areas of interest include function test leadership and test automation.

Sheryl A. Reed *IBM Systems Group, 5600 Cottle Road, San Jose, California 95193 (sherylr@us.ibm.com).* Mrs. Reed is an advisory software engineer in the System/390 Device Support Software department. She received a B.S. degree in computer science from California State University at Chico in 1980. She has been an IBM employee since 1989, working as a system programmer and DFSMS device function tester. Her areas of interest include z/OS-based test automation, system/hardware configuration, and project management. Before joining IBM, she was a system programmer for Litton Applied Technology and NEC Electronics.