WebSphere® Application Server for distributed
WebSphere Application Server for z/OS®
Versions 3.5 through 6.0

Common malpractices

Eleven ways to wreck a deployment

Version date: April 10, 2006
Level: Introductory

IBM® Corporation
WebSphere Serviceability Team – SWAT

Katie Barrett (katie@us.ibm.com), Manager, WebSphere Serviceability, IBM Austin Lab
Yasuko Bryan (ybryan@us.ibm.com), WebSphere Serviceability, IBM Austin Lab

Alan Booth (aebooth@us.ibm.com), Manager, WebSphere Customer Support, IBM Raleigh Lab
David Cai (davecai@us.ibm.com), WebSphere Serviceability, IBM Pittsburgh Lab
Melinda Carter (mlcarter@us.ibm.com), WebSphere/390 – Level 2 Support, IBM Poughkeepsie Lab
Mark T. Schleusner (schleus@us.ibm.com), WebSphere Serviceability Development, IBM Rochester Lab
Rex Simmons (simmonre@us.ibm.com), WebSphere Knowledge Engineering, IBM Austin Lab
William Wentworth (wkwentw@us.ibm.com), WebSphere Information Development, IBM Austin Lab

Many thanks to the following people for their contributions to this white paper:
David Burdine, John Bukiewicz, and Paul Griffiths

Table of contents

Abstract

The WebSphere Serviceability Team has the opportunity to visit customer sites to solve complex situations. Although this situation is not a regular practice, we extend our service when on-site support is absolutely necessary. After each visit, we document what we have done and observed on site. Often, we revisit our database to determine how and why our customers got into trouble.

This white paper explains the practices and patterns that we have observed while assisting customers. This information is used to raise your awareness. Examples are provided for a better understanding of how and why the trouble occurred. We also explain the possible solutions for each practice.

Introduction

Did your mother tell you "do not run with scissors!" or "Brush your teeth?” Running with scissors is a bad idea. Brushing your teeth is a good idea. If this white paper were about child raising, running with scissors would be called a "malpractice." Malpractice is something that you should not do; a practice that, if you do it repeatedly, the practice can eventually hurt you. Brushing your teeth represents a best practice as it is something in which you would benefit if you did it repeatedly.

This white paper talks about malpractices in the context of Information technology (IT) organizations. The information is based upon our observations from working with customers in emergency situations for the last 5 years. Companies increasingly depend upon their Web sites as part of their business operation. Any downtime costs money, which makes time and tempers short. Getting the Web site up quickly is essential. However, getting the Web site up while performing malpractices will backfire on you one day. We have observed that two thirds of our customers with extended serviceability engagements had committed at least one malpractice. However, we did not observe any malpractice with the remaining one third of our extended serviceability engagements.

We do not think that you will wake up one morning and realize that what you are doing is actually a malpractice. We do not think that you intentionally plan to commit one, two or even all eleven malpractices. Maybe you have never done any of these malpractices. However, watch out when someone tries a short cut as it might be a malpractice.

Katie Barrett
Manager, IBM WebSphere Serviceability Team

Malpractices in the project life cycle

We observed eleven malpractices in our engagements and we have separated them in the life cycle of WebSphere Application Server. We divide the WebSphere Application Server life cycle into the following four stages:

- Planning
- Development
- Validation
- Production and post production

We have listed the most common malpractices that are seen in each stage as shown in the following diagram. The observations spanned all of the WebSphere Application Server platforms, including distributed and z/OS.

Stages	***Malpractice***
Planning	***No capacity or transaction plan*** ***No education*** ***No current architecture plan***
Development	***Application error***
Validation	***No production traffic diagram*** ***No load or stress testing*** ***No test environment = production environment*** ***Changes were put directly into production***
Evolution	***No migration plan*** ***No record of changes***

For all stages ----------------->***Communication breakdown***

Definition of malpractices

This section defines the different types of malpractices.

In-planning stage (requirements - functionality and constraints)

This fundamental stage determines what the final application does and how the resources are allocated. Key plans for the life cycle of the application are made or neglected in this stage.

1. No capacity or scalability plan

Part of the planning stage determines the amount of data that will flow into, through, and out of the application. The type of data and the volume of that data will grow over time. The following examples are problems that we see from a customer who does not have a capacity or scalability plan:

- No prediction for their needed production capacity or response time
- No prediction of which transactions will occur, in what combinations, and how often
- No consideration for the length of each transaction
- No prediction of how many concurrent users will be in production
- No periodic update to the plan as market changes, for example, the holiday season

2. No education

Time and resource constraints to go to class or study educational materials results in limited knowledge on the following topics:

- Problem determination
- Performance tuning
- Available features

3. No current architecture plan

One of the following architecture conditions exist:

- A diagram of the application flow between the various software products does not exist.
- A diagram of where these products reside in the topology does not exist. Where is IBM DB2® located? Where is WebSphere Application Server located? What is in the cluster?
- A current architectural diagram does not exist, but a diagram does exist that is based upon the architecture from 4 years ago.

Development stage (design, implementation)

The development stage is the creation stage, which builds a foundation for the production stage. This stage is more than just the original development of the application; it also includes all of the enhancements or modifications to the application.

4. Blind to an application error

The following list explains some common issues:

- Allocation of large objects causes the heap size to grow too big.
- Redundant computation calls cause increased CPU usage.
- Infinite loops cause increased CPU usage.

Validation stage

This testing phase verifies that the application works as you planned in a complex environment. The environment is similar to your production environment in terms of the network, hardware configuration, back-end database, and load.

5. No production traffic profile

- A diagram of network, routers, switches, and hubs does not exist.
- Data on the capacity of the networks or network segments does not exist.

6. No load or stress testing

- Load or stress testing is not done.
- Load or stress testing is done, but the testing is not based upon the production load.
- The transaction pattern is not simulated accurately to the production transaction pattern.
- The load is simulated accurately, but the back-end database size is significantly smaller.

7. No test environment = production environment

- The test environment is too small or not available when it is needed.
- The test hardware, network, and software are different from the production environment.
- The z/OS LPAR on the same machine or network is not isolated from the production system.
- The configuration settings for the test systems are different from settings for the production systems.

8. Changes put directly into production

- Changes to a configuration and a fix are put into the production system directly before it is discovered that the changes do not work.

- A test environment does not exist that is equivalent to the production environment, which can be used to simulate the problem or the load. The customer is forced to test the changes live in the production environment.

Production or post-production stage

This stage is the key for the life cycle of the application and is far longer than the original application development. However, it is possible to overlook this stage. The application is now in production so problems are very public and costly.

The software that the application is built upon evolves. Thus, co-requisite packages need upgrades and integration testing. Hardware and operating system might be upgraded and then the application itself might expand or interface with additional applications. All these changes require adequate planning, documentation updates, and plan completion. The following related issues have been observed:

9. No migration plan

- Adequate time is not allocated for migration.
- Enough investigation time is not taken to determine how new Java™ 2 Platform, Enterprise Edition (J2EE) specifications affect the system.
- Developers are unaware of the co-requisite software requirements.

10. No record of changes

- A record of changes is not made. Changes are made under stress.
- Multiple groups make changes to the environment without records and without communicating with each other.

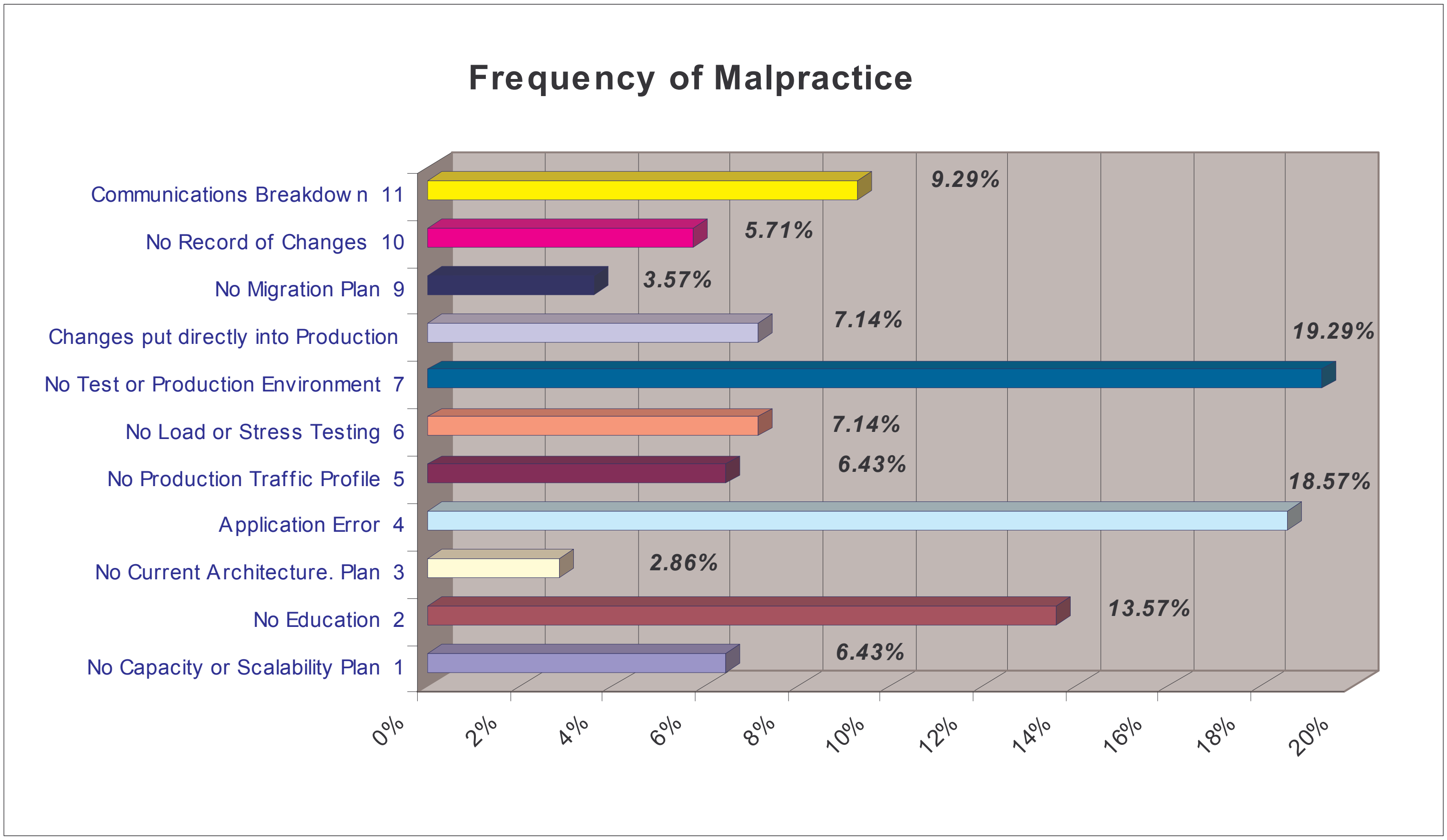
For all stages

11. Communication breakdown

Communication is critical in today’s business environment where things change rapidly. The key is clear, clean communication between the right people. Miscommunication frustrates other people and can lead to a misdirected effort. In some cases, effective communication was not observed between customers, IBM and customers, and customers and the various vendors.

Percentage of occurrences observed in 2005 WebSphere Application Server Extended Serviceability Team engagements

WebSphere Application Server Extended Serviceability Team works with a small percentage of WebSphere Application Server customers. However, malpractices are a very common occurrence among those customers and are often directly related to the situation in which the customers find themselves. The following chart describes the frequency of the malpractices for the WebSphere Application Server Extended Serviceability customers in 2005.
Note: Although the sample was taken from WebSphere Application Server customers, the malpractices are not specific to WebSphere Application Server. For example, if you do not complete stress testing on WebSphere Application Server, then you might not complete it on the other products or applications in that same environment.



Observation:

- “No test environment that is equivalent to production environment” has the highest occurrences among the customers who got into the highly critical situation.
- “Blind to application error” is the second highest occurrence.
- “No education” is the third highest occurrence.
- “No test environment that is equivalent to production environment “often triggers ”Changes put directly into production“ and ”No load or stress testing”

Sample story and possible solution for each malpractice

Planning requirements - functionality & constraints

1. No capacity or scalability plan

While it is true that a capacity plan is not absolutely necessary, it is a plan that you need if the e-business ever needs to grow. The plan is not as simple as buying a bigger piece of hardware. The simple plan will work for a while, but it probably is not the most cost-effective way to increase an application's capacity.

When a good capacity plan is put into place, it includes a good High Availability (HA) plan. The need for a HA plan is not unusual. Every business wants to get the most work with the least amount of hardware for the entire business day. A business cannot accomplish those goals without a HA and capacity plan in place. If the entire application is put on a single piece of hardware, it might be able to handle the capacity for a while. However, if that piece of hardware goes down, the entire application is not handling traffic and the business is not receiving income. If that piece of hardware is two smaller pieces of hardware that can handle the same capacity, the hardware cost is nearly the same price, but the chances of losing the entire application are now much less. If this process of thought is taken out even further and in more detail, then it is a good beginning to a pair of capacity and HA plans.

Case study:

The following case study shows the importance of a good, well-researched capacity plan. The customer thinks that the client's workload is not getting properly distributed, which results in a capacity issue. The customer had planned for enough hardware and had incorporated a good HA plan into the capacity plan. However, the application is not able to handle more capacity as demand increases. CPU, memory, and network utilizations are all thought to be the problem.

As the code review of the application is completed, the problem is discovered in the application. The client is doing cell-scoped Java Naming and Directory Interface (JNDI) calls where every client looked up the enterprise beans through the deployment manager. Even though the application is spread out across six nodes, the deployment manager is the problem.

The point here is that a capacity plan is more than a hardware plan. The capacity planning needs to take the application, the application deployment, and the application client use into account. When the naming calls were restructured to go to cluster members, the problem went away and the application was able to handle more traffic. The WebSphere Application Server Development Team learned from this problem and now caches naming more strategically in response to the client calls.

Possible solution and preventive care:

When you put a capacity plan together remember that the plan is for what both the hardware and the application can handle. The biggest mistake in capacity plans is overlooking the application. With a well-tuned application design, you can save hardware costs. Do not overlook what the application uses.

Use the following checklist to prevent this malpractice:

- ✓ What does the capacity plan for the back-end database say it can handle for an additional load?
- ✓ What can the network routers handle for increased network traffic?
- ✓ What can the Lightweight Directory Access Protocol (LDAP) server handle for user authentication?
- ✓ If the application is spread out across a Wide Area Network (WAN), are remote calls kept to a minimum?
- ✓ Is part of the hardware also being used for disaster recovery or HA?
- ✓ What is the maximum amount of hardware utilization before you need to provide additional hardware?
- ✓ Are object types being properly used for increased capacity?
    - Entity enterprise beans with a caching policy for the bulk of client database reads
    - Stateful enterprise beans have been passivated to a fast storage device

2. No education

WebSphere Application Server built to comply with an open, but evolving standard. Each new release of WebSphere Application Server conforms to a succeeding standard of J2EE for application servers. As the standards change, so do the succeeding releases of WebSphere Application Server.

Customers generally take the time to learn their first application server, but unfortunately a few customers do not take the same time to learn the changes that have occurred in subsequent releases. For example:

- What is new?
- What is deprecated?
- What is not there any more?

The typical reasons for not investing in education are “We do not have time to send someone away to the class” or “Do not tell me about education now!” or “We can not afford to have key people out of the office”. The lack of education leads to the following production crisis:

- Migration problems: Each Application Server release evolves from the preceding release. Customers who do not review their application for any changes it might need to run efficiently under the new release might have the following problems:
- Performance problems: What was efficient in the *n* release might not be optimal in *n* + 1 or *n* + 2 release.
- Functional problems:

- The function that is used in the *n* release might be depreciated in the *n*+1 release and removed in the *n*+2 release.
- New function is not used. This opportunity might be missed because each enhancement reflects the increasing capability that is requested by customers and reflected in the new J2EE standards.
- The original application was not J2EE compliant, but the release to which the customer is moving only handles J2EE compliant applications.

2. Integration problems: Each release has established explicit levels of pre-requisite or co-requisite software. You might encounter problems and affect the performance of your application if you do not check and migrate to the pre-requisite and co-requisite software,
3. Frustration, which usually shows when WebSphere Application Server does not work as the customer expects.

Case study:

A toy company started their online site using WebSphere Application Server Version 3.5. The company successfully moved to version 4.0 using the version 3.5 webserver plug-in without making any application changes. Now, version 4.0 is out of support. Their online business has also grown so they need the added performance of version 5.1 or version 6. The company decides to move to version 6 because it would provide them with more time on a single level of WebSphere Application Server.

The company buys WebSphere Application Server Version 6, installs it, and then deploys their application. However, the application does not deploy and nothing works. The company files several Problem Management Resolution (PMR) reports, tells IBM Support that their application used to work correctly, and believes that WebSphere Application Server must be the culprit because they did not change their application. They are frustrated.

Were they aware that WebSphere Application Server Version 5.1 and Version 6.0 conformed to the J2EE standard? Yes, but because the application ran on Version 4.0, which also was J2EE compliant, they believe that the problem has to be with WebSphere Application Server.

Did they investigate what is in WebSphere Application Server Version 6, which is their target version level? Did they check all of the changes that are required to move from the version 4 plug-in to version 6? - No

Did they allow any time to learn the WebSphere Application Server enhancements and changes? - No

Did they plan to learn about these enhancements and changes? - No

Possible solution and preventive care:

Allow time to periodically review the application for the following changes:

- Changes in business needs
- Sections that are affected by J2EE capability changes

  This issue implies that you need the time to learn about the new functions or the depreciated functions in the new WebSphere Application Server releases. Education no longer implies that you must go to the classroom or decipher a manual by yourself. IBM now has the following free educational opportunities:

a. IBM Education Assistant:

The IBM Education Assistant (IEA) enables you to learn from your desktop without going to a classroom. There are many educational plug-ins that are available for Rational®, WebSphere, and Lotus® products, including WebSphere Application Server and WebSphere Portal Server. For more information, see the following Web site:

http://www.ibm.com/software/info/education/assistant/

b. Technical Exchange:

The WebSphere Support Technical Exchange is another free learning opportunity. These learning opportunities involve live presentations that are open to all customers on topics of interest. You can suggest topics for these sessions. All sessions include a question and answer time and presentations are done several times per month. Previous talks are available for viewing after the live presentation. For more information, see the following Web site:

http://www.ibm.com/software/websphere/support/supp_tech.html

c. Useful technotes

Technotes and techdocs contain information that is provided by the IBM Support and Information Development teams to assist you in maintaining your WebSphere Application Server environment. You can search on any topic to find a related technote, but IBM Support suggests that you periodically check the featured documents. These featured documents are technotes that apply to many customer situations, from performance to migration, and so on. For more information, see the following Web sites:

For WebSphere Application Server for Distributed platforms, see the following Web site:
http://www.ibm.com/support/docview.wss?rs=180&uid=swg27006233
For WebSphere Application Server for z/OS platform, see the following Web site:
http://www.ibm.com/support/docview.wss?rs=404&uid=swg27006898

d. Bookmark IBM Redbooks

http://www.redbooks.ibm.com/redbooks.nsf/redpapers/

e. Navigate IBM Support Assistant (ISA)

http://www-306.ibm.com/software/support/isa/

- Join the Global WebSphere User Group Community: http://websphere.org

3. No current architecture plan

What does your application do and how does it do it? Where does it get the data and where does it put the answers… and all the other things in between? How does the next person know how to safely add a feature when new function is added? How do you re-arrange the environment when different systems, either software or hardware, are added, removed, or replaced in the environment? The key to doing any of these tasks is having a current, up–to-date architecture plan. Having an expert who knows it all is convenient, but not sufficient. Having a rough sketch of the original architecture plan is handy, but it not very helpful 6 months later.

It is certain that changes will occur. The best you can do is to plan to make that change easier for yourself. In the case of the architecture, having an up-to-date architecture plan can help avoid production outages and, in the case of an outage, save you weeks or months of problem determination.

Case study:

A large retail company uses WebSphere Application Server for its online business. When the company started, they hired an IBM Software Services for WebSphere consultant to assist them with developing the architecture to ensure that they had the needed capability. The architecture was defined, documented, and implemented.

The company purchased other businesses and planned to add these businesses to their online shopping application. The company also added other services to their online Web site, such as signing up for the store credit card and making personal services appointments. Each division of the company brought these requests to their central IT department, where a person was assigned to add the function. Some of these additions were documented in a memo, in notebooks, and one was documented in an addendum to the original architecture plan.

Three years and many changes later, there is a production outage and no one can figure out what is wrong. The customer opens a severity one PMR for WebSphere Application Server and says, “Nothing was changed. It just hung”. An IBM engineer visits the customer on-site and asks for the architecture plan so that he can understand the data flow. The customer looks for the architecture plan, but is unable to locate it. The last update was two years earlier and it only included three updates after the original design was complete. The situation reveals that most of the additions over the recent two-year period were never added to the original architecture plan. No one at the customer site really knows how the application flowed, or how it should flow.

The situation results in two weeks of intense work by both the customer and IBM, and many subsequent applications hang instances. The problem resulted from a new credit card feature that was added several months ago. The feature caused a database lockup with an older credit card billing feature when the feature was accessed during that customer's quarterly bill calculation. The error did not surface until the quarterly billing cycle began and additional customers from the new business acquisitions were added.

What is still not known is whether there are other potential deadlocking flows still in the application or whether new additions will cause new problems. Without a current architecture plan, time-consuming debugging must be done to determine the application flow and deadlocks. Without processes in place to keep the architecture document up-to-date, the document becomes less useful as changes are made and personnel turnover occurs.

Possible solution and preventive care:

The key here is self-imposed self-discipline: Place a priority on keeping your architecture documentation up-to-date. Maintain an architecture document that can minimally answer the following questions:

- Who changed what, when, and why?
- What is the data flow?
- What is the environment including both hardware and software?
- What are the internal structures?

Additionally, you can use one of the architecture patterns that are recommended for the level of WebSphere Application Server that you are using. These architecture plans are defined in IBM Redbooks. The pattern recommendations might vary by WebSphere Application Server version level and do not absolve you of maintaining documentation for your own installation, but do provide a solid starting point. For more information on IBM Redbooks, see the following Web site: http://www.redbooks.ibm.com/

Development-design and implementation

4. Application error

Application error is an area that IBM encounters a lot of problems when we work with customers. These problems are often thought to be WebSphere Application Server problems. Most often, however, the application error is because of the previous section; a current architecture plan does not exist. Arguably, the most important piece of the architecture is the application. Without the application, there would not be an e-business. For that reason alone, tightly follow the application life cycle. The problem is that you might forget how important it is to manage the application life cycle. Even more customers forget to manage the application life cycle after the application has been deployed into production. Application errors often are the result of not managing the application life cycle.

Case study:
A large U.S. company ventures into a new area of business and has some major problems with their application. The problems are so severe that it holds up the production date of the application, which is a serious issue for the company. It is eventually decided that they would put the application into production and the daily outages would be endured until the problems are addressed. The issue is that WebSphere Application Server just restarted by itself.

The application was developed by a third-party company and included pieces from other companies in the application as well. The application went thru a good design review with the customer and IBM also participated in these reviews. The application was thoroughly unit tested and held to a good application life cycle. It was later used in pre-production where estimated live loads were simulated. In pre-production, WebSphere Application Server just restarted itself.

IBM and the customer spend a lot of time collecting trace information from the Application Server and from the Java virtual machine (JVM). The third-party company, who built the application that contains pieces of extraneous code and is causing WebSphere Application Server to restart. The extraneous code that the third-party company thought would not cause any harm is causing the JVM to hit page faults and throw a kill signal, which causes the Application Server to restart. After the extraneous code is removed from the application, the problem goes away.

Possible solutions and preventive care:

Ask yourself the following questions on a regular basis about the application life cycle:

- Does the application do what it is intended to do?
  Every application is written to solve a problem. The problem might range from offering a product for sale to helping solve a medical mystery. The point is that the application is written to solve a problem. The problem is that point is forgotten over time. You add a little function to an application here and a little function there. The next thing that you know, the application is now selling books on medical mysteries and helping solve the medical mystery too.

  Keep the application distinct and use the application for what it was originally intended to do. If the application needs to do multiple things, like sell books and do research, separate it into multiple applications. Thus, when there is a problem with the application it is easier to isolate what that problem is and help reduce the impact to the business. If the whole business is written into a single application, and that single application has problems, then the whole business has problems.

- Is the application designed using a modeling language?
  The quality of an application starts with the quality of the idea for which the application is intended. The design of that idea is the next step in producing a high quality application. If the application design is scratched out on napkins verses in a modeling tool using Unified Modeling Language (UML), the application will reflect it. There are times when applications have problems that are on a much higher level than a NullPointerException exception from something that was not properly checked.

Problems can and do begin in the application design. When problems begin at the application design level, the problems become extremely difficult not only to discover from a serviceability point of view, but even more difficult to fix. Make sure that you have a solid design. Review the design, re-review it, and keep it up-to-date. If the design needs to change because of a technical oversight, update the design and determine how that change impacts the rest of the application. It is not a good time to find out what the impacts are when the application is deployed and is responding to traffic.

- Is the application code-reviewed by someone other than the person who wrote it?
  Like the application design, the quality of the design implementation reflects how well the application performs the task that it is designed to accomplish. The design implementation phase is also a phase when a lot of problems occur. Every person knows that if you put a design in front of 100 programmers, the chances are extremely good that those 100 programmers will implement the design in 100 different ways. The different implementations are not a result of a poorly written design, but occur because each programmer has his or her own interpretation and implementation.

  During the code review, also look at how the translation of the design to code is implemented. Where are the best data structures used most effectively? Were the objects themselves managed properly to prevent memory leaks?

  The best way to make sure that the code matches the design is to have a second or third person, or a team, review and compare the design to the code. The second person or team also needs to make sure that the code is technically correct. This process saves a lot of time later when the application problem is not in the design.

- Is the application serviceable?
  During the code review, was the code reviewed from the serviceability point of view? Programmers generally do not think that their code is going to have problems. Well, if that situation was true, you would not be reading this paper.

  Code serviceability is extremely important. Both WebSphere Application Server, and the applications that run upon it, must have good serviceability. There are times when the serviceability quality of WebSphere Application Server is used to help find, troubleshoot, and resolve problems that are purely application problems. In these instances, the application developer did not put good serviceability code into the application.

  To resolve application errors, the last measure of defense is application logging. Ask the following questions:
  - Is it more important to save a few hours of a programmer's time at the cost of a programmer's wage?
  - Is it more important to get an e-business back in business at the cost of lost revenues that the e-business is not generating while it is unavailable?

Validation to ensure that the software meets the needs

5. No production traffic profile

This malpractice involves a missing network diagram for the routers, networks, switches, and hubs. It also involves the lack of data on the capacity of the networks or network segments.

How do you get somewhere you have never been before? Do you ask for directions? Do you use a map? How well does this process work out for you? What happens when you get bad directions or a road is blocked with traffic? These questions might seem silly for an article on Web application development, but cars are not all that different from your data. If a highway is too clogged to handle the level of traffic, it grinds to a halt. This situation happens in computer networks all the time. We know this is going to happen and we know how to prevent it, but our budgets and deadlines do not always allow for proper, timely intervention.

How do highway planners know what size road to build? How do they know where a new stoplight will save lives? How do shops and stores know where to locate for visibility? Municipalities create complex traffic studies and models of future growth prior to investing in new highway infrastructure. This planning enables municipalities to plan for future growth and accommodate their citizens for decades to come. The same type of study and attention to detail is needed when building an enterprise-level Web application.

How does your data know how to get from one area to another? A routing table is analogous to a road map. The routing table makes for an accurate, but not very detailed roadmap. You need to take this process a step further to really get control of your data. You must have a detailed view of your network topology.

Case study:

An insurance company claims to have a hang condition in their WebSphere Application Server configuration. IBM Serviceability analysts are engaged and an apparent hang situation is not reported in the software. The operating system does not report any problems. Problem determination rules out all of the possible problems except in the network.

The system analyst that is assigned to help troubleshoot the issues suspects that there might be problems in the network and asks to review the network flow charts. Unfortunately, current diagrams of the network do not exist! The analyst interviews the teams that, together, are responsible for maintaining the network. Slowly, they realize that a new hub was recently added to the network. This hub is used to share the traffic with a different area and is making the rest of the network slow down to the degree that WebSphere Application Server seems to hang.

Possible solutions and preventive care:
Create a network diagram and keep it current! Add hardware to your network carefully and communicate your plans with the entire team as even the most innocuous changes can have far-reaching impacts that you have not anticipated.

You need to have the following information:

- Know the location of the routers and firewalls.
- Know where the servers are located on each network segment.
- Know how much traffic each server is expected to handle.
- Ensure that there is sufficient bandwidth to handle the traffic.

Matching network design to the expected traffic is both an art and a science. Combining experience and data will tell you how well your network will hold up under pressure.

6. No load or stress testing

After functional tests, users are expected to conduct thorough and vigorous load and stress tests on the application before moving it to the production system. Ideally, the load and stress tests involve every part of the application and simulate the real-world workload and user patterns. The purpose of these load or stress tests are to determine the performance impact by putting the system under extreme situations and to uncover any remaining program errors that have not been found during function test. A truly robust, production-like series of tests executed repeatedly can be very useful in the identification of 'slow' Java application memory leaks. If measurements are made of memory usage, in addition to the measuring of performance, this will do well to identify memory leaks from the application. These 'slow' memory leaks need the volume and repetition of stress and performance test to help make their presence known before the application goes into the production environment.

Despite its importance, some customers choose to bypass load or stress tests and move directly from a development system to production system. The consequences can be dire, as shown by the following real-world lessons:

Case study 1:
A European company is experiencing severe performance issues with WebSphere Application Server Version 6.0 for the Linux® on zSeries platform. The utilization rate causes a system slow down, which, in turn, leads to long response times for the users.

Upon close examination, the IBM Serviceability team finds the following problems:

- The z/OS virtual machine (zVM®) is not tuned properly. It is set to 2 gigabytes (GB) of Extended Storage when the recommended amount for their setup is 4 to 6 GBs.
- The application was designed poorly and has a potential deadlock.
- The application has many large objects.

To find a proper setting and spot potential program errors, the Serviceability Team conducts load tests with 1 to 500 simulated users. Through the load testing, IBM and the customer determine the optimum settings and the deadlock within the application code. After the modifications are put into the production system, the application runs smoothly without a performance issue.

Case study 2:
A large media company finds that its catalog application experiences frequent slow down and the Web browser returns blank pages. The IBM Serviceability Team examines the log file and finds excessive activities for the garbage collector.

The Serviceability Team conducts stress tests in a quality assurance environment and notices a correlation between the garbage collector and downloading large files that are greater than 1 GB. The root cause is a servlet that attempts to cache large files in memory. The problem disappears after servlet caching is disabled.

Possible solutions and preventive care:
The previous two cases show how important it is to conduct thorough and rigorous load and stress tests. In both cases, if the customers had conducted these load and stress tests in a quality assurance environment, the problems might have been spotted long before they moved into a production system and caused the outage.

You must realize that unit and system tests are not substitutes for load and stress tests. Typically, load tests are conducted to verify that the new system satisfies its design specification. Stress tests verify the stability of system under load and extreme situations.

Existing applications
For existing applications, you must have a good understanding of their peak loads, duration, and their time of occurrence. You also must analyze the pattern of usage, for instance, how many concurrent users are actively engaging in transactions, how many users are browsing the catalog, and so on. Those patterns might have a significant effect on system workload.

You can use this knowledge to prepare the testing script. Prepare a test script that can drive up the workload to a level that is similar to the peak load with a comparable usage pattern. Ideally, perform the load and stress tests on the same production system before it goes live in a production environment. If it is not feasible, find a Quality Assurance system that is similar to the production system.

New applications
When you do not know how a new application might be used, you must rely on the design document and common sense when you prepare the test script. It is prudent to be conservative at beginning and prepare for the worst-case scenario.

Useful links for Java memory leaks:

- **Handling Memory leaks in Java programs**
  http://www.ibm.com/developerworks/java/library/j-leaks/
  In this article, you'll learn what causes Java memory leaks and when these leaks should be of concern. You'll also get a quick hands-on lesson for tackling leaks in your own projects.

- **Java Performance Hints**
  http://www.ibm.com/servers/eserver/zseries/software/java/javafaq.html#perform

This is the section of Java on z/OS with several other resources references.

- **VerboseGC Diagnostics - Diagnostic Tool for Java Garbage Collector**
  http://alphaworks.ibm.com/tech/gcdiag
  This helps you size your heap properly and make sense of the overwhelming amount of data in verboseGC output.

- **Heap Analyzer**
  http://www.alphaworks.ibm.com/tech/heapanalyzer
  This gives some statistics about your heap and allows you to look at object of interest. It takes a heapdump as input.

- **MDD4J - Memory Dump Diagnostic for Java**
  http://www.ibm.com/developerworks/websphere/downloads/memory_dump.html
  Memory Dump Diagnostic for Java technical preview can analyze common formats of memory dumps (heap dumps) from the Java virtual machine (JVM) that is running the WebSphere® Application Server.

7. No test environment = production environment

It is recommended that you have a separate test system that is identical to the production system. You can conduct load and stress tests on the separate test system before moving to production system as described in previous section. The following advantages exist for a separate test system:

- Separates the test system from production system and prevents unintended production disruption
- Provides a platform for you to perform functionality and integrity tests before performing major upgrades.

Most WebSphere Application Server users have certain kinds of test environments. However, these environments are not necessarily identical to the production system because of the cost factor. There are cases where the lack of a separate test environment caused a product outage.

Case study:

A major bank in South America discovers that their WebSphere Application Server for z/OS environment experiences frequent product outages during tax season, which disrupts online tax business for their customers and damages the relationship with their customers. The IBM Serviceability Team investigates and finds many factors that contribute to this situation, but primarily finds configuration errors.

One particular mistake is that they created two application servers on a single installation of the base version for WebSphere Application Server. One of the application servers is the test server and the other application server is the production server. Because both application servers run on the base Application Server version, their logs are shared; their ports are carefully configured to avoid conflict; and, most importantly, because they run on the same binary codes base, any upgrade to the Software

Development Kit (SDK) disrupts both of the application servers. While frequent updating to the test system is necessary, the disruption to production system is intolerable.

Recognizing this mistake, the Serviceability Team urges the customer to separate the Application Server test and production systems on different logically partitioned modes (LPAR) so that they do not affect each other. The subsequent problem determination is smooth after the separation is completed.

Some developers fail to create a separated test system for practical or economic reasons. It might be convenient to have the test and production system together, or it might look wasteful to assign a dedicated test system. You must understand that this convenience comes at a price and product system outage is very costly.

Possible solution and preventive care:

There are a number of approaches for creating a test environment. Each approach varies in cost, planning, and risk. Having the ability to test the application environment, not just the application, is a necessity.

Use the following checklist for the test environment:

- ✓ Understand that a test system is a must, not a luxury; do not use a production system for test purposes.
- ✓ Physically separate a test system from a product system and ideally have two different groups of operators manage these systems.
- ✓ Clearly label the test and production systems with a different set of security identities.
- ✓ Install identical software on the test system as the production system; it is recommended that you use similar hardware for the test and production systems.
- ✓ Upgrade WebSphere Application Server and the applications themselves on the test system first before upgrading the production system.

The following list describes the different types of test environments that are based upon the availability of resources:

- ○ Maintain an exact duplicate of the production environment.
- ○ Maintain a scaled-down environment with load generators that duplicate the expected load.
- ○ Maintain a model or simulation of the environment.

It is recommended that you implement and test all of the changes in a test environment before they are added to the production environment.

8. Changes put directly into production

Although most companies understand the difference between test and production systems, many of them find it convenient to put the change into the production system directly, which bypasses the test system altogether. The common argument for bypassing the test system is usually that the update to a production system is so urgent that the system administrator must make an exception.

It is true that sometimes the production system needs urgent patching. However, it is recommended to use the formal channel. For example, apply changes to test systems, conduct thorough tests, and then move the changes to the production system. Applying changes directly into the production system defeats the purpose of test system and leaves the production system with undocumented changes. Additionally, direct updates to the production servers might create different versions and patched levels of WebSphere Application Server on one system and make future maintenance a nightmare.

Case study 1:
A large media company experiences frequent Web site outages. The IBM team determines that the outages are a result of a memory issue; application code causes heap fragmentation. IBM suggests different garbage collector settings, but receives inconsistent results from different machines. Close examination reveals that the four WebSphere Application Server installations are not identical. Each installation has a different SDK level, different fix pack, and even a different interim fix. It is surprising that WebSphere Application Server worked at all. After upgrading each installation to the latest SDK version with the proper garbage collector settings, and the application code, the heap fragmentation problem disappeared.

Case study 2:
A large bank diagnoses a product outage issue. When IBM checks the design document, the team determines that the design document has not been updated in two years. Numerous modifications have been made to the production system without any record.

It might be convenient to put changes directly into a production system. However, it is not a recommended best practice and should be avoided. On a large system, it is very difficult to apply identical updates to multiple systems and even more difficult to keep track of those changes. When a production system has problems, it is very difficult to perform problem determination. Furthermore, the casual attitude towards the production system can potentially compromise the security of critical data.

Possible solution and preventive care:

Use the following checklist to prevent changes from being put directly into a production system:

- Establish a rigorous procedure for making any changes to the production system. Record all of the changes.
- Establish a production system administrative group that is independent from the development team and is responsible for the production system only.
- Assign specific maintenance time for patching and updating the production system. Do not make changes to the production system during other times.
- For urgent situations, particularly security-related situations, you may make changes to the production system directly, but make sure that the changes are well-documented and

considered exceptions. Also, ensure that the changes are applied to the test systems as soon as possible so as to keep all the systems identical.

Production and post production

9. No migration plan

Product migration can be a real challenge for even experienced developers. J2EE migration is relatively easier in comparison to other products because the J2EE standard has been trying to make the J2EE server more platform- and vendor-independent.

The following types of major migration occur:

- Upgrading WebSphere Application Server from one major version to another version, for instance, from version 4.0 to version 5.0.
- Migrating from another J2EE server or another operating system.

To ensure a smooth transition, make a detailed list of differences and modify the code, if necessary. Otherwise, the migration might adversely affect the performance and stability of the production system.

Case study:

A major bank in the Asia Pacific region migrates from WebSphere Application Server Version 4.0 to Version 5.0.The bank found that performance degrades significantly. They upgraded to a newer WebSphere MQ Workflow version and find that the system crashes randomly.

Close examination reveals the following issues:

- WebSphere Application Server Version 5.0 is J2EE Version 1.3 specification compliant, which requires the J2EE container to implement a “shareable” connection, as the default setting while the default setting in J2EE Version 1.2 is not shareable. Developers did not change either the code or settings. This situation causes the application that was written for J2EE Version 1.2 to hold the connection longer than necessary particularly for deeply recursive and long forwarding method calls.

For WebSphere MQ Workflow, the newer version uses the Java Native API instead of the Java Native Interface (JNI). Java Native API adds certain synchronization blocks to prevent concurrent access to WebSphere MQ Workflow session objects. The developer failed to change the application code, which caused the deadlock and system crash.

Clearly, developers should carefully analyze the differences between versions, analyze the J2EE specifications and other settings, and make the necessary modifications to either the application code or the default settings. The J2EE specification tries to make migration easier, but it does not catch everything.

Possible solutions and preventive care:

During migration, developers should refer to the existing documents on the current system, compare them with the newer WebSphere Application Server version, and document all of the differences and how to reconcile these differences.

Make note of the following items to prevent some migration hazards:

- Changes in the J2EE specifications
- Differences in the default settings between versions
- Differences in the vendor extensions
- Changes to the operating system-specific configurations

Useful links for migration:

- **WebSphere for z/OS Version 5.1 - Migrating Configuration from V5.0 to V5.1**
  http://www.ibm.com/support/docview.wss?rs=404&uid=tss1wp100441
  Some work is required on your part prior to using the migration utilities for WebSphere Application Server. This site provides a concise, step-by-step explanation of the process that is used to migrate your configuration from version 5.0 to version 5.1. The "WebSphere 502 to 51 Migration Worksheet" document provides a set of concise data-capture worksheets. The worksheets are helpful tools to use when you are capturing the critical data that is required by the migration process.

- **Migrating from WebSphere for z/OS V5.x to V6 - An example migration**
  http://www.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100559
  This white paper provides an illustration of an actual step-by-step migration so that you can see how it is accomplished and what some of the issues might be when you migrate. This white paper is not the definitive source for migration information; rather, it is an example of one actual migration.

- **IBM Education Assistant: WebSphere Application Server for z/OS V6.0.1 installation and migration**
  http://www.ibm.com/support/docview.wss?rs=404&uid=swg27006058

10. No record of changes

We forget what we have changed in matter of a few days. It might be an application change, a WebSphere Application Server configuration change, or an operating system-wide change.

The more people who have access to the production and test systems, the more important it is that you have a centralized change record. In this record, everyone who has access to the system can keep track of the changes so that the record can help you to narrow down the problem areas.

Case study:
A pharmaceutical company is testing a new application in their test environment. Everything proceeds well and they decide to install the new application into their production environment. The application

server does not start and issues a ClassCastException exception. The test server and the production server configurations are identical. The Application Server, application, and parameters are the same. This becomes a puzzle. They need to launch this application as the rest of the company has been informed that testing was going well.

They search though the change records in their head to remember what changed. They wish that they had a centralized change record, but it does not exist. Several developers and system administration staff members have access to both the test and production environments. Unfortunately, these people are on vacation.

One of the developers remembers that a shared library is a slightly different version from the test environment to the version in the production environment. After they change that library, everything works fine.

Possible solution and preventive care:
Use the following checklist to prevent this malpractice:

- ✓ Use a centralized change record system and maintain it accurately.
- ✓ Create a strict process to inform every one of the changes.
- ✓ Be aware of the fact that taking a little time to follow this process can prevent huge problems and embarrassment later.

In all stages

11. Communication breakdown

Communication might break down in many different ways. Communication is always two-way and requires reinforcement by delivering the information in multiple forms such as speech and in writing. In project management, you communicate to get things done, pass on and obtain information, reach decisions, achieve joint understanding, and develop relationships. Using these techniques correctly determines the success of a deployment.

You must recognize the following communication barriers:

- Installing products without considering product updates
- Understanding the environment
- Communicating with management
- Considering cultural issues
- Timing deployment
- Coordinating the impending deployment with the affected products and services owners
- Having a back out plan

Who needs to be communicating what to whom, how, and when that should be done is key to having a smooth and productive life cycle for a product, application, or even a business process.

Case study:

A large grocery chain runs their inventory nightly on a zSeries® machine. The company installs a new web application that generates orders to their suppliers based upon the in-store inventory summary for each store.

This new application is written in Java and has been tested on distributed machines, but the application runs in production on the zSeries machine. The application prints a list of warning messages if unusual patterns are noticed in the order or if the application has errors. The error printouts are in the application because this application is so new.

The application is delivered and deployed on the zSeries machine as requested by the development organization. The operations group, which runs the zSeries machine, generally starts programs, stops programs, and performs other system programmer duties. They know what printouts to expect from the z/OS system. They do not know anything about what the application does because the applications only write to disks.

The application developers know that the people in the operations group are called system programmers so they make the assumption that these people write programs, not that they do system administration. Hence, they do not explain how the application runs, nor what to do when the application sees errors or unusual patterns exist.

The new application does not seem to be doing well at ordering new supplies for the stores. However, because the developers have not received printouts, they do not know that there is a problem. When management confronts them, they say that they do not know anything about it as they have not received any printouts. Meanwhile, the system programmers in the operations group are throwing away the extra paper that is now printing out because they do not know what the printouts are for or why they are printing.

Someone started following the life cycle of the application development and deployment discovered the communication breakdown. Because tempers are now hot, the two groups must work not only on communicating but also on putting their defensiveness and old habits aside.

Possible solution and preventive care:

Use the following checklist to prevent communication breakdown:
- ✓ Identify who needs to communicate on what subject and to what extent.
- ✓ Create a checklist to make sure the current information is synchronized.
- ✓ Have a periodic meeting to make sure that everyone understands the process.
- ✓ When someone is out, the backup person must know exactly what is occurring.

Impact of production outages to your business

“Web site down!” can be a nightmare for any e-business vendor. Besides the loss of revenue, the outage hurts the credibility of the vendors and opens doors for competitors. Many e-business vendors rank stability as the most important factor when they choose products. However, most vendors do not have a clear understanding of actual cost of product outage.

The following work sheet estimates the monetary cost that is incurred during product outage or a Web site is down. Other non-monetary costs, such as the loss of customer loyalty and goods, are difficult to quantify and are not discussed here.

For an online-only retailer

Transactions/hour	Average transaction value ($)	Down time (hours)	Cost of the outage ($)
T	V	H	C=T*V*H
100 (Normal season)	$56[1]	1[2]	$5,600
300 (Peak season)	$56	1	$16,800

For a retailer with both online and brick & mortar shops

Transactions/hour	Average transaction value ($)	Percentage of online business	Down time (hours)	Cost of the outage ($)
T	V	P	H	C=T*V*P*H
500 (Normal season)	$56	50%	1	$14,000
1500 (Peak season)	$56	50%	1	$42,000

1. The average transaction value for an American retailer is $56.
2. The typical recovery time from an outage is 45 minutes during a normal business hour. However, the time can be considerably longer if the outage occurs during non-business hours.

The previous table illustrates the cost of a product outage for a typical small-to-medium e-business vendor. Different vendors can plug in their own numbers to calculate the actual cost of their product outage. You can use these numbers to find the balance between the cost and benefit, what kind of service level to provide (24/7, 8 a.m. to 5 p.m.), and how much money should be invested to improve the product availability.

Keep in mind that these costs are the direct losses from a product outage. For example, the losses can come from a loss of revenue during which time the product server is unavailable. There are many other indirect costs, such the loss of good faith, loss of customer loyalty, and an attack from competitors, to mention a few.

For a large online business, the indirect loss can be substantially larger than the direct loss. The negative publicity that is generated by the outage is particularly damaging. However, those losses are not discussed here because producing a precise monetary estimation of those losses can be a daunting task and beyond the scope of this document.

Conclusion

Extensive cost cutting can sometimes cause businesses a loss rather than a savings. Having a test environment that is equivalent to the production environment is fundamental to a successful information technology project. Yet, we often fail to utilize the test environment before propagating the changes into the production environment. In a rapidly moving world, it is difficult to follow a meticulous process and keep the record of the changes. However, it is important to track these changes.

Appendices

Appendix A – Worksheet for the education plan

Product or process	**Education topic**	**Source or class**	**Who attended**	**When attended**	**Was the education a good use of time? 1-10***

* 1 represents the lowest rating while 10 represents the highest rating

Appendix B – Worksheet for tracking the architecture plan

Date of the change	**What was changed?**	**Revised architecture location**	**Who designed the revisions?**	**Who reviewed the revisions?**	**Who approved the revisions?**

Appendix C – Worksheet for keeping track of changes

Date and time of the change	**Hostname**	**Scope of the change (cell, server, application, or operating system)**	**Change made from → to**	**Reason**	**Contact person and phone number**	**Change information is distributed to whom**

Document change history

Check the date in the footer of the document for its version.

April 25, 2006	Revised hyperlinks and activated Table of Content hyperlinks
April 14, 2006	Final revision for publishing
April 10, 2006	Original document