How to Control and Manage Change in a Commercial Data Center Without Losing Your Mind

Sally J. Howden and Frank B. Northrup – Distributed Computing Consultants, Inc.

ABSTRACT

Most computer system problems today are caused by change. Change is an innate characteristic of an active computer system. This paper presents an approach whose goal is to minimize and control the impact of problems by controlling and managing change. It is geared towards the system administrator's role in meeting this goal in a commercial data center environment.

System administrators have been given the task of providing reliable, available and supportable computing environments for their clients. A system which does not meet these requirements results in, at the very least, lost productivity, but may also cause a financial loss, and in the worst cases may result in injury to the customers which the business serves (see [1] for an example). In order to provide a reliable, available and supportable computing system it is necessary to minimize the impact on the system's users when problems occur. Almost all computer system problems today are caused by change: changes in hardware components; changes in system or application software; and to a lesser extent changes in processes and/or procedures, or in personnel. The extent to which a system administrator is able to control and manage change is the extent to which they are able to provide a reliable, available and supportable computing system to their client(s).

This paper describes a platform independent approach for pro-actively managing problems in a computing system by managing change well. This approach includes: the process of documenting the computer system's current state; the process of documenting the change; and the process and conditions under which the change is first implemented in a test environment, then in a pre-production environment and finally in a production environment. This approach saves time and effort in the long-term administration of computer systems. The documentation necessary to facilitate this approach is described and some examples provided. This approach is currently being used by Distributed Computing Consultants, Inc. (DCCI) with its clients.

Introduction

Almost all computer system problems today are caused by change. A computer system is defined as a collection of people, hardware, software, and documentation that provides a particular service to customers. Hardware components of a computer system tend to fail during one of two times: the initial *burn-in* period – shortly after having been added to the system; or after having been in operation for greater than five years. However, in today's rapidly evolving computing environment hardware is often replaced with new and improved components well before five years pass. Furthermore, the failure of a hardware component may have little or no effect on users due to the increasing number of vendors who provide systems with redundant components such as CPUs, power supplies and fans; disks with automatic failover capabilities; and hot swappable disks (coupled with mirroring or other RAID approaches). Thus, the hardware failure which is most likely to impact users is that which occurs when a change is made to the hardware.

Likewise, the majority of software failures occur (or in the case of bugs are discovered) when the software is first used following initial installation, after a version update or in a way which is different than originally intended. Again, a problem occurs when a change is made.

When a problem or failure occurs within a computing environment, the first question an experienced system administrator will ask is: *What changed?* Was something added? Was something modified? Is someone trying to use the system in a way different from the defined use? Has something in the computing system's environment changed? Was there a recent power outage? Is there a problem with the environmental controls? This course of investigation – figuring out what *change* has occurred – will often lead the system administrator to the cause of the failure.

So, if change *is* the problem in many cases, a logical course of action would be to minimize the number and length of disruptions to the computer system caused by change. Why should a system administrator be concerned with pro-actively managing changes to a computer system? Why not just react to change by *fixing* the problems as they occur?

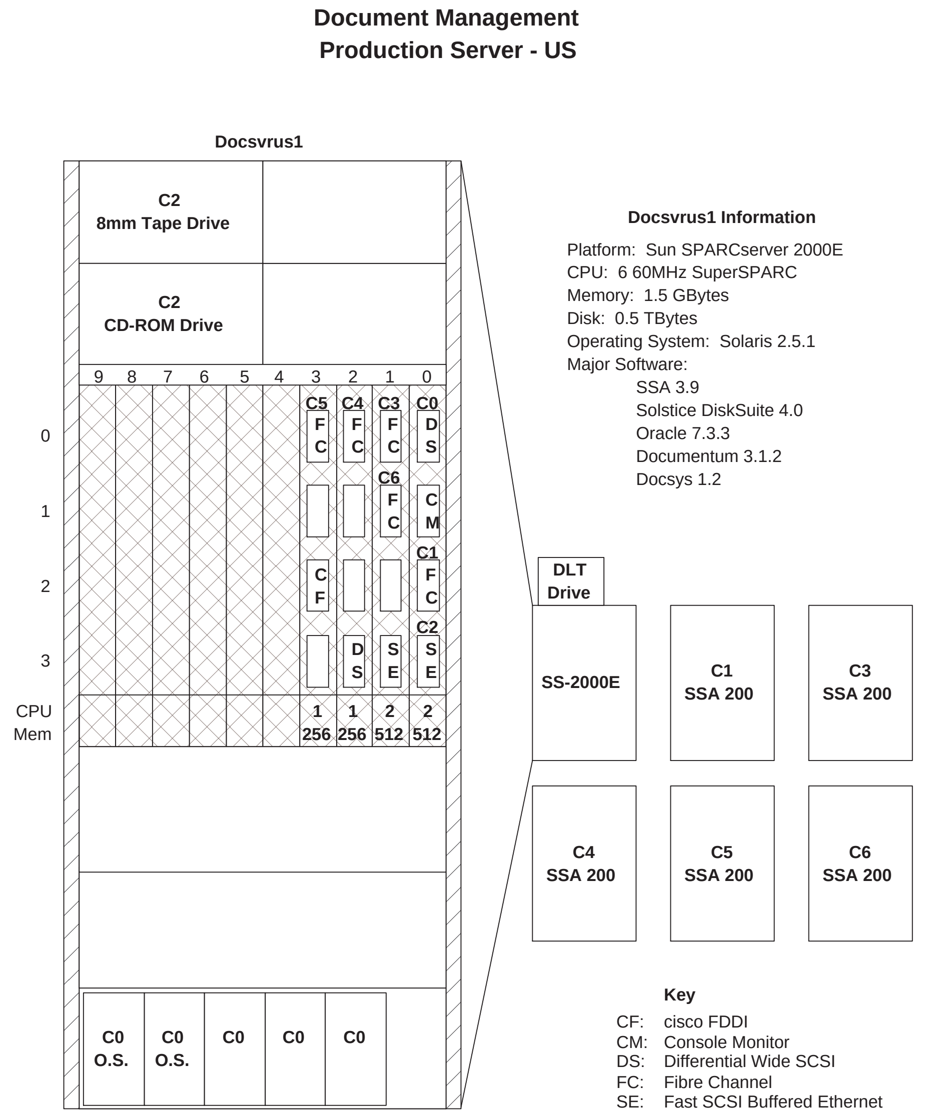
Today, computer processes are integral to almost every aspect of a company's business. An outage can impact product research, development, marketing, time to market, accounting, finances, regulations, payroll, etc. In several of these areas, any outage directly impacts a company's bottom line and ability to compete effectively in the open market.

In a commercial data center every day matters. In pharmaceutical companies, from the time the company submits an application for a new drug to the United States Federal Drug Administration (FDA) it has seven years during which it owns exclusive rights to the drug. However, the company can not sell the drug in the United States until the FDA approves it. After the seven year period, other companies are free to copy and market the drug. The time between the FDA approval and the end of the seven year period can be highly profitable for the pharmaceutical company (i.e., hundreds of thousands of dollars *per day*). Therefore, any computer outage that lengthens the time between application to and approval by the FDA has significant detrimental impact.

Now suppose the company has several drugs at this same stage when the outage occurs.

And the financial well-being of a company is not the only concern. With hospitals, insurance companies, doctors' offices, pharmaceutical companies and pharmacies the loss of information access can directly impact a person's life.

In many ways universities are very similar to commercial entities. A key product for most universities is education. At many universities major changes are scheduled during breaks. The university's ability



Docsvrus1 Information

Platform: Sun SPARCserver 2000E
CPU: 6 60MHz SuperSPARC
Memory: 1.5 GBytes
Disk: 0.5 TBytes
Operating System: Solaris 2.5.1
Major Software:
- SSA 3.9
- Solstice DiskSuite 4.0
- Oracle 7.3.3
- Documentum 3.1.2
- Docsys 1.2

Key

CF:	cisco FDDI
CM:	Console Monitor
DS:	Differential Wide SCSI
FC:	Fibre Channel
SE:	Fast SCSI Buffered Ethernet

Figure 1: Example of a Server Configuration.

to deliver its product to market would be greatly hampered if its computer system was unavailable during the term.

So, if an unavailable computer system can have such a drastic affect on the client, it is vitally important to the client that the computer process not be interrupted. For this reason it is important that the system administrator do as much as possible to insure that the computer process is not interrupted.

The truth is, even a seemingly insignificant change can cause a problem which requires valuable time (including a system outage) to correct. This does not imply that we should never make changes to production systems. Today, change is the rule, not the exception. Therefore system administrators need to think carefully about where, when and how changes are performed. The rest of this paper presents one approach for proactively managing problems by managing change well throughout the life of a computer system [2] rather than reactively dealing with changes as they occur. This approach advocates the following [3]:

1. Establishing & documenting the system's baseline.
2. Understanding the change.
3. Testing the change on test and pre-production systems.
4. Documenting the change before, during and after implementation.
5. Reviewing the change with peers, manager and client before, during and after implementation.
6. Defining a back-out strategy for the change.
7. Training all individuals impacted by the change.

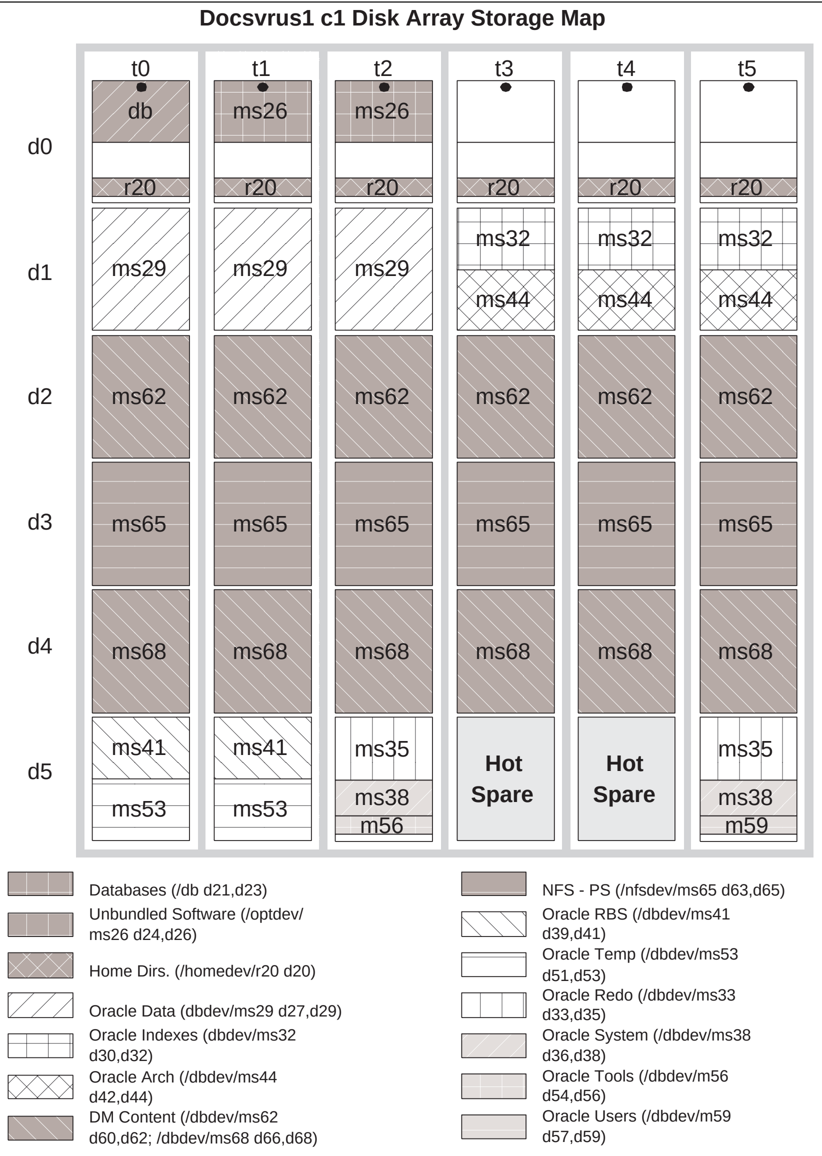

Docsvrus1 c1 Disk Array Storage Map
t0
t1
t2
t3
t4
t5
d0
d1
d2
d3
d4
d5
db
ms26
r20
ms29
ms32
ms44
ms62
ms65
ms68
ms41
ms53
ms35
ms38
m56
m59
Hot Spare
Databases (/db d21,d23)
Unbundled Software (/optdev/ms26 d24,d26)
Home Dirs. (/homedev/r20 d20)
Oracle Data (dbdev/ms29 d27,d29)
Oracle Indexes (dbdev/ms32 d30,d32)
Oracle Arch (/dbdev/ms44 d42,d44)
DM Content (/dbdev/ms62 d60,d62; /dbdev/ms68 d66,d68)
NFS - PS (/nfsdev/ms65 d63,d65)
Oracle RBS (/dbdev/ms41 d39,d41)
Oracle Temp (/dbdev/ms53 d51,d53)
Oracle Redo (/dbdev/ms33 d33,d35)
Oracle System (/dbdev/ms38 d36,d38)
Oracle Tools (/dbdev/m56 d54,d56)
Oracle Users (/dbdev/m59 d57,d59)

Figure 2: Disk Layout Example.

8. Revisiting the defined roles and responsibilities of system personnel.

Establishing & Documenting the System's Base-Line

Before a change is made to a production environment the current state of that environment must be well understood and documented. If the current state is not documented, then all work on the change implementation must halt and the current system state must be documented. It is important to know the state of the computer system in order to properly evaluate the effects of the change. It is just as important to have a defined state to which the system can be restored should it become necessary to back out the change. The documentation specifying a system's base-line should include at least the following:

- **Server Configuration**. This includes the server component layout, disk layout, platform type, amount of memory, number of CPUs, etc. The example in Figure 1 shows a server component layout, platform type, amount of memory, and number of CPUs. An example disk layout is provided in Figure 2 for one of the disk arrays in Figure 2.
- **Software List**. This list includes all major software which has been installed on the system including versions and patch levels. The example in Figure 1 lists the major software packages installed on the system.
- **Roles and Responsibilities**. Each role required to support the computer system is included in this document and its responsibilities defined. The roles should include staff, system administrator(s), dba(s), client(s), application administrator(s), etc.
- **Network Diagram**. The network diagram shows how the system's various nodes are related to each other and the communication path(s) between them. This includes all development, pre-production and production clients and servers, as well as support systems.
- **Installation Steps and Logs**. This documentation includes the steps taken for installing the current operating system and major software applications, and the logs of the actual procedures.
- **Standard Operating Procedures**. A SOP defines the procedure for accomplishing a specified, encapsulated task. It includes all relevant information and instructions detailed enough for an entry level system administrator to perform the procedure. A SOP is written for all common changes to and procedures performed for the computer system. An example of the former is replacing a faulty disk. An example of the latter is performing routine monitoring of the system. Figure 3 shows A SOP template.
- **Change Control Form**. A change control form (CCF) is created to document any change not covered by a SOP if that change has a reasonable chance of impacting use of the computer system. Figure 4 presents a CCF template.

Procedure Name
Standard Operating Procedure
Author
Date

I Procedure Description
II Scope of This SOP
III Complexity
IV Estimated Outage Time
V Backout Contingency Plan
VI Note(s)
VII Supporting Documentation
VIII Procedure
  Step 1
  Step 2
  . . .
  Step N
IX Test Plan

Figure 3: Standard Operating Procedure Template.

Change Name
Change Control Form
Author
Date

I Problem Definition
II Solution
III Scope of This Change
IV Project Team
V Complexity
VI Impact of Not Making Change
VII Estimated Outage Time
VIII Priority
IX Re-Validation Recommendation
X Backout Contingency Plan
XI Status
XII Note(s)
XIII Supporting Documentation
XIV Procedure
  Step 1
  Step 2
  . . .
  Step N
XV Test Plan
XVI Notes on Implementation

Engineer: ____________________ Date: ________
Customer Rep: ________________ Date: ________
QC Specialist: ________________ Date: ________

Figure 4: Change Control Form Template.

- **Backup Procedure**. This document includes a backup schedule indicating the backup levels, a retention schedule, and off-site storage arrangements.
- **Disaster Recovery Procedure**. The list of potential disasters ranges from losing or corrupting data and failure of a system component

to losing an entire site. While not every conceivable disaster can be reasonably accounted for, there should be (3-5) procedures that cover various, reasonable points within that range. Given such a set, it should be possible to find a procedure which addresses a disaster similar in nature to an actual disaster.

- **Test Scripts**. The application Quality Assurance/Quality Control [4] (QA/QC) group should provide an appropriate set of test scripts to be run after any change which impacts the application. As Figure 5 shows, a *test script* is a list of tests to be performed and for each test the expected results, a place to indicate the outcome of the test, and a comment section.

Test Script Name
Test Script
Author
Date

Test List
Test 1
Description
Expected Results
Actual = Expected? (Responsible Initials):
Yes: *No:*
Comments
:
:
Test N
Description
Expected Results
Actual = Expected? (Responsible Initials):
Yes: *No:*
Comments

Figure 5: Test Script Template.

Additional items that may be documented are discussed in [5]. Once a system's base-line has been established and documented, attention can be turned to understanding the change to be made.

Understanding the Change

Before a change is implemented on the production system, it is necessary to understand the change. The better a change is understood, the better it can be implemented and maintained. Many questions need to be answered. What problem is being solved? How does the change solve it? What are the side effects of making this change? What are the implications of not making this change? Who is affected? How are they affected? What is the reliability of this change? What training will be needed as a result of implementing the change? What is the best way to make the change? What is the scope of the change? Who is experienced in this area, with this change? That is, who can help if an unforeseen problem occurs during or after the change? All changes should be well thought out beforehand. This process can be greatly facilitated by having a change control form (see Figure 4) which brings these questions/issues, as well as other's, into consideration. Once a change has been *thought out* it is tested on a non-production system.

Testing the Change

Testing the Change on a Test System

In order to fully understand a change, including how best to implement it and the effect it will have, it is necessary to *make* the change. First, all appropriate documentation is read, including any vendor provided information. Then, an initial Work Plan (WP) is developed. Finally, the change is implemented on a non-production system by following the initial WP, an example of which is given in Figure 6.

At this phase a separate test system is preferred over a common test system or a development system in order to minimize the change's impact on others. However, a common test or development system may be used as long as the impact on users of this system is taken into consideration. This initial testing of the change is not performed on a production system as the change's impact may not be fully understood at this time.

Testing the Change on a Pre-Production System

Once the change itself is well understood and the work plan has been updated from the previous stage, the change needs to be tested within the production environment. However, the test at this stage is performed on a separate pre-production system. This system should exactly match the production system in every way possible, certainly in every way that matters. The development or test system can not be used for this purpose because there are usually numerous differences between these systems and the production system. Developers and system administrators need an environment where they have the freedom to learn and implement new systems, subsystems, ideas, etc. On the other hand, production users need an environment that is reliable, available and supportable [6]. These are very different environments and should be kept separate.

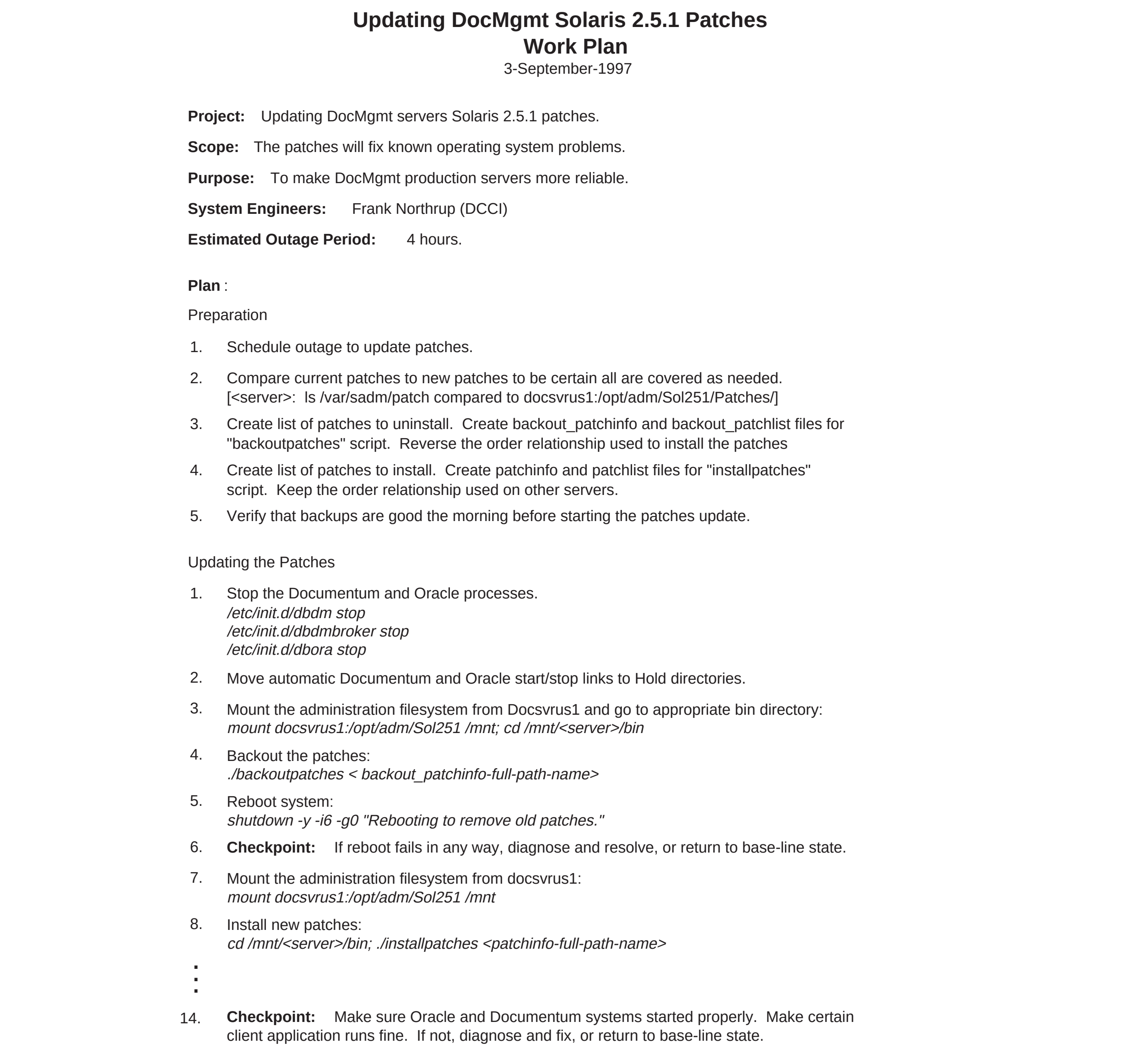
Since the development or test system is significantly different than the production system, this final testing of the change is completed on a pre-production system. This testing includes executing the appropriate QC generated test scripts. These test scripts should provide evidence that the system resulting after implementation of the change still meets the specified QC requirements. Once all testing is complete, the final work plan is created.

Documenting the Change

To keep the current state of the production system documented we must document all changes. Control of a system is kept only to the extent of detail with which each change is documented [7]. This is best done with a combination of a change control form, SOPs, work plans and change logs. Before the change is implemented on the pre-production system the change control form is filled out and the initial work plan is created. The change control form and work plan differ in that the change control form is written for the client and reviewers and contains a high level description of the change implementation. The work plan is written for the system administrator(s) who will be executing the implementation and is a much more detailed description of the procedure. Before the change is implemented on the production system the final work plan is created. As the change is implemented on the production system the actual steps taken, which may differ from the final work plan for unforeseen reasons, need to be logged. Afterwards all other impacted documentation (for example, SOPs) needs to be checked and updated as necessary.

Reviewing the Change

It is always a good idea to have a change reviewed by peers, manager and client via the change control form, work plan and a presentation. Peers are usually the best at understanding, and hence critiquing, the computer system's technical aspects of the change and the overall approach. Furthermore, the better that peers understand the change, the better they will be able to assist in making the change. A manager brings to the table a global view concerning the impact of the change. Furthermore, it is important to have a manager's support of the process. A client representative knows well how the change will impact the client's specific processes and environment, as well as the relative priority with which the client views the change. The more involved a client is in the change process, the more confident they and the system



Updating DocMgmt Solaris 2.5.1 Patches
Work Plan
3-September-1997

Project: Updating DocMgmt servers Solaris 2.5.1 patches.

Scope: The patches will fix known operating system problems.

Purpose: To make DocMgmt production servers more reliable.

System Engineers: Frank Northrup (DCCI)

Estimated Outage Period: 4 hours.

Plan :

Preparation

1. Schedule outage to update patches.
2. Compare current patches to new patches to be certain all are covered as needed.
   [<server>: ls /var/sadm/patch compared to docsvrus1:/opt/adm/Sol251/Patches/]
3. Create list of patches to uninstall. Create backout_patchinfo and backout_patchlist files for "backoutpatches" script. Reverse the order relationship used to install the patches
4. Create list of patches to install. Create patchinfo and patchlist files for "installpatches" script. Keep the order relationship used on other servers.
5. Verify that backups are good the morning before starting the patches update.

Updating the Patches

1. Stop the Documentum and Oracle processes.
   /etc/init.d/dbdm stop
   /etc/init.d/dbdmbroker stop
   /etc/init.d/dbora stop
2. Move automatic Documentum and Oracle start/stop links to Hold directories.
3. Mount the administration filesystem from Docsvrus1 and go to appropriate bin directory:
   mount docsvrus1:/opt/adm/Sol251 /mnt; cd /mnt/<server>/bin
4. Backout the patches:
   ./backoutpatches < backout_patchinfo-full-path-name>
5. Reboot system:
   shutdown -y -i6 -g0 "Rebooting to remove old patches."
6. **Checkpoint:** If reboot fails in any way, diagnose and resolve, or return to base-line state.
7. Mount the administration filesystem from docsvrus1:
   mount docsvrus1:/opt/adm/Sol251 /mnt
8. Install new patches:
   cd /mnt/<server>/bin; ./installpatches <patchinfo-full-path-name>

⋮

14. **Checkpoint:** Make sure Oracle and Documentum systems started properly. Make certain client application runs fine. If not, diagnose and fix, or return to base-line state.

Figure 6: Example of a Work Plan.

administrator will be in the reliability and consistency of the resulting system. This will, in turn, foster a good working relationship between system administrator and client.

Defining a Back-Out Strategy

Although following the change control processes mentioned above greatly reduces the chance of the change causing problems on the production system, something may still go wrong. Thus a back-out strategy needs to be defined prior to implementing the change on the pre-production system. This back-out strategy should take the system back to its base-line state. The back-out strategy itself should be well-documented as part of the change control form and work plan.

Training All Individuals Impacted by the Change

All folks affected or potentially affected by a change need to have training on what the change is and how it impacts them. If a change results in a corresponding change in how clients use the system, then they will need training so that the time it takes to use the system again at the same level or better (if the change resulted in an enhancement to the system) is minimized. Support personnel (staff, system administrators, operators, etc.) need training so that they can continue a high level of maintaining the system and supporting users of the system.

Change Name
Change Management Checklist
Author
Date

Date	Item	Comments
____________	☐ Base-line established	________________________________
	☐ Server configuration	________________________________
	☐ Software list	________________________________
	☐ Roles and responsibilities	________________________________
	☐ Network diagram	________________________________
	☐ Installation steps and logs	________________________________
	☐ Standard operating procedures	________________________________
	☐ Change control form	________________________________
	☐ Backup procedure	________________________________
	☐ Disaster recovery procedure	________________________________
	☐ Test scripts	________________________________
____________	☐ Change control form	________________________________
____________	☐ Work plan - Initial	________________________________
____________	☐ Test on test system	________________________________
____________	☐ Review - Client	________________________________
____________	☐ Review - Manager	________________________________
____________	☐ Review - Peers	________________________________
____________	☐ Back-out strategy	________________________________
____________	☐ Test on pre-production system	________________________________
____________	☐ Work plan - Final	________________________________
____________	☐ Training - affected individuals	________________________________
____________	☐ Roles and responsibilities revisited	________________________________
____________	☐ Implemented on production system	________________________________

Figure 7: Example of a Change Management Checklist.

Revisiting Roles and Responsibilities

Finally, the roles and responsibilities which were defined with respect to the production computing system must be re-evaluated. Some roles may have been obsoleted by the change and/or new responsibilities or roles may need to be filled. In any case, all those affected by the change need to be involved in the evaluation and redefinition of the roles and responsibilities.

Discussion

There are several elements to the approach presented in this paper. In order to help keep track of the completion of these elements a Change Management Checklist is provided in Figure 7. This approach is being used successfully by DCCI with its clients. At one site there have been only five unscheduled outages in 2.5 years for 17 production, pre-production and training UNIX servers with 44 CPUs, 9.5 GBytes of RAM and 1.7 TBytes of disk space. This set includes: one production server with 8 CPUs, 1.5 GBytes of RAM and 0.3 TBytes of disk space; a second with 8 CPUs, 1.0 GBytes of RAM and 0.5 TBytes of disk space; and a third with 6 CPUs, 1.5 GBytes of RAM and 0.5 TBytes of disk space.

At first, it seemed that this approach would improve reliability and availability of production systems, but with the trade-off of significantly increasing the amount of time and effort spent by the system administrator on any given change. However, what we have found is that while the initial effort is greater, we actually save time and effort:

- when repeating a change a month or more after a previous occurrence;
- when transferring knowledge to other system administrators performing the same or similar change;
- when filling in for the primary system administrator;
- by assisting in promoting site consistency;
- by assisting in promoting site standards;
- when diagnosing problems (because everything is already documented).

Conclusion

Bug-free software and transparent changes are a system administrator's ideal scenario. Unfortunately, these are rare occurrences. Further complicating matters is an environment in which employers are reducing their computing system support personnel while increasing the number of systems which need to be supported. It becomes increasingly important that system administrators take and keep control of the systems for which they are responsible. The extent to which a system administrator is able to control and manage change is the extent to which they will be able to provide a reliable, available and supportable computing system to their clients *and* maintain their sanity. This paper describes a platform independent approach for doing just that.

Acknowledgements

Special thanks to Kevin Suboski, owner and President of DCCI, for leading us down this path. Thanks to Kevin, fellow DCCI employees and Pharmacia & Upjohn, Inc. employees for encouraging us during the writing of this paper. Many thanks to Pharmacia & Upjohn, Inc. for providing an environment in which this approach could be developed, tested and refined. Thanks also to our LISA '97 reviewers for their insightful comments and suggestions. Thanks to both of our families for their loving support.

Author Information

Sally J. Howden is currently an Open Systems Consultant with DCCI. She has worked as a UNIX system administrator or consultant for six years. She earned a B.S. in Computer Science and Mathematics from Calvin College, a M.S. and a Ph.D. in Computer Science from Michigan State University. Sally can be reached via email at sallyj@distcom.com .

Frank B. Northrup is currently an Open Systems Consultant with DCCI. Previously he managed the computing facilities for the CS and CIS departments at Michigan State University and Ohio State University, respectively. He has managed UNIX systems for ten years. He earned a B.S. in Computer and Information Science from Ohio State University. Frank can be reached via email at frank@distcom.com .

References

[1] Armour, J. and W. S. Humphrey, "Software Product Liability," Software Engineering Institute, TR CMU/SEI-93-TR-13, ESC-TR-93-190, August, 1993.

[2] "Quality Systems: Part 13. Guide to the application of BS (British Standard) 5750: Part 1 to the development, supply and maintenance of software," BS 5750: Part 13: 1991, ISO 9000-3: 1991, BSi Standards.

[3] "Computer Validation Policy for Pharmacia & Upjohn," Pharmacia & Upjohn, Inc., Version 1.0, April, 1996.

[4] "IEEE Standard for Software Quality Assurance Plans," IEEE Std. 730-1984.

[5] Nemeth, E., G. Snyder, S. Seebass and T. Hein, *UNIX System Administration Handbook,* Second Ed., Prentice Hall PTR, Upper Saddle River, NJ, ISBN 0-13-151051-7, 1995, pp 10, 741-742.

[6] Kern, H. and R. Johnson, *Rightsizing the New Enterprise – The Proof, Not the Hype,* Sun Microsystems, Inc., Mountain View, CA, U.S.A., ISBN 0-13-132184-6, 1994.

[7] Agalloco, J., "Computer System Validation – Staying Current: Change Control," *Pharmaceutical Technology*, January, 1990.