Improving Process to Increase Productivity While Assuring Quality: A Case Study of the Volume Shadowing Port to OpenVMS AXP

by

William L. Goleman, Robert G. Thomson, Paul J. Houlihan

ABSTRACT

The volume shadowing team achieved a high-quality, accelerated delivery of volume shadowing on OpenVMS AXP by applying techniques from academic and industry literature to Digital's commercial setting. These techniques were an assessment of the team process to identify deficiencies, formal inspections to detect most porting defects before testing, and principles of experimental design in the testing to efficiently isolate defects and assure quality. This paper describes how a small team can adopt new practices and improve product quality independent of the larger organization and demonstrates how this led to a more enjoyable, productive, and predictable work environment.

INTRODUCTION

To achieve VMScluster support in the OpenVMS AXP version 1.5 operating system one year ahead of the original plan, OpenVMS Engineering had to forego early support of Volume Shadowing Phase II (or "shadowing"). Shadowing is an OpenVMS system-integrated product that transparently replicates data on one or more disk storage devices. A shadow set is composed of all the disks that are shadowing (or mirroring) a given set of data. Each disk in a shadow set is referred to as a shadow set member. Should a failure occur in the software, hardware, firmware, or storage media associated with one member of a shadow set, shadowing can access the data from another member.

The ability to survive storage failures is quite important to customers of OpenVMS systems where data loss or inaccessibility is extremely costly. Such customers typically combine shadowing and VMScluster technologies to eliminate single points of failure and thereby increase data availability. For these customers, delayed support for shadowing on the OpenVMS AXP system meant either foregoing the advanced capabilities of an Alpha AXP processor within their VMScluster systems or foregoing the additional data availability that shadowing provides. To resolve this dilemma, OpenVMS Engineering began a separate project to rapidly port shadowing to the OpenVMS AXP system. This project had three overall goals.

- Provide performance and functionality equivalent to the OpenVMS VAX system

- Allow trouble-free interoperability across a mixed-architecture VMScluster system

- Deliver to customers at the earliest possible date

All three goals were met with the separate release of shadowing based on OpenVMS AXP version 1.5 in November 1993, more than six months ahead of the original planned release for this support.

In the following sections, we describe how we achieved these goals by reshaping our overall process, reworking our development framework, and redirecting our testing. In the final section on project results, we demonstrate how our improved process assures quality and increases productivity. This paper assumes familiarity with the shadowing product and terminology, which are described fully in other publications.[1,2]

RESHAPING THE OVERALL PROCESS

Because the need was urgent and the project well-defined, we could have leapt directly into porting the shadowing code. Instead, we took a step back to evaluate how best to deliver the required functionality in the shortest time and how best to verify success. Doing so meant taking control of our software development process.

Effective software process is generally acknowledged as essential to delivering quality software products. The Capability Maturity Model (CMM) developed by the Software Engineering Institute embodies this viewpoint and suggests that evolving an entire organization's process takes time.[3,4] Grady and Caswell's experience implementing a metrics program at Hewlett-Packard bears out this viewpoint.[5] Our experience with the continuous improvement of software development practices within Digital's OpenVMS Engineering does so as well.

However, our engineering experience also suggests that the current emphasis on evolving an entire organization's process tends to overshadow the ability of a small group to accelerate the adoption of better engineering practices. Within the context of an individual software project, we believed that process could be readily reshaped and enhanced in response to specific project challenges. We further believed that such enhancements could significantly improve project productivity and predictability.

Identifying Process Challenges

At the project's outset, we identified four major challenges that we believed the project faced: configuration complexity, defect isolation costs, beta test ineffectiveness, and resource

constraints.

Configuration Complexity. Our most significant challenge was to devise a process to efficiently validate the product's complex operating environment: a mixed-architecture VMScluster system comprising both Alpha AXP and VAX processors (or nodes).[6] Digital's VMScluster technology currently supports a configuration of loosely coupled, distributed systems comprising as many as 96 AXP and VAX processors. These nodes may communicate over any combination of four different system interconnects: Computer Interconnect (CI), Digital Storage Systems Interconnect (DSSI), fiber distributed data interface (FDDI), and Ethernet. VMScluster systems support two disk storage architectures -- the Digital Storage Architecture (DSA) and the small computer systems interface (SCSI) -- and dozens of disk models. Once ported, shadowing would be required to provide a consistent view across all nodes of as many as 130 shadow sets. Each shadow set may involve a different model of disk and may span different controllers, interconnects, nodes, or processor architectures. The potential number of configuration variations is exponential.

Defect Isolation Costs. A second major process challenge was to contain the cost of isolating defects. A defect is defined to be the underlying flaw in the OpenVMS software that prevents a VMScluster system from meeting customer needs. System software defects can be triggered by VMScluster hardware, firmware, and software. Since few individuals possess the combined skills necessary to troubleshoot all three areas, defect isolation normally involves a team of professionals, which adds to the cost of troubleshooting VMScluster operating system software.

Debugging of shadowing code is difficult since it executes in the restricted OpenVMS driver environment: in kernel mode at elevated interrupt priority level. Shadowing is also written mostly in assembly language. To maintain shadow set consistency across all 96 nodes of a VMScluster system, much of the shadowing code involves distributed algorithms. Troubleshooting distributed algorithms can greatly increase isolation costs, since a given node failure is often only incidental to a hardware, firmware, or software defect occurring earlier on another VMScluster node.

Many shadowing problem reports ultimately prove to contain insufficient data for isolating the problem. Other problem reports describe user errors or hardware problems; some are duplicates. For example, Figure 1 shows the trend for Volume Shadowing Phase II problems reported, problems resolved, and defects removed between December 1992 and April 1993. During this period, only one defect was fixed for every ten problem reports closed. Because this low ratio is not typical of most OpenVMS subsystems, it is not readily accommodated by our traditional development process.

[Figure 1 (Problem Handling and Defect Removal on VAX: December 1992 to April 1993) is not available in ASCII format.]

Beta Test Ineffectiveness. A third process challenge was that customer beta testing had not contributed significantly to shadowing defect detection. Justifiably, most customers simply cannot risk incorporating beta test software into the kind of complex production systems that are most likely to uncover shadowing problems. Figure 2 shows the distribution of shadowing problem reports received from its inception in January 1990 to January 1993. During these three years, only 8 percent of the problem reports came from customer beta test sites. In contrast, 46 percent of the problem reports came from stress test and alpha test sites within Digital, where testing was based on large, complex VMScluster configurations.

[Figure 2 (Sources of Shadowing Problem Reports: January 1990 through January 1993) is not available in ASCII format.]

Resource Constraints. A fourth process challenge for the shadowing port was competition for engineering resources. Only the development and validation project leaders could be assigned full-time. The ongoing demands of supporting shadowing on OpenVMS VAX precluded members of the existing shadowing team from participating in the port. Most other engineering resources were already committed to the accelerated delivery of VMScluster support in OpenVMS AXP version 1.5. As a consequence, the majority of the shadowing team comprised experienced OpenVMS engineers whose familiarity with shadowing was limited, whose individual skill sets were often incomplete for this particular project, and whose availability was staggered over the course of the project. Moreover, the team was split between the United States and Scotland and, hence, separated by a six-hour time difference.

Making Process Enhancements

To meet these challenges, we believed our overall process required enhancements that would provide

- Independent porting tasks within a collaborative and unifying development framework
- Aggressive defect removal with an emphasis on containing porting defects
- Directed system testing that preceded large-scale stress testing
- Clear validation of shadowing's basic error-handling capabilities

Figure 3 shows our reshaped process for the shadowing port. Each step in the process is depicted in a square box starting with planning and ending with the project completion review. New steps

in the process are shaded gray. The most significant enhancements were the insertion of inspection and profile testing steps. To evaluate our progress in removing defects, we incorporated defect projections for each development step into our release criteria. To track this progress, we supplemented the organization's problem-reporting database with a project-defect database. Emphasizing error insertion during profile and acceptance test allowed for validation of shadowing's error-handling capabilities.

In making these process enhancements, we were careful to maintain both consistency with prevailing industry practices and compatibility with current practices within OpenVMS Engineering. We felt that adopting ideas proven in industry and having a common framework for communication within our organization would increase the probability of success for our enhancements. How we implemented these enhancements is described in the following sections.

[Figure 3 (Enhanced Development and Validation Process) is not available in ASCII format.]

Measuring Process Effectiveness

Establishing Release Criteria. In formulating the release criteria for shadowing given in Table 1, we used Perry's approach of

- Establishing the quality factors that are important to the product's success
- Mapping the factors onto a set of corresponding attributes that the software must exhibit
- Identifying metrics and threshold values for determining when these software attributes are present[7]

Defining release criteria based on threshold values provided a clear standard for judging release readiness independent of the project schedule. These criteria spanned the development cycle in order to provide a basis for verifying progress at each stage of the project. The emphasis of most metrics for these criteria was on containing and removing defects. Other metrics were selected to corroborate that high defect detection equated to high product quality.

Table 1 Shadowing Release Criteria

Quality Factor	Software Attribute	Software Metric	Threshold Value
Reliability	Error tolerance	Profile test completion	100%
	Operational	Defects detected	240

	accuracy	Incoming problem reports per week	Near 0
	Operational consistency	Acceptance test with error insertion	200 node-hours
Integrity	Data security	Unresolved high-severity defects	None
Correctness	Completeness	Code ported	100%
		Module test completion	100%
		Code change rate per week	0
Efficiency	Processing time	Queue I/O reads and writes	Comparable
	Throughput	Copy and merge operations	to VAX
Usability	Ease of training	Documentation completion	100%
Interoperability	Backward compatibility	Stress test completion	12,000 node-hours
	Transparent recovery	Unresolved high-severity defects	None
Maintainability	Self-descriptiveness	Source modules restructured	100%
	Consistency		

Tracking Defects. Projecting defect detection levels for all stages in the development cycle was a departure from the traditional practice of our development engineers. Previously, only the test engineers within OpenVMS Engineering established a defect goal prior to beginning their project work. Extending the scope of this goal for shadowing resulted in a paradigm shift that permeated the entire team's thinking and encouraged each member to aggressively look for defects. Since all team members were more committed to meeting or exceeding the defect goal, they were eager to provide detailed information on the circumstances surrounding defect detection and removal. This detail is often lost in a traditional development project.

Because data on code modification and defect removal during an OpenVMS port was not readily available, we derived our projections as follows.

1. We determined how many lines of code would be modified during the shadowing port. This estimate was based on a comparison between the ported and original sources for another OpenVMS component of similar complexity. The resulting estimate for shadowing changes was 2,500 noncomment source statements (NCSS) out of a total of roughly 19,400.

2. We projected the rate at which these modifications would occur for shadowing. We based this projection on the actual time spent porting, inspecting, and debugging the code of a second OpenVMS component of similar complexity. We revised it upon reaching the first major porting milestone to reflect our actual performance. The revised projection is shown by month in Figure 4.

3. Using the results from our first milestone, we estimated that 250 defects would be introduced as a result of the complete port. This estimate included not only defects introduced through code modifications but also defects induced in existing code by these modifications.

4. We projected the schedule of defect detection for each stage of the development cycle. This projection assumed that defects were distributed uniformly throughout the code. Based again on the results of our first porting milestone, we estimated that our efficiency at removing the defects in this release (or defect yield) would be roughly 60 percent through inspections and 25 percent through module testing. We assumed that an additional 10 percent of the defects would be removed during profile and stress testing. A 95 percent overall yield for the release is consistent with the historic data shown in Figure 2. It is also consistent with the highest levels of defect removal efficiency observed in the industry where formal code inspections, quality assurance, and formal testing are practiced.[8] Figure 5 shows our projections for removing 240 defects (95 percent yield of our estimate of 250 defects) by both month and method.

[Figure 4 (Projections of Code Changed by Month) is not available in ASCII format.]

[Figure 5 (Projected Defect Detection by Month and Method) is not available in ASCII format.]

Tracking defects was difficult within our larger organization because the problem reporting system used by OpenVMS Engineering did not distinguish between defects, problem reports, and general communication with test sites. To work around this shortcoming, we created a project database to track defects using off-the-shelf personal computer (PC) software to link defects to problem reports.

REWORKING THE DEVELOPMENT FRAMEWORK

Only the development project leader satisfied all the requirements for executing a rapid port of the shadowing code:

- Expertise in shadowing, VMScluster systems, OpenVMS drivers, the VAX assembly language, the AMACRO compiler (which compiles VAX assembly language for execution on Alpha AXP systems), and the Alpha AXP architecture[9]

- Experience porting OpenVMS code from the VAX to the Alpha AXP platform[9]

- Familiarity with the defect history of shadowing and the fixes that were being concurrently applied to the VAX shadowing code

- Availability throughout the duration of the project to work on porting tasks at the same time and the same place

To compensate for the lack of these capabilities across all team members and to improve our ability to efficiently port the code while minimizing the number of defects introduced, we reworked our development framework in two ways. First, we restructured the modules to improve their portability and maintainability. Second, we inspected all porting changes and most fixes to assure uniform quality across both the project and the code.

Restructuring Modules

Examining the interconnections between the shadowing modules in the OpenVMS VAX software revealed a high degree of interdependence based on content coupling.[10] Modules frequently branched between one another with one module using data or control information maintained in another module. These modules also exhibited low cohesion with subroutines grouped somewhat by logical ordering but primarily by convenience.[10]

Structured in this fashion, the shadowing modules were not only more difficult to maintain but also less separable into independent porting tasks. Moreover, differences between the VAX and the Alpha AXP architectures, together with the transformation of the VAX assembly language from a machine assembly language to a compiled language, precluded the continued use of content coupling in the ported code.

To remedy these structural problems, we partitioned the shadowing code into functional pieces that could be ported, inspected, and module tested separately before being reintegrated for profile and stress testing. Figure 6 shows both the original and the reworked relationships between shadowing's source modules and its functions. During restructuring, we emphasized not only greater

functional cohesion within the modules but also improved coupling based primarily on global data areas and I/O interfaces. As a consequence, most shadowing functions were directly dependent on only one other function: mounting a single member. Once the port of this function was complete, all the others could be largely ported in parallel. Where a particular module was used by more than one function, we coordinated our porting work using a scheme for marking the code to indicate portions that had not been ported or tested.

[Figure 6 (Code Restructuring by Shadowing Function) is not available in ASCII format.]

Inspecting Changes

We believed inspections would serve well as a tool for containing our porting defects. Industry literature is replete with data on the effectiveness of inspection as well as guidelines for its use.[8,11,12] Since our code was now structured for parallel porting activities, however, the project also needed a framework for

- Integrating engineers into the project
- Coordinating overlapping tasks
- Collaborating on technical problems
- Sharing technical expertise and insights
- Assuring that the engineers who would maintain shadowing understood all porting changes

We believed that group inspections could provide this framework.

Tailoring the Inspection Process. Our inspections differed from the typical processes used for new code.[12] Only the changes required to enable the VAX code to execute correctly on the Alpha AXP platform were made during the port. These changes were scattered throughout the sources, primarily at subroutine entry points. With our time and engineering resources quite constrained, we chose to inspect only these changes and not the entire code base. Because the project involved the port of an existing, stable product, no new functionality was being introduced and therefore no functional or design specifications were available. Instead, inspections were made using the VAX code sources as a reference document.

Integrating Engineering Resources. Prior experience in OpenVMS Engineering indicated that engineers working on unfamiliar software could be very productive if they worked from a detailed

specification and used inspections. Using the VAX sources as a "specification" for the shadowing port provided such a focus. Inspecting only code changes alleviated the need for team members to understand how a particular shadowing function worked in its entirety. Simply verifying that the algorithm in the ported sources was the same as that in the original VAX sources was sufficient.

Inspections of ported code preceded both module testing and the integration of the porting changes into the overall shadowing code base. This assured that any differences in coding standards or conventions were harmonized and any misunderstanding of code operation was corrected before the code underwent module testing. Inspections occurred before the engineers who performed the port returned to their original duties within OpenVMS Engineering. By participating in these inspections, the engineers who would maintain the ported code understood exactly what was changed, in case additional debug work was needed.

Sharing Experience and Expertise. The inspection process provided a forum for team members to share technical tips, folklore, background, and experience. Having such a forum enabled the entire team to leverage the diverse technical expertise of its individual members. The resulting technical synergy increased the capacity of the team to execute the porting work. It also led to rapid cross-training between team members so that everyone's technical skills increased during the course of the project. This teamwork and increased productivity led to more enjoyable work for the engineers involved.

In retrospect, the use of inspections proved the greatest single factor in enabling the project to meet its aggressive delivery schedule.

REDIRECTING THE TESTING

To detect both new and existing defects, shadowing has historically undergone limited functional testing followed by extensive stress testing. The effectiveness of this testing has been constrained, however, because it

- Provided no measurement of actual code coverage
- Lacked an automatic means for forcing the execution of error paths within shadowing
- Failed to target the scenarios in which most shadowing failures occurred

To compensate for these shortcomings and improve our ability to efficiently detect defects, we formulated profile testing: a method of risk-directed testing that would follow module testing

and precede large-scale stress testing.

Defining Profile Testing

Profile testing focuses on operating scenarios that pose the greatest risk to a software product. Engineering experience clearly indicated that the highest-risk operating scenarios for shadowing involved error handling during error recovery. Examples of such scenarios include media failure after a node failure or the unavailability of system memory while handling media failure. Problems with such error handling have typically occurred only in large and complex VMScluster systems. Test profiles are simple, clearly defined loads and configurations designed to simulate the complex error scenarios and large configurations traditionally needed to detect shadowing defects.

Fundamentally, profile testing is the application of the principles of experimental design to the challenge of "searching" for defects in a large test domain. Deriving profile tests begins with a careful identification of operating conditions in which the product has the greatest risk of failing. These conditions are then reduced to a set of hardware and software variables (or "factors") and a range of values for these factors. A test profile is the unique combination of factor values used in a test run.

Combining a large set of test factors and factor values can result in unmanageable complexity. For this reason, profile testing uses orthogonal arrays to select factor combinations for testing. These arrays guarantee uniform coverage of the target test domain described by the test factors. Instead of selecting tests based on an engineer's ingenuity, we used these arrays to systematically select a subset of all possible factor combinations. As a result, we uncovered nonobvious situations that customers often encounter. By relying on combinatorics rather than randomness to detect defects, the event sequences leading to a defect can be more readily reproduced.

The following sections show how we used this approach to design and implement profile testing for shadowing. They also reveal that the cost-effectiveness of our testing improved significantly as a result.

Describing the Test Domain

Both AT&T and Hewlett-Packard have used operational profiles to describe the test domain of complex software systems.[13] Such operational profiles were typically derived by monitoring the customer usage of the system and then methodically reducing this usage to a set of frequencies for the occurrence of various systems functions. These frequencies are then used to prioritize system testing. For the purposes of validating shadowing, we extended this notion of an operational profile by decomposing the

test domain into four distinct dimensions that characterize complex software systems:

- System configuration
- Software resources
- Operational sequences
- Error events

The emphasis of our test profiles was not on how the system was likely to operate, but rather on how it was likely to fail. For our project, rapid characterization of the test domain was of greater importance than precise reproduction of it.

Identifying the Test Factors

Assessment of shadowing's test domain identified the factors that characterized its high-risk operating scenarios. This assessment was based on a review by both test and development engineers of the product's functional complexity, defect history, and code structure as characterized by its cyclomatic complexity.[14] The resulting factors provided the basis for formulating our test profiles.

System Configuration. The following key factors in system configuration describe how a shadow set is formed and accessed across the range of components and interconnects that VMScluster systems support.[6]

- Number of shadow set members (MEMBCNT)
- Device MSCP serving (MSCPSERV)
- Controller sharing (SEPCTRL)
- Emulated versus local disk controller (SERVSYS)
- Alpha AXP or VAX I/O load initiator (LOADSYS)
- Location of the storage control block (SCBBEG)
- Size limits for I/O transfers (DIFMBYT)
- Controller time-out values (DIFCTMO)
- System disk shadow set (SYSDISK)
- Disk device type (DISKTYPE)

Software Resources. Although running an application in OpenVMS can involve competing for a wide range of finite system and

process resources, only two software resources initially appeared significant for targeting error handling during error recovery within the shadowing product:

- System memory used for I/O operations
- VMScluster communication resource (send credits)

Operational Sequences. Shadow set membership is controlled by the manager of an OpenVMS system.[2] The manager initially forms the shadow set and then adds or removes members as needed. Applications use these shadow sets for file creation and access. During its use, a shadow set can require copy and merge operations to maintain data consistency and correctness across its members. Profiles that target these activities involve sequences of the following key operations.

- Merge, assisted merge, copy, and assisted copy
- Member mounts and dismounts
- File creation and deletion
- Random reads and writes; repeated reads of a "hot" block on the disk

Error Events. All complex software systems must deal with error events that destabilize its operation. For the shadowing product, however, reliably handling the following set of errors represents the essence of its value to OpenVMS customers.

- Removal of a VMScluster node (NODEERR)
- Process cancellation (PROCERR)
- Controller failure (CTRLERR)
- Disk failure (DISKERR)
- Media failure (MEDIAERR)

Managing Test Complexity

When the list of key factors for targeting shadowing's high-risk operating scenarios was enumerated, the resulting test domain was unmanageably complex. If just two test values for each of 25 factors are assumed, the set of all possible combinations was 2**25 or more than 33 million test cases.

To reduce this combinatorial complexity to a manageable level, we structured profiles in two test dimensions, error event and system configuration, using orthogonal arrays.[15,16] Columns in these orthogonal arrays describe the test factors, and rows describe a balanced and orthogonal fraction of the full set of

factor combinations. Because of their balance and orthogonality across the test factors, such arrays provide a uniform coverage of the test domain.

Figure 7 is a composite table of error-event profiles created using a D-optimal array and shadow set configuration profiles created using a standard array.[17,18] These two arrays formed the base of our profile test design. Operational sequences were applied (as described below) to 18 test profiles formed by relating the error event and shadow set arrays as shown in Figure 7. These profiles were arranged into groups of three; each group included two types of disks, a shadowed system disk, and the presence of each type of disk error. The test values assigned to the factors SYSDISK and DISKTYPE as a result of this grouping are shown in the two columns positioned between the arrays in Figure 7.

[Figure 7 (Set of Composite Test Profiles) is not available in ASCII format.]

Note that physically configuring the test environment prevented us, in some instances, from using the prescribed assignment of factor values. As a consequence, only those factors whose columns are shaded in gray in Figure 7 retained their balance and orthogonality during implementation of the test design.

At this point, profile test design is complete. The use of orthogonal arrays allowed us to reduce the tests to a manageable number and at the same time have uniform coverage of all test factors.

Configuring the Test Environment

In addition to the shadow set profiles, the following practical constraints guided the configuration of a VMScluster system for conducting our profile testing.

- Minimize hardware requirements and maximize ease of test execution
- Configure profiles from the shadow set array in groups of three to expedite test execution
- Reflect both anticipated customer usage of shadowing and historic usage as characterized in existing surveys of VAXcluster sites in the United States
- Enable the formation of either one integrated or two separate VMScluster systems based on either the DSSI or the CI system interconnect
- Require no physical reconfiguration during testing
- Maintain a consistent batch/print and user authorization

environment

- Follow the configuration guidelines set forth in Digital's software product descriptions for OpenVMS, VMScluster systems, and volume shadowing for Alpha AXP and VAX systems

Constructing a test configuration that reflected all these constraints and supported the shadow set profiles in Figure 7 was quite a challenge. As a result of having clear configuration guidelines, however, we could re-create shadowing's high-risk operating scenarios using substantially less hardware than required for large-scale stress testing. Table 2 contrasts the hardware requirements of the two test approaches.

Table 2 VMScluster Configuration Size by Test Method

	Profile Testing	Large-scale Stress Testing
Systems	5 AXP 4 VAX	10 AXP 12 VAX
Interconnects	1 CI 1 Ethernet	2 CI 9 Ethernet 1 FDDI
CI Storage Controllers	1 HSC 1 HSJ	6 HSC 1 HSJ
Test Disks	31	165
Shadow Sets	9 two-member 9 three-member	23 two-member 5 three-member

The resulting test configuration for our profile testing was formally described in a configuration diagram, which appears in a simplified form in Figure 8. During our testing, each profile shown in Figure 7 was uniquely marked on this diagram to show both the disks comprising each shadow set and the nodes to load them. Taken together, Figures 7 and 8 proved quite useful in transferring the task of executing a particular test profile from one test engineer to another. In addition, development engineers found them to be invaluable tools for clarifying the fault loads and the configuration of a particular test profile.

[Figure 8 (Total System Configuration for Profile Testing) is not available in ASCII format.]

To illustrate how we used these two tools, consider the first three test profiles shown in Figure 7. These profiles, respectively, define the configuration and loading of shadow sets

DSA1, DSA2, and DSA3. Running these three profiles in parallel would be difficult to describe precisely without these two tools. As an example, profile 1 in Figure 7 indicates that DSA1 must comprise two RZ devices that are not used as system disks. At least one of these devices must be MSCP served. The two disks must share a controller and must be accessed via an AXP node. Both must have their storage control blocks located at their first logical block. As a result of their physical configuration, both must have the same size limit on I/O transfers and the same controller time-out value. Finally, this profile indicates that another AXP node in the VMScluster system must load DSA1 and that this load must involve the simulation of fatal disk errors. Devices DKA400 and DKA500 in Figure 8 satisfied these requirements; the load was to be applied from node MEBEHE.

The complete configuration for these three profiles is denoted by the gray box in Figure 8, which requires only a small subset of the total test configuration to execute

- Two AXP systems (MEBEHE and HEBEME)
- One VAX system (WEBEYU)
- One DSSI interconnect
- One SCSI and 4 DSSI controllers
- Two RZ26 disks (DKA400 and DKA500)
- Two RF72 disks (DIA301 and DIA202)
- Two RF73 disks (DIA200 and DIA201)

Executing Test Profiles

Testing Tools. Executing the profile tests involved the use of four Digital internal test tools (XQPXR, IOX, CTM, Faulty Towers) and two OpenVMS utilities (BACKUP and MONITOR). XQPXR and IOX both provided read and/or write loads to shadow sets with XQPXR utilizing the file system for its I/O. CTM provided a means of loading multiple subsystems across the VMScluster system. Faulty Towers was used to inject faults into the VMScluster System Communication Architecture (SCA) protocol tower during loading to create the error profiles shown in Figure 7. MONITOR measured the loads applied during profile testing. BACKUP was used to verify that the data on shadow set members was consistent following a test run.

Of all the test tools we used, Faulty Towers was both the most critical to our success and the most innovative in simulating large-scale VMScluster environments. Historically, large-scale stress testing of shadowing has depended largely on the occurrence of random events or manual intervention to exercise shadowing error paths. Because SCA underlies all communication

within a VMScluster system, Faulty Towers could instead automatically force the exercise of these paths by simulating errors within the system. The set of faults that Faulty Towers provided came from our examination of how VMScluster systems, especially large-scale systems, fail. This set included forcing esoteric states throughout the VMScluster system, simulating device errors, exhausting essential resources, breaking VMScluster communication channels, and creating excessive I/O or locking loads.

The fault loads that Faulty Towers provided were predictable and quite repeatable. When problems occurred during test execution, the precise fault loads involved could be readily reproduced to accelerate the process of diagnosing the underlying defect and verifying a proposed fix. Faulty Towers also provided a means of easily tailoring, controlling, and monitoring the automatic insertion of faults during our profile testing. The result was better coverage of error paths using a much simpler test environment.

Staging Test Implementation. To stage the introduction of profile complexity, we gradually increased the number of error events applied to successive groupings of test profiles. We began our testing with a simple base profile to which further load complexity could be progressively added. This base profile involved only three two-member shadow sets with just one of the targeted error events occurring during each run. System load was limited to reads and writes across the shadow sets. No system disks were shadowed in this base profile.

During the initial execution of the base profile, we tested resource exhaustion. With each subsequent round of testing, we systematically incorporated additional complexity: more test configurations, three-member shadow sets, shadowed system disks, complex error profiles, and system-wide loading.

Operational Sequence for Profile Test Execution. Another important aspect of the profile testing was the use of a prescribed operational sequence during profile test execution. This sequence is shown in Figure 9.

[Figure 9 (Operational Sequence for Profile Testing) is not available in ASCII format.]

Profile test runs began with the mounting of a single shadow set member. The addition of a second or third member caused the initiation of a copy operation from the existing member to the added device(s). The removal of a VMScluster node that had a shadow set mounted would cause shadowing to initiate a merge operation on the shadow set. To maintain consistency across our test runs, we would manually add back into a shadow set any member(s) that were expelled due to the node removal. At this

point, shadowing is expected to progress sequentially through copy and merge operations to create a fully consistent shadow set.

The I/O required by these copy and merge operations formed the base load on the system. User I/O to the shadow sets incremented the effective load on the system as did the disruption of I/O due to error events. During the period when user I/O and error events were sustained at their heaviest levels, I/O for copy operations could stall entirely. Winding down error insertion and user I/O enabled copy and merge operations to complete. At that time, the shadow sets could be dismounted and the individual members compared for data consistency.

During the test execution sequence, each step represented a new threshold at which failures were more likely to occur. As testing continued and the shadowing code stabilized, early test execution sequences tended to generate fewer new defects. To find additional defects, we increased the complexity of the test execution sequence, fault loads, and configurations. The result was a sustained effectiveness in defect detection throughout our profile testing.

PROJECT RESULTS

The process described above enabled us to satisfy both the quality and schedule goals of our project to port shadowing to the OpenVMS AXP system. How significantly the process contributed to this accomplishment is shown below with data describing our improvements in process, product quality, and productivity.

Improving Process

Inspections and profile testing were our two key process enhancements. Data that tracked defect detection and product stabilization during these steps underscores their contribution.

Tracking Defect Detection. The solid line in Figure 10 shows defect detection by week throughout the life of the project. Note that the solid line in the figure tracks very closely and, at times, overlaps the dashed line used to indicate the inspection time. Only high severity defects that resulted in a code change are represented by these defect counts. The time period before January 4 involved neither inspections nor testing; the time period after June 14 involved only testing. During March, porting work stopped due to team members being temporarily reassigned to critical development tasks in support of OpenVMS AXP version 1.5. Allowing for that gap in March, the trend in defect detection from both inspections and testing exhibited a steady decline from mid-January through October. This trend provides a strong indication that the project was on schedule and not deferring the bulk of defect removal to the latter, more costly stages of

development.

The dashed line in Figure 10 shows the amount of time spent weekly in inspections. It shows that the highest rates of defect detection resulted from, and were in rough proportion to, time spent in inspections. Early defect removal using inspections represented a marked change from traditional practices within OpenVMS Engineering.

[Figure 10 (Inspection Time and Defect Detection) is not available in ASCII format.]

Tracking Product Stabilization. The manner in which we designed and implemented profile testing gave rise to a pair of metrics for tracking both test effectiveness and product stabilization by tracking test execution results. The first of these metrics was a ratio between the test execution time as measured in days of test execution per VMScluster node (node-days) and the number of problem reports that resulted. The second was a ratio between the number of problem reports submitted to development engineers and the number of defects that were detected as a result.

Figure 11 shows these two metrics plotted weekly during the course of our profile testing. The key to interpreting these trends lies in contrasting the two ratios. For example, low node-days/problem report accompanied by high problem-reports/defect in July 1993 indicates test execution errors as we learned how to load and test shadowing in a VMScluster system. In late September, high node-days/problem report with no defects indicates the execution of an ineffective test profile. Near-zero curves in early July and early September indicate when vacations were interrupting our testing.

[Figure 11 (Metrics for Evaluating Test Effectiveness and Product Stabilization) is not available in ASCII format.]

During the two product stabilization periods indicated in Figure 11, the increase in node-days/problem report accompanied by a near one-to-one ratio between problem reports and defects indicates that the product was stabilizing in spite of sustained test effectiveness. The first period preceded beta test; the second preceded product release.

Assuring Quality

Containing and removing defects were at the core of our release criteria. As Figure 12 shows, actual defect removal during the project exceeded projections by 8 percent. By selecting our defect goal based on industry-standard defect densities and exceeding this goal, we have assured quality in our product. Of the total defects actually removed, 232 were in the shadowing code. This suggests a defect removal rate of 12 defects per 1,000 NCSS for the shadowing code base, which is consistent with

industry data reported by Schulmeyer.[19]

[Figure 12 (Comparison of Projected and Actual Defect Detection and Removal by Method) is not available in ASCII format.]

Of the 176 defects removed through inspections, 43 were found in the nonported shadowing code. This suggests a defect removal rate for the ported code of 53 per 1,000 NCSS, which again falls within the range reported by Schulmeyer. The combination of testing and inspections resulted in the removal of 59 defects from the unmodified shadowing code base. This represents a net reduction of 3.5 defects per 1,000 NCSS within the unmodified shadowing code as a result of the port!

The significance of this level of defect removal as an indicator of high quality was corroborated in two ways. First, all release criteria were satisfied prior to releasing the ported shadowing product. Second, the character of defects detected during the testing phase of the project changed. Whereas most defects detected in early test runs were introduced during the port, virtually all defects detected in later test runs were residual in the underlying code base. Again, removing this latter type of defect meant better overall quality for customers running mixed-architecture VMScluster systems.

Another aspect of quality was identifying those defects that should not be fixed but only contained. Because this project had very limited scope, duration, and resources, we had to carefully evaluate changes that could destabilize the entire product and jeopardize its overall quality. The comparison of detected to removed defects in Figure 12 shows that several problems fell into this category. Many of these defects were triggered by system operations that exceeded the fundamental design limitations of the product. Some occurred in obscure error paths that could only be exercised with our new test methods. Others were due to new combinations of hardware possible in mixed-architecture VMScluster systems. In each of these instances, we assured that the scope of the defect was limited, its frequency low, and its impact predictable. When necessary, we constrained supported hardware configurations or system behavior so that the defect could not cause unrecoverable failures. Finally, we fed back our analyses of these defects into the ongoing shadowing support effort so that they could be removed in future releases through appropriate redesign.

Increasing Productivity

The following data shows how our enhanced process contributed to the quality. This data shows the relative cost-effectiveness, and hence improved productivity, achieved through inspections and profile testing.

Engineering Costs. The proportion of total engineering hours expended during each step of the project is depicted in Figure

13. This Figure indicates that only 15 percent of the hours (inspections and module testing) resulted in the removal of 85 percent of all defects. Inspections alone accounted for only 5 percent of the engineering hours but 68 percent of the defect removal!

[Figure 13 (Distribution of Total Engineering Hours by Process Step) is not available in ASCII format.]

The actual cost for removing defects by inspection averaged 1.7 engineer hours per defect. During module testing, when engineers worked individually to debug the functions they had ported, the cost of defect removal jumped to 15 engineer hours per defect. During integration testing, where the entire shadowing driver was tested in a complex environment, an average of 85 engineer hours was spent per defect exclusive of the time spent to execute the tests. Clearly, removing the bulk of the defects from the ported code prior to beginning testing of the integrated shadowing product dramatically reduced both the cost and time required for defect removal.

Defect Yield. Assuming a 95 percent overall defect yield for shadowing prior to release, the relative yield of inspections during this project was 65 percent. When calculated against defects found only in shadowing, the yield for inspections jumps to 75 percent--a very high removal efficiency when compared with industry data.[8] Relative defect yield was consistent with industry data for module testing at 16 percent, but low for profile and stress testing at a combined value of 6 percent. Given the high engineering cost shown above for removing defects during integration test of shadowing, this is in fact quite a favorable result.

Test Cost-effectiveness. As Figure 13 indicates, testing remained the most costly portion of the project. Executing and debugging problems from roughly 86,000 node-hours of stress, performance, and profile testing accounted for 69 percent of the project's total engineering hours. In fact, the ratio of engineer hours expended for test versus implementation was roughly 1.8 times higher than Grady reports for 48 projects involving systems software.[20] Given the complexity of the VMScluster systems historically required to test shadowing, this ratio is no surprise.

Nevertheless, our project's results indicate that profile testing was significantly more cost-effective than large-scale stress testing for detecting defects. Profile testing's overall ratio of test engineer days per defect was 25 percent better at 6.2 days than stress testing's 8.3 days. Moreover, profile testing's overall ratio of machine test time per defect was more than an order of magnitude better at 7.4 node-days than stress testing's 95.2 node-days!

This improvement in cost-effectiveness was achieved with no loss in defect removal capability. When compared with large-scale stress testing, profile testing of shadowing proved equally effective overall at detecting defects. During the beta test period for shadowing, each of these test methods accounted for roughly 20 percent of the defects detected when allowing for duplication between methods. The number of problem reports per defect was also comparable with ratios of 2.4 for profile testing and 2.0 for stress testing.

Figure 14 contrasts the cost-effectiveness of each method of defect detection employed during the validation of shadowing. Note that because this chart uses a log scale, any noticeable difference between bar heights is quite significant. This chart bears out conventional wisdom on defect removal: detection prior to integration through the use of inspections and module testing is by far the most cost-effective. It also suggests that profile testing has a cost-effectiveness on the same order of magnitude as module testing, while providing the same defect detection effectiveness as large-scale stress testing.

[Figure 14 (Relative Cost-effectiveness of Defect Detection Methods) is not available in ASCII format.]

CONCLUSIONS

At the outset of the shadowing port, given its unique challenges, we believed that the existing development process for OpenVMS would not enable us to meet the project's goals. By taking charge of our engineering process, however, we not only met those goals but also demonstrated that changes to the established process could result in higher productivity from our engineering resources and better quality in the delivered product.

The volume shadowing port from OpenVMS VAX to OpenVMS AXP was successful in meeting an aggressive schedule and in delivering a stable, high-quality product. There were two key process innovations that led to our success. The first was the use of inspections to detect and remove a large percentage of the porting defects before any investment in testing. By finding a majority of the defects (68 percent) at the lowest possible cost (1.5 hours per defect), fewer overall resources were required.

The second key innovation was the use of profile testing to cover a very large and complex test domain. Having a test strategy that used well-defined and repeatable tests to target problem areas in shadowing code allowed us to efficiently find defects and verify fixes. With profile testing, we managed to achieve the defect detection effectiveness of large-scale integration testing at the same relative cost as module testing.

The results of our process innovations would not have been realized if we had waited for our organization's process to

evolve up the CMM levels. By changing the engineering process of a small team, we delivered high-quality software on schedule and at a lower cost.

FUTURE DEVELOPMENTS

As a result of our project's accomplishments, OpenVMS Engineering is giving serious consideration to our practices. Many groups are acknowledging the gains possible when formal inspections are used to contain defects. Moreover, the organization's problem reporting system is being upgraded to include mechanisms for tracking defects that incorporate many of the fields used in our defect tracking database.

OpenVMS Engineering is also evaluating further use of the profile testing methodology in its testing efforts. As Figure 13 indicated, improving the effectiveness of our integration testing may offer the most significant opportunity for reducing engineering costs and accelerating the schedules of our software releases. Since experimental design principles were used in the creation of the tests, statistical evaluation of the results is a potentially exciting opportunity. An analysis of variance that explores the relationship between test factors and defects could indicate which test factors contribute significantly to defect detection and which do not. This could give us a clear statistical indication of where to direct our testing efforts and development resources.

ACKNOWLEDGMENTS

The success and the final form of the our software development process were a direct consequence of the diverse skills and unflagging efforts of the shadowing team members involved in its implementation: Susan Azibert, Kathryn Begley, Nick Carr, Susan Carr, Mary Ellen Connell, Tom Frederick, Paula Fitts, Kimilee Gile, Greg Jordan, Bruce Kelsey, Maria Leng, Richard Marshall, Kathy Morse, Brian Porter, Mike Stams, Barbara Upham, and Paul Weiss. Support from Howard Hayakawa, manager of the VMScluster group, provided the opportunity for us to initiate such extensive process enhancements in the context of a critical project for OpenVMS.

REFERENCES

1. S. Davis, "Design of VMS Volume Shadowing Phase II--Host-based Shadowing," Digital Technical Journal, vol. 3, no. 3 (Summer 1991): 7-15.

2. Volume Shadowing for OpenVMS (Maynard, MA: Digital Equipment Corporation, November 1993).

3. M. Paulk, B. Curtis, M. Chrissis, and C. Weber, Capability Maturity Model for Software

V1.1 (Pittsburgh, PA: Carnegie-Mellon University, Software Engineering Institute, Technical Report, CMU/SEI-93-TR-24 ESC-TR-93-177, February 1993).

4. W. Humphrey, Managing the Software Process (Reading, MA: Addison-Wesley Publishing Company, 1989/1990).

5. R. Grady and D. Caswell, Software Metrics: Establishing a Company-Wide Program (Englewood Cliffs, NJ: Prentice-Hall, 1987): 112.

6. VMScluster Systems for OpenVMS (Maynard, MA: Digital Equipment Corporation, March 1994).

7. W. Perry, Quality Assurance for Information Systems: Methods, Tools, and Techniques (Boston: QED Technical Publishing Group, 1991): 541-563.

8. C. Jones, Applied Software Measurement (New York: McGraw-Hill, Inc., 1991): 174-176, 275-280.

9. N. Kronenberg, T. Benson, W. Cardoza, R. Jagannathan, and B. Thomas, "Porting OpenVMS from VAX to Alpha AXP," Digital Technical Journal, vol. 4, no. 4 (Special Issue 1992): 111-120.

10. R. Pressman, Software Engineering: A Practitioner's Approach (New York: McGraw Hill, 1992): 334-338.

11. M. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," IBM Systems Journal, vol. 15, no. 3 (1976): 182-211.

12. D. Freedman and G. Weinberg, Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products (New York: Dorset House Publishing, 1990).

13. J. Musa, "Operation Profiles in Software Reliability Engineering," IEEE Software (New York: IEEE, March 1993): 14-32.

14. T. McCabe and C. Butler, "Design Complexity Measurement and Testing," Communications of ACM, vol. 32, no. 12 (December 1989): 1415-1425.

15. G. Taguchi and M. Phadke, "Quality Engineering Through Design Optimization," Conference Record, vol. 3 (IEEE Communications Society GLOBECOM Meeting, November 1984): 1106-1113.

16. T. Pao, M. Phadke, and C. Sherrerd, "Computer Response Time Optimization Using Orthogonal Array Experiments,"

Conference Record, vol. 2 (IEEE International Communications Conference, June 1985): 890-895.

17. S. Schmidt and R. Launsby, Understanding Industrial Designed Experiments (Colorado Springs: Air Academy Press, 1989): 3-1 to 3-32.

18. SAS/QC Software, Version 6 (Cary, NC: SAS Institute, Inc., 1992).

19. G. Schulmeyer, Zero Defect Software (New York: McGraw-Hill, Inc., 1990): 73.

20. R. Grady, Practical Software Metrics for Project Management and Process Improvement (Englewood Cliffs, NJ: Prentice-Hall, 1992): 42.

TRADEMARKS

BIOGRAPHIES

William L. Goleman Principal software engineer Bill Goleman led the project to port the Volume Shadowing Phase II product to OpenVMS AXP. He is currently involved in planning OpenVMS I/O subsystem strategies. Bill joined Digital's Software Services organization in 1981. He worked in the Ohio Valley District and then moved to VMS Engineering in 1985. Since then Bill has worked on the MSCP server, local area VAXcluster, mixed-architecture clusters, volume shadowing, and the port of cluster I/O components to OpenVMS AXP. Bill received a B.S. in computer science from Ohio State University in 1979.

Paul J. Houlihan As a principal engineer with OpenVMS AXP Engineering, Paul Houlihan participated in the port of OpenVMS software components from the VAX to the Alpha AXP architecture, including volume shadowing, the tape class driver, and the system communication services layer. Paul's recent focus has been on the OpenVMS I/O subsystem and software quality. He is the creator of Faulty Towers, a test tool that injects software faults into VMScluster systems. Paul received a B.A. in computer science and a B.A. in political science from the University of Wisconsin at Madison.

Robert G. Thomson A senior software engineer in the OpenVMS AXP

Group, Robert Thomson led the validation of volume shadowing's port to the Alpha AXP platform. During the OpenVMS port, he measured quality and contributed to functional verification, for which he was co-recipient of an Alpha AXP Achievement Award. Since joining Digital in 1986, Robert has also contributed to improvements in system availability measurement and in symmetric multiprocessing and backup performance. He has a patent and three published papers based on this work. Robert holds an M.S. in computer engineering from Boston University.