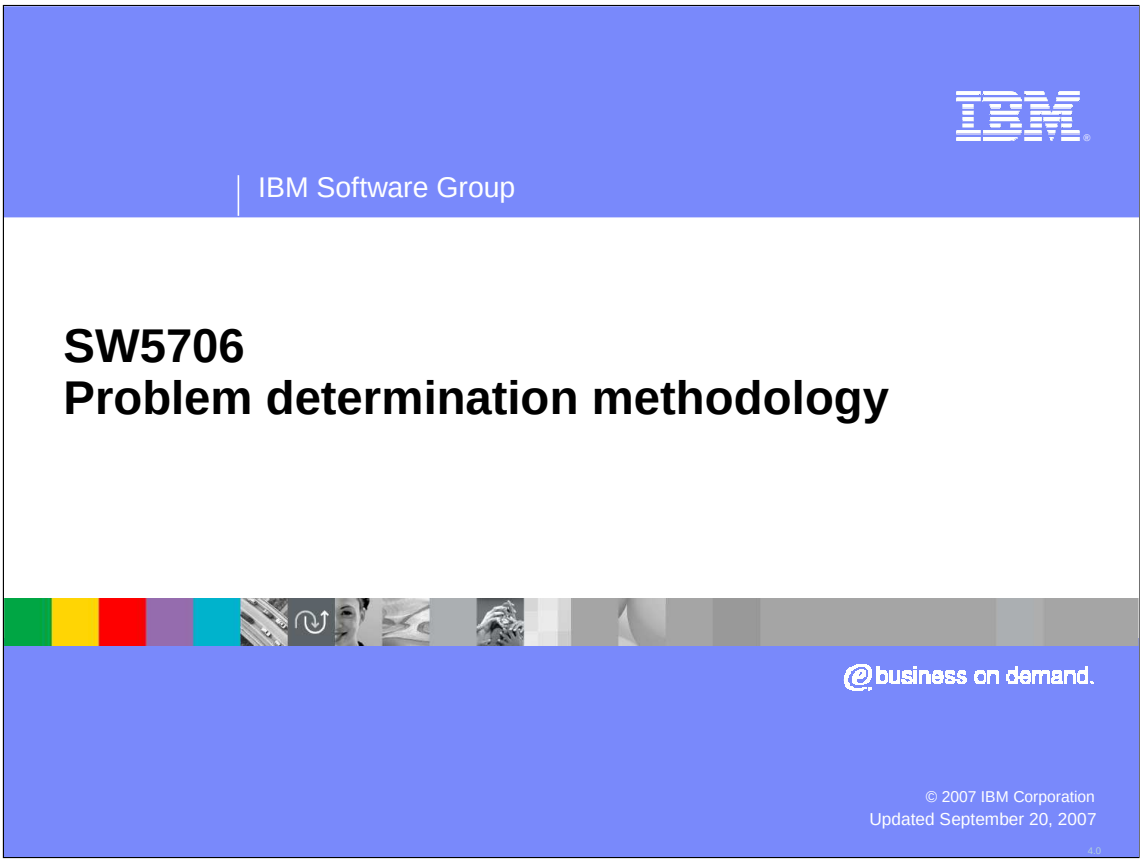


This unit describes the problem determination methodology used in this course.

Unit objectives

After completing this unit, you should be able to:

- Understand a problem
- Devise a plan of action
- Carry out the plan
- Examine the solution

After completing this unit, you should be able to understand a problem, devise a plan of action, carry out the plan, and examine the solution.

Types of problem symptoms

- Here are the common types of symptoms that you might see. Almost every symptom falls into one of these categories:
  - Cannot install or migrate WebSphere Application Server or install an application into WebSphere Application Server.
  - Experience difficulties in WebSphere Application Server system management or configuration.
  - An application or WebSphere Application Server process (for example, an application server, node agent, or deployment manager) is unable to start.
  - An application does not respond to incoming requests.
  - An application produces unexpected results (possibly errors or exceptions).
  - An application cannot connect to an external system or resource.
  - An application performs slowly or its performance degrades over time.

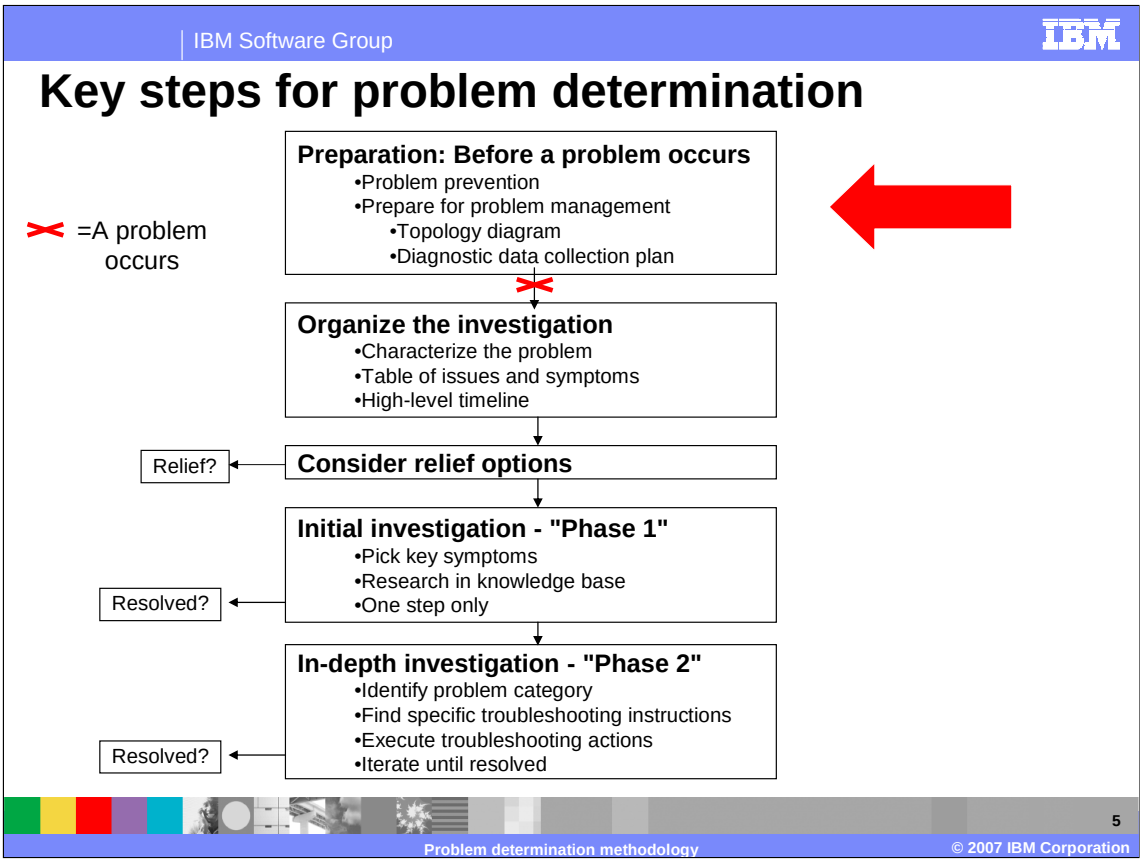
Most problems are due to environment or configuration issues, misunderstandings or miscommunication, and hard-to-diagnose application issues. Typically, less than 10% of reported issues are due to product code defects in WebSphere Application Server or related products. Problem determination is a matter of common sense, thoroughness, and clear communication. Some examples of typical problem symptoms include, but are not limited to: the inability to install or migrate WebSphere Application Server, difficulties in WebSphere Application Server management or configuration, problems starting applications in WebSphere Application Server, and application related issues such as unresponsiveness, unexpected results, external resource management, and poor performance.

General observations about problem determination

- Problem determination is not an exact science.
- Problem determination is not rocket science.
- Problem determination is often a *cooperative* and *iterative* process.
- The biggest obstacle to problem determination is poor communication.
- Not every problem requires the most complex problem determination skills and techniques.

Problem determination is not magic, but a matter of common sense, thoroughness, and clear communication.

Problem determination is often a cooperative and iterative process, dealing with unanticipated problems. Clear communication is commonly the biggest obstacle in proper problem determination. Customer self-service and automation for common problem determination tasks is available.



The job of a troubleshooter does not begin after a problem occurs. There are many ways to prepare the infrastructure, the internal team, and the end-users for a problem. Often, troubleshooters immediately begin long and complex analysis when the most important thing for users may be some form of relief or workaround. Proper preparation, including back-up plans, relief planning, and problem communication mediums are critical to create before the first problem occurs.

Preparation: before a problem occurs

- Good problem determination starts long before anything bad happens
- Implement problem prevention best practices
- Perform monitoring and problem detection
- Keep good system documentation
- Have a diagnostic data collection plan
- Have a relief or recovery plan
- Keep a maintenance plan: scheduled and emergency

Good preparation involves a variety of mechanisms before a problem occurs. First, problem prevention best practices should be implemented. A list of problem prevention best practices is provided in the reference material. Next, perform proper monitoring and problem detection so that problems do not go unnoticed. Maintaining system documentation such as topology and system baselines prepares the troubleshooter for problem analysis, Creating a diagnostic data collection plan ensures faster and more effective analysis once a problem occurs. A relief and recovery plan lays out, in advance, steps to take to restore functionality to users. Finally, an emergency maintenance plan outlines how to update the system quickly in the case of a problem which necessitates a system update.

Monitoring and problem detection (1 of 2)

- Often, problems go undetected for a long time, or only get detected indirectly through some secondary effect. You need tools and a plan to effectively detect problems or any potential anomaly when they emerge.
- Monitoring is a trade-off: You want to detect important events, and yet not adversely impact the normal operation of the system.
- Monitoring is an entire technical area in itself, different from problem determination. This course will only cover a few pointers.

Many problems go undetected for a long time or only get detected indirectly through a secondary effect. Monitoring is a key tool in detecting problems as soon as they occur. It is a trade-off between detecting important events and the overhead of monitoring on the system.

Monitoring and problem detection (2 of 2)

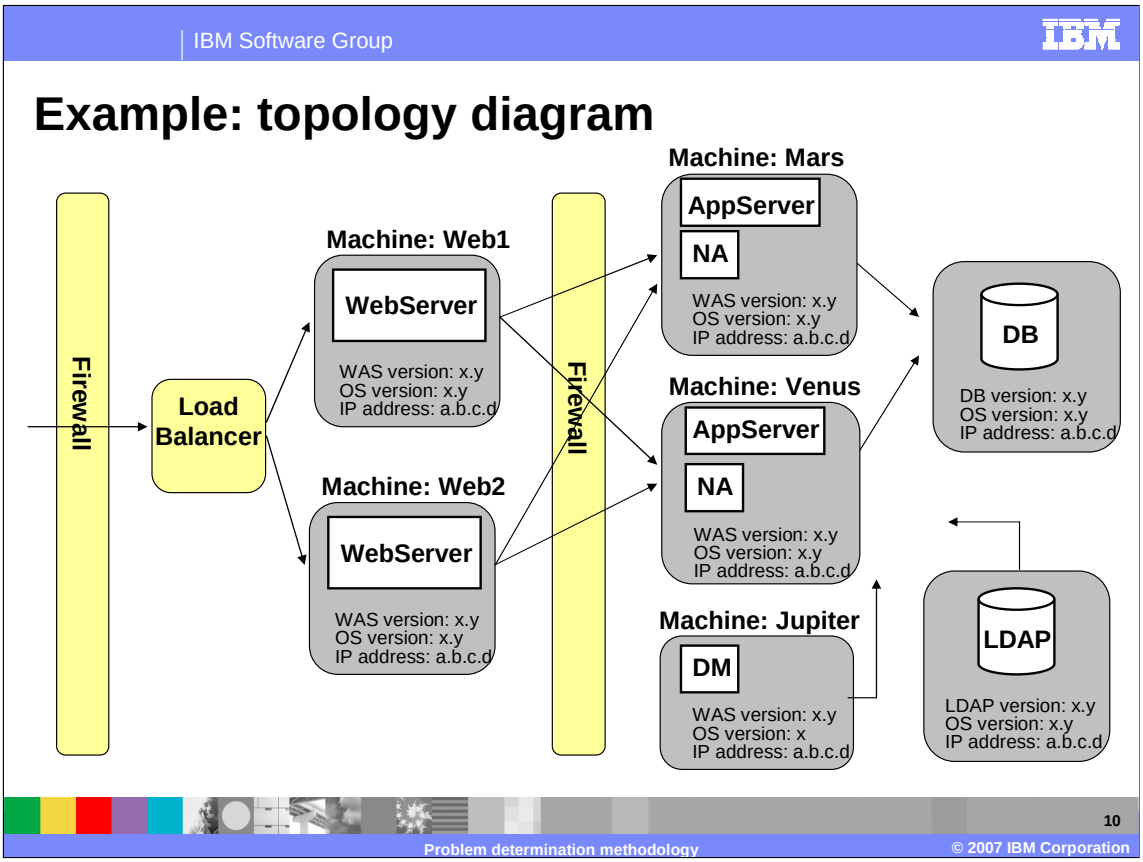
- Passive monitoring
  - At all levels: network, operating system, WebSphere, application, dependent systems (for example: database, directory, and so on).
  - Monitor the main system log files for errors and events.
    - For example, detecting application server restarts
  - Monitors and alerts on key system metrics.
    - OS: memory usage, CPU, and so on
    - WebSphere: default PMI, Performance Advisors
  - A few tools:
    - Event Alerter (subscribes to JMX notifications from WAS)
    - Tivoli tools
    - Many ad-hoc scripts and tools for each installation
- Active monitoring
  - Beyond passive monitoring; the best way to make sure that everything is all right is to periodically "test" the operation of the system.
  - Localized "pinging" (for example, one server, one database connection, and so on).
  - End-to-end "pinging": periodically send an entire "dummy" transaction through the system, verify that it completes.
    - For example, using TMTP, Web-based load programs

There are two forms of monitoring: passive and active monitoring. Passive monitoring watches the system at all levels, including the network, operating system, WebSphere, application, and dependent systems such as databases and directories. It monitors the main system log files for errors, as well as key system metrics such as CPU, memory usage. Active monitoring is more involved than passive monitoring and tests the activity of the system, including end-to-end pinging, sending dummy transactions through the system and verifying results. Active monitoring ensures the functioning status of the system, whereas passive monitoring ensures the overall health of the system and its components.

Topology diagram

- Show all the components and all the main flows in the system
- Useful for
  - Communicating with all parties involved in the problem determination effort
  - Identifying discrepancies between the expected environment and the current reality
  - Identifying points where monitoring or health checking can be done
  - Identifying points where diagnostics data can be collected
- Be specific and detailed
  - Indicate software versions, machine names, and IP addresses if possible
- Identify the main execution flows through this diagram

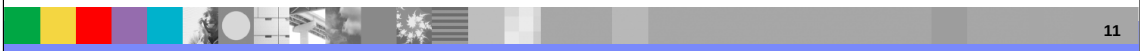
A topology diagram shows all the components and all the main flows in the system. It is useful for communicating with the parties in the problem determination effort, for identifying discrepances in the environment, and for identifying points where monitoring or diagnostics data will help in further problem determination.



This is an example topology diagram showing the key elements in the system, with detailed version and identification information. The diagram should show the relationships between components and any logical or physical delineations in the system. A separate network topology may also be useful.

Establish baselines

- Much of what you will do for problem determination is to observe various aspects of the behavior of the system and try to determine if these are normal or if they represent a symptom of a possible problem.
- Therefore, you must know before a problem occurs, what a "normal" system looks like.
- Some example criteria:
  - What are the typical values for common system metrics (CPU usage, memory size, PMI stats, and so on)?
    - These are often the same metrics that you should be monitoring for problem detection
  - What is a typical load, typical response time for various operations?
  - What should you normally see in the logs during normal operation of the system?



Establishing baselines expectations and metrics prepares you for what are anomolies and what are expected fluctuation or benign errors. Some example criteria are CPU, memory, PMI statistics, what the expected load and response times are, and what are the expected log entries during normal operation.

Diagnostic data collection (1 of 2)

- Decide in advance which diagnostic elements should be captured by default on the first occurrence of any problem
  - This is a trade-off: you want to capture as much as possible, without too much impact to the normal operation of the system
  - Some things to consider:
    - Default WebSphere logs
    - WebSphere First Failure Data Capture (FFDC) facility
    - HTTP access logs (what are default levels?)
    - Enable core dumps, Java dumps, and so on
    - Consider enabling verbose garbage collection
    - Consider a minimal level of monitoring of operating system resources
    - Consider a minimal level of monitoring of network health
    - Consider application logging

Decide in advance which diagnostic elements should be enabled and the respective levels of these diagnostics that may be required for general problem occurrences. This is a trade-off between system performance and hard drive space on one end, and the granularity of debug information in the case of a problem on the other end. Verbose garbage collection is one parameter which has a very minimal impact on performance but provides a benefit for problem determination.

Diagnostic data collection (2 of 2)

- Identify active diagnostic actions to take for the most likely problems. For example:
  - Trigger a Java core or core
  - Attempt pings and application checks
  - Collect specific PMI statistics
- Make arrangements so that all the pre-determined diagnostic elements get reliably captured
  - Clearly identify where they are
  - Established procedures to collect them and practice
  - Synchronize the clocks between all machines to facilitate analysis
  - Manage the logs and all other diagnostics artifacts to avoid surprises during normal operation
  - Make sure you have enough disk space available for diagnostic artifacts
- Don't forget the human element:
  - Who is in charge, when there is a production problem, of executing the diagnostic data collection plan and take recovery actions?
  - What groups need to be informed or coordinated with?

Create plans for active diagnostic actions and diagnostic capture in the event of a problem. These plans should include common actions such as triggering java cores, pinging applications, and collecting PMI statistics. These actions are performed when you suspect that there is a problem; therefore, you should consider invasive actions in this case.

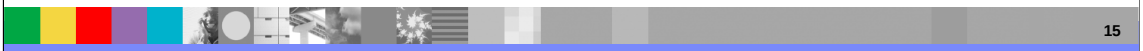
Relief or recovery plan (1 of 2)

- When a problem occurs, one of the main priorities, independent of any investigation, should be to restore function to the end-users.
- The relief or recovery plan lays out, in advance, the steps that you will undertake to accomplish this, without knowing in advance exactly what problem will occur.
- It is impossible to predict all possible situations in advance, but on the other hand, much confusion, wasted time, and even greater disasters occur when relief actions are attempted "in the heat of the moment". Preparation pays off.

The relief or recovery plan lays out, in advance, the steps that you will underatke to accomplish restoring function to the end-user.

Relief or recovery plan (2 of 2)

- Try to predict the most common types of problems, based on knowledge of the system topology and flows.
  - For example, loss of one machine, loss of database connectivity, and so on.
- Identify different regions of the system that can be isolated or restarted independently (for example, one machine, one cluster, and so on).
- Have clear and documented processes for who will decide what to do and who will do it, and have criteria for making such decisions.
- Practice the most common relief actions in advance, to ensure that you know how to execute them, and that they have no unexpected side-effects.



The relief or recovery plan should include actions on the most common types of problems based on the knowledge of the system topology and flows. Identify different regions that can be isolated and restarted independently, and as always, create a clear communication pattern to designate who should do what in failure scenarios. As with all drills, practice makes perfect.

Maintenance plan

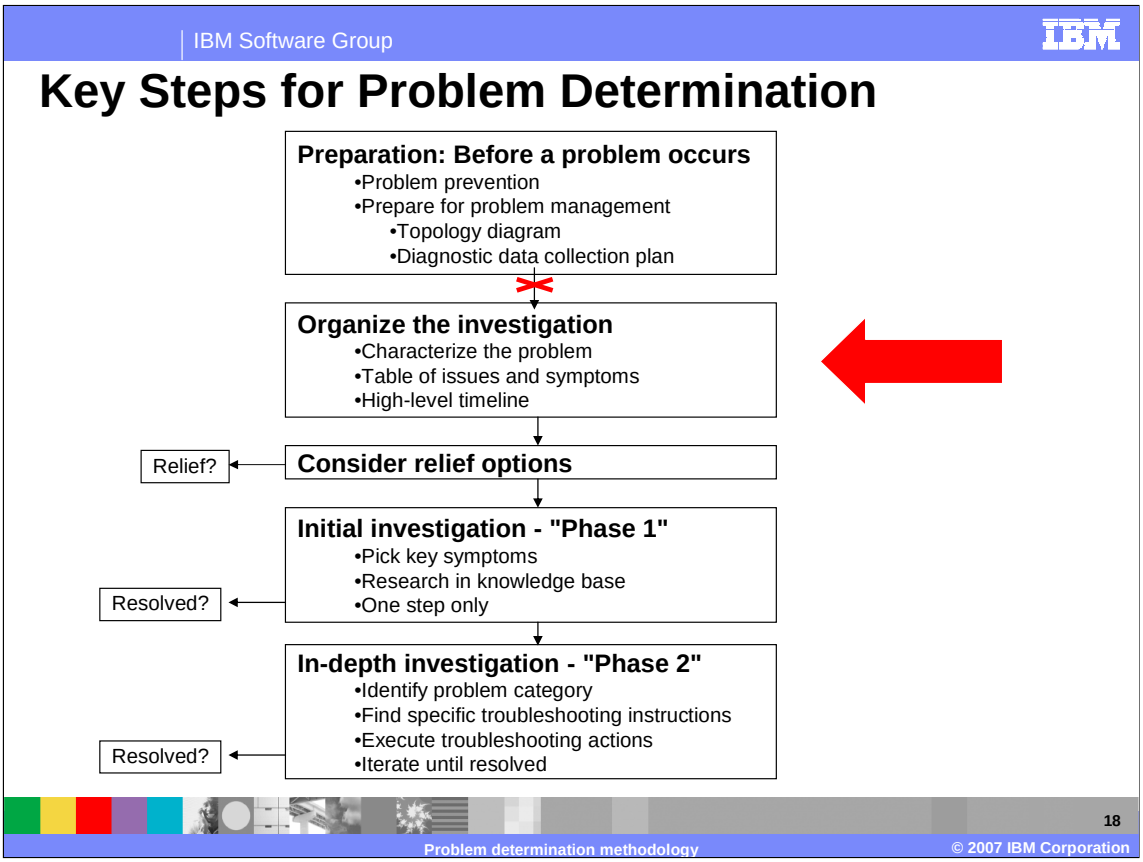
- Applying regular maintenance is one of the key factors to reduce the probability and impact of problems. The maintenance plan establishes how you will do this on a regular basis.
- In addition to regular scheduled maintainance, you may also need to perform emergency changes or maintenance to the system, in response to a newly-diagnosed problem. The emergency maintenance plan outlines how to do this safely and effectively.
- Maintenance is at all levels:
  - OS, and each of the products involved in the system.
- Keep track of current maintenance levels on the topology diagram.
  - Have processes in place for verifying the maintenance levels regularly and after each incident.

Execution of a regular maintenance plan reduces the probability and impact of problems. A maintenance plan should outline regular maintenance, emergency maintenance, at all levels, from the Operating System to the products. All maintenance should be logged for auditing and problem determination.

Maintenance strategy: a word about Fix Packs

- IBM Support often recommends upgrading to the latest available Fix Pack during the course of an investigation.
  - http://www.ibm.com/support/docview.wss?rs=180&uid=swg27004980
- There is often a lot of concern about the risk of destabilizing the system by installing a Fix Pack.
- It is a difficult trade-off:
  - The risk can never be completely eliminated.
  - On the other hand:
    - There is also considerable risk in using too many individual fixes; no one can possibly test all the possible interactions between all individual fixes.
    - Because of the complexity of the system and the difficulty of reproducing problems and gathering diagnostics, it is not always practical to determine exactly which APAR resolved a particular situation.
- Overall, the best bet is to have a strong maintenance strategy.
  - Regular (small) updates.
  - Reasonable re-testing before each update.

IBM Support often recommends upgrading to the latest available Fix Pack as the first step during problem determination. The risk of upgrading is a difficult trade-off; however, in general, risk can never be eliminated, and there is considerable risk in using too many individual fixes.



Organizing the investigation is the next key step in problem determination.

Characterize the problem (1 of 2)

- Take the time to carefully understand the problem and its context. Listen and ask questions.
  - In many cases, this is all it takes to solve simple problems.
  - For complex problems, failure to do so often results in considerable delays
- Develop a clear and specific description of **what** happened, error messages, observed abnormal behavior, and so on.
  - How would you recognize the same problem if it happens again?
  - What exactly would you expect to be different once the problem is solved?
  - Beware of vague terms like *crash* or *fails.*
    - A *crash* is not the same as a *hang*, is not the same as an *exit*, and so on.
  - Be alert for possibly-unrelated symptoms, and the possibility that there may be several independent problems happening at once.
- **Where** exactly did the problem occur, what machine, what server, and so on

First, take the time to carefully understand the problem and its context so that it can be characterized. Next, developer a clear and specific description of what happened, including error messaged and observed abnormal behavior. Be wary of vague terms like crash, hang, and failure. Finally, where did the problem occur, on what machine, and under what conditions.

Characterize the problem (2 of 2)

- **When** exactly did the problem occur?
  - Did it happen only once, or does it happen repeatedly?
  - If the problem happens repeatedly, characterize the circumstances
    - Apparently random times? What frequency?
    - Every time you do X, or every time some other external event happens?
- Consider **why** this problem occurred here and now, and has not occurred before.
  - Is this the first time that you attempted something new (for example, first product installation)?
  - If not, has anything at all changed in the environment (configuration changes in the failing system or any other system that is also present in the environment)?
  - Does this problem occur in all similar environments (multiple production systems) or not?

Next, characterize the problem with when exactly the problem occurred. Did it happen once or did it happen repeatedly. Finally, consider why the problem occurred, and why it has not occurred before. Is this the first time attempting a new function? Has something changed in the environment recently? Does this problem occur in similar environments?

Best practice: table of issues and symptoms

- List all top-level issues reported by users of the system, and all low-level symptoms observed during the investigation
- Useful for:
  - Organizing the investigation: find all symptoms, always identify what to check on next
  - Tracking progress
  - Making sure nothing is overlooked
  - Managing complex situations, with many unrelated high-level issues and many unrelated symptoms
- Prioritize and cluster entries to reflect the current state of the investigation
  - Constantly update and revise as the investigation progresses

Developing a table of issues and symptoms is a best practice to help in organizing the investigation and prioritizing the current state of the investigation.

Example: table of issues and symptoms

Cluster 1: Slow response + crash

Groupings		Num	Status/Priority	Symptom or Issue	Where Observed	Plan of Action / Disposition
		#1	Open/High	AppServer unresponsive to HTTP requests	PMR 11111 • Incident 11/24	Awaiting result of investigation on #8
		#9	Open/Medium	Large number of httpd processes • Observed during #1	PMR 11111	Defer
		#8	Open/High	"CONM6026W: Timed out waiting for a connection from DataSource" • Observed during #1	PMR 11111	Enable connection pool diags
		#5	Open/Medium	Application error: "Cannot validate credit card"	PMR 11111	Awaiting feedback from application developer
		#2	Open/High	AppServer crashes • Happens sometimes after #1, but not always	PMR 11111 PMR 22222	Awaiting result of investigation on #6
		#6	Open/High	verboseGC reports allocation failure for large (2Meg) object • Last entry in log before #2	PMR 22222	Capture heapdump
		#7	Open/Low	verboseGC reports "Mark Stack Overflow" • Sometimes in log before #2	PMR 22222	Doc advises probably benign - defer

Cluster 2: Unrelated issues

Groupings		Num	Status/Priority	Symptom or Issue	Where Observed	Plan of Action / Disposition
		#3	Closed	No javacore produced in response to kill-3 • During PD work for #1	PMR 22222 PMR 33333	Solved by fixing OS privileges
		#4	Open/Low	SECJ0049E: Configuration error encountered while starting the server	PMR 44444	Defer – assume independent problem

One example of a table of issues and symptoms includes clustering of issues, the priority and description of each issue, and the plan of action and diagnosis.

High-level timeline

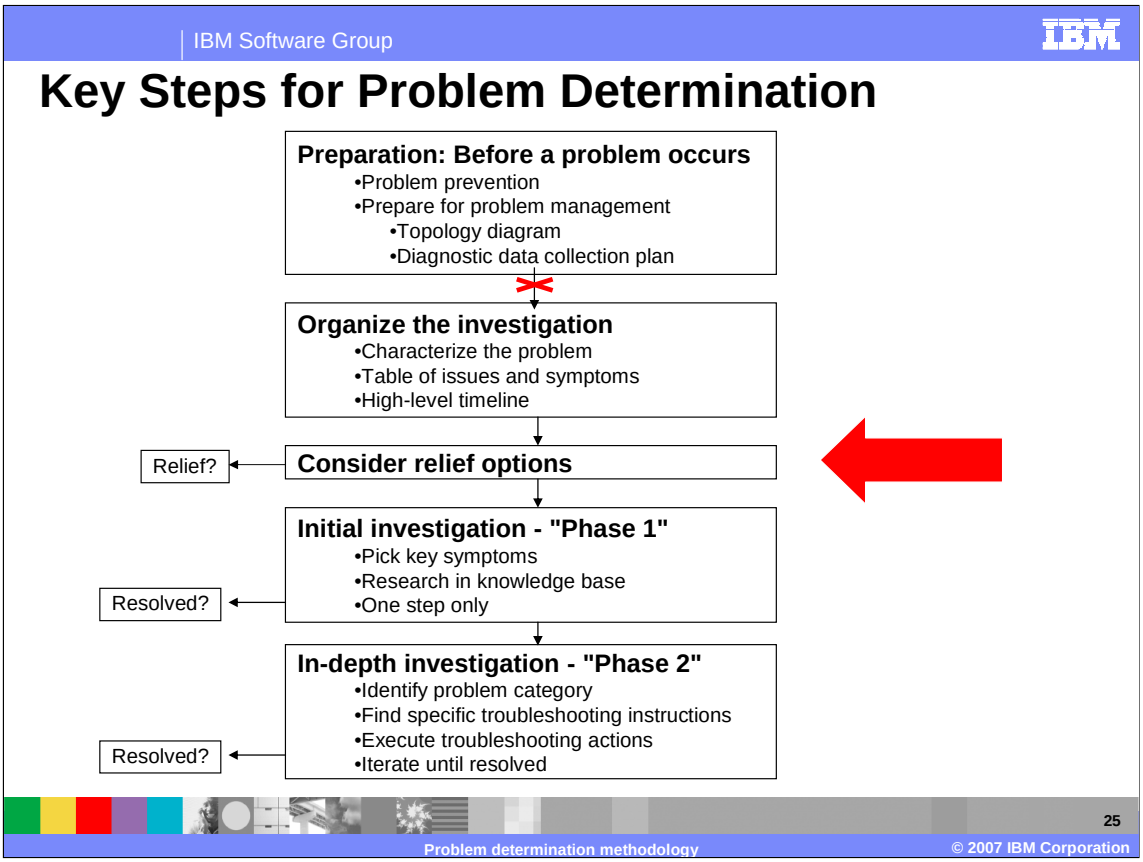
- List all major events when they occur
  - Incidents (new occurrences of the same or other problem)
  - Data collection actions and experiments
  - Relief and remedy actions
  - Other maintenance
- Useful for
  - Helping to identify patterns, cause and effect, and so on
  - Avoiding confusion over time about what really happened
  - Facilitating communication between all parties (Customer, IBM Support, and so on)
  - Keep track of all available diagnostics artifacts and what they correspond to, along with precise timestamps to look for in logs
  - Reinforcing change control policies
    - If too many things change at once, it may never be possible to understand the problem

A high level timeline is a best practice to help in identifying patterns, cause and effect, and avoiding confusion over time. This list may include all major events when they occur, the data collection actions and experiments, and the relief and remedy actions.

Example: high-level timeline

Timestamp	Machine	Event / Action	Artifacts
11/24 6:05 pm	Venus	AppServer unresponsive	
11/24 6:16 pm	Venus	AppServer crash -> restart	
11/25 5:31 pm	Mars	AppServer unresponsive (no crash)	
11/25 5:32 pm	Mars	Attempt javacore (kill -3) -> no result	
11/25 8:30 pm	Mars, Venus	Fix OS directory privileges	
11/25 9:00 pm	Mars	Enable CM trace (no restart)	
11/26 12:00 am	Mars	Increase Conn Pool size to 20	
11/26 12:00 am	Mars	Scheduled restart	
11/26 10:03 am	Mars	AppServer unresponsive	
11/26 10:04 am	Mars	Collect javacore (kill -3)	Javacore1.txt
11/26 10:06 am	Mars	Collect javacore (kill -3)	Javacore2.txt
11/26 10:10 am	Mars	AppServer crash -> restart	
11:26 10:10 am	Mars	Collect CM trace	Trace1.log
11:26 1:15 pm	Venus	AppServer unresponsive	

An example high-level timeline shows the timestamp, where it occurred, the events and actions taken, and the artifacts that each event produced such as logs and traces.

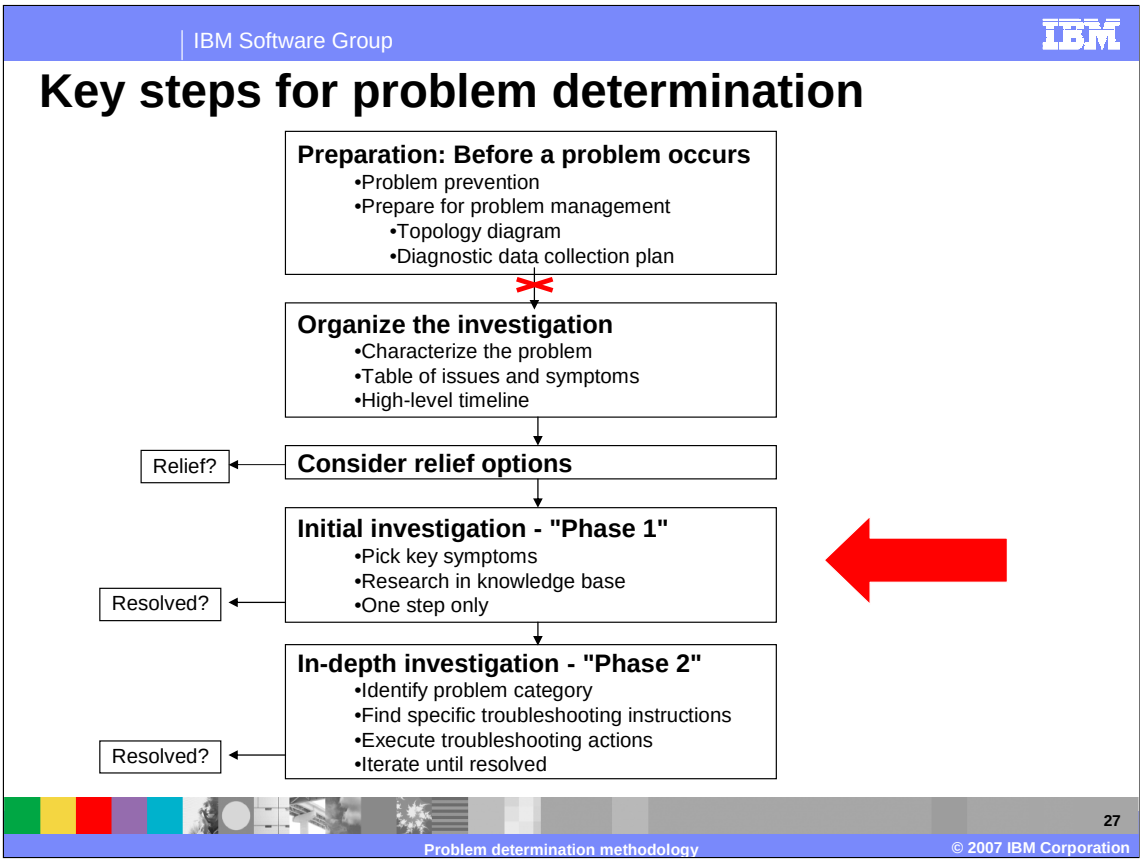


The next key step in problem determination is to consider relief options.

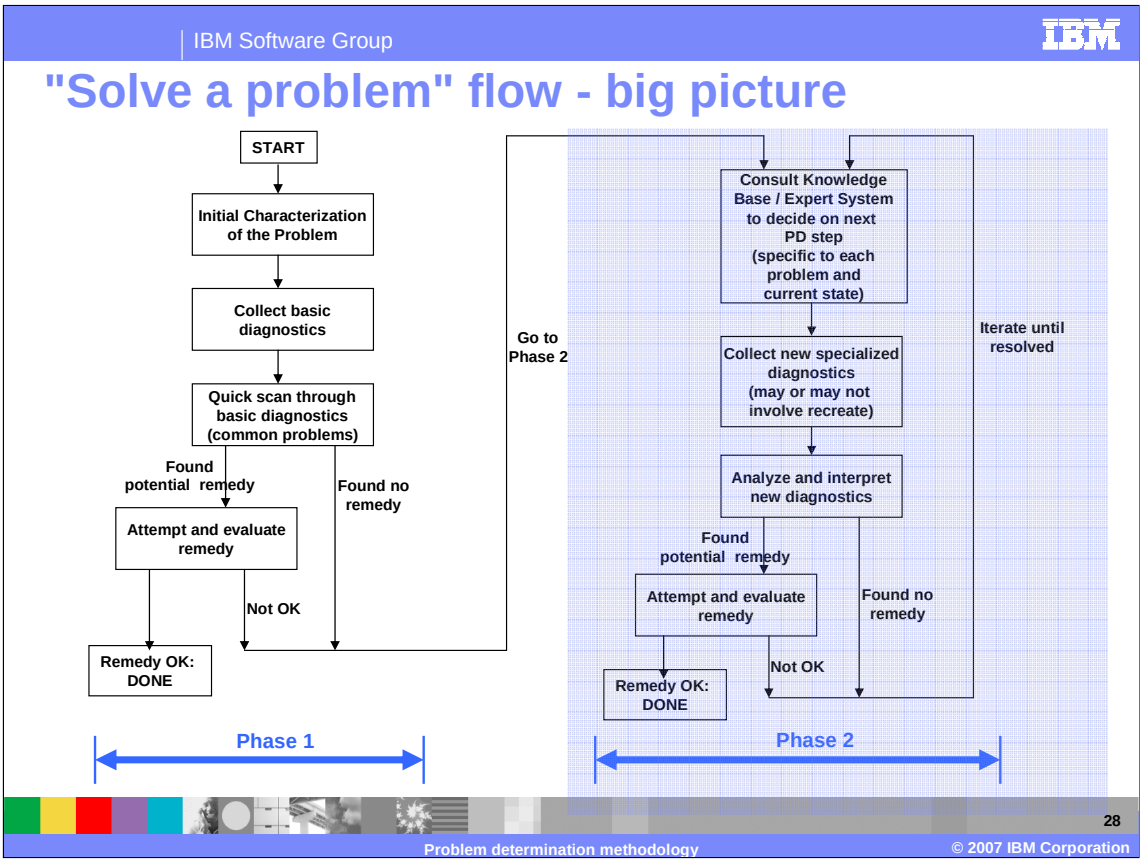
Relief options

- If the problem occurs in production or other critical-availability system, you may need to provide relief long before you fully resolve the problem.
- Consider relief timeline:
  - How long do you have to investigate, before you need to provide some relief?
  - Can you keep the system in its failed state, in case there is some additional information that you want to collect later?
- Identify the relief actions:
  - Often, restart one or more components:
    - But beware of chain reaction effects of stopping and starting some components in a live system.
  - Sometimes, change the usage characteristics of the application:
    - Reduce the load.
    - Avoid some “dangerous” operations.
- Re-evaluate relief options at every step through the investigation.

If the problem occurs in production or another critical-availability system, you may need to provide relief before resolving the problem. Consider the relief timeline, identify the relief actions, and continuously re-evaluate the relief options at every step through the investigation.



The next key step for problem determination is the initial investiation.



Solving a problem can be broken into two logical phases. Phase 1 represents easy, generic steps for each problem, and phase 2 gets into more depth and detail for a specific problem.

Phase 1: initial investigation (1 of 2)

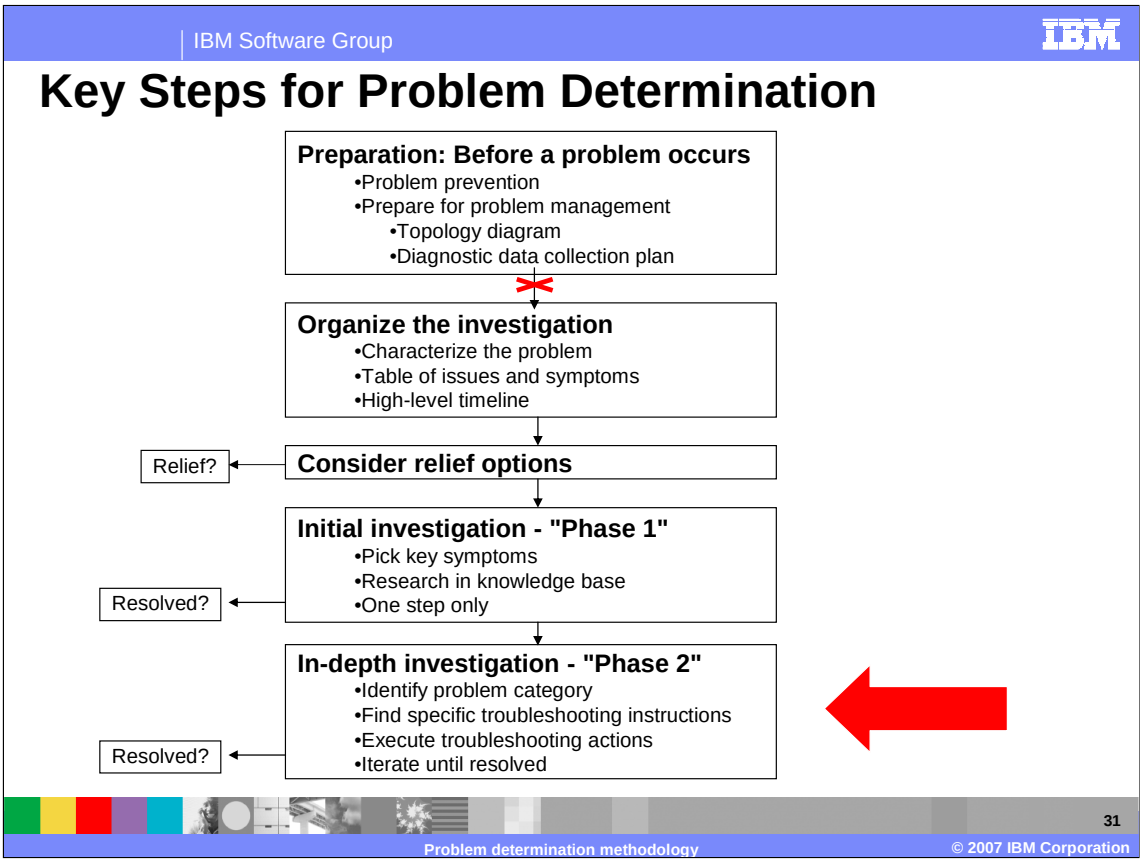
- The goal at this stage is to look for already known problems, and to get a starting point for further, deeper investigation if necessary.
- Make an inventory of all pertinent anomalies and potential symptoms:
  - Errors, warnings, exceptions, out-of-range statistics, any other unusual behavior (scan (grep) through available logs).
  - Ideally, you should check everything, but this research might be guided by understanding the flows in the topology diagram.
  - Integrate into the table of issues and symptoms.
  - Assess and prioritize symptoms:
    - Not a perfect science; hard to tell which symptom is a cause, and which symptom is a consequence of the original problem.
    - Use the topology diagram and baselines prepared earlier.

Phase 1 – the initial investigation – looks at known problems and makes an inventory of all pertinent anomalies and potential symptoms, such as errors, warnings, exceptions, or any other unusual behavior. This information can be integrated into the table of issues and symptoms for use in the preparation for organizing Phase 2.

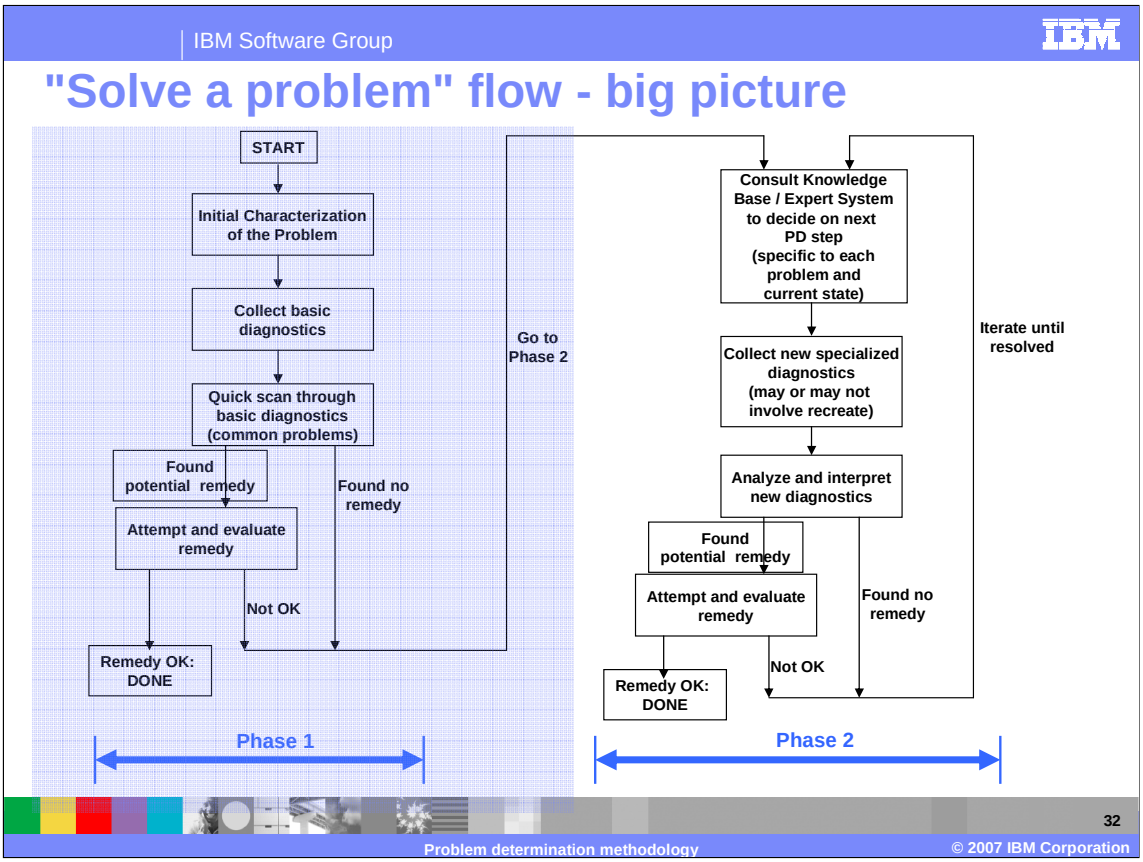
Phase 1: initial investigation (2 of 2)

- Build a low-level, detailed timeline of events for one incident, to clarify what may have caused what
  - Include pertinent but normal events, in addition to anomalies and symptoms
  - Keep track of multiple components in parallel and attempt to correlate
- Research the top symptoms in the WebSphere Knowledge Base
  - Search for matching Technotes
  - Search the Information Center
  - This will be covered in the next unit

Building a low-level timeline of the events for one incident may clarify what could have caused the issues discovered in the previous phase. Use the WebSphere Knowledge Base to search for the symptoms and issues and apply and recommended fixes or solutions.



The next key step in problem determination is the in-depth investigation.



If phase 1 did not uncover the issue, phase 2 will be the in-depth analysis of the data gathered in phase 1.

Phase 2: in-depth investigation (1 of 2)

- The initial research did not find a solution, so you must undertake a more in-depth investigation, with a specific set of methods and tools for each specific problem.
- There are several sources of information to start with:
  - Troubleshooting section in the InfoCenter
  - Problem determination IBM Redbook: http://www.redbooks.ibm.com/abstracts/sg246798.html?Open
  - MustGather documents on the WebSphere Application Server Support Web site:
    - Define initial set of diagnostics to focus on (either for IBM Support or for internal investigation).
    - From there, search for other Technotes with troubleshooting instructions, tools, and so on.
    - The Troubleshooting Guide on the support Web site also provides a good starting point for finding relevant documents.

There are a variety of methods and tools for researching a more specific problem. MustGather documents indicate all the steps necessary to gather data on a particular type of problem. The Information Center also provides product information, and there is an IBM Redbook describing problem determination next-steps.

Phase 2: in-depth investigation (2 of 2)

- The WebSphere Knowledge Base provides the starting point for the investigation process. What happens next depends on what you find.
- Two main investigation techniques:
  - Analysis Approach
    - Exploit available diagnostic data in increasing detail until solution is found
  - Isolation Approach
    - Isolate and simplify the problem to the smallest, simplest possible sub-unit of the original system, to reduce the amount of data to analyze
- In practice, these two approaches are often pursued in parallel, and feed into one another
- Consider if you might attempt to reproduce the problem in a test environment

The WebSphere Knowledge Base provides the starting point for the investigation. The two main invesigation techniques are the analysis approach, which exploits available diagnostic data in increasing detail until the solution is found, and the isolation approach, which isolates and simplifies the problem to its smallest unit to reduce the amount of data to analyze. Consider trying to reproduce the problem in a test environment to help in both approaches.

Analysis approach

- Exploit available diagnostic data in increasing detail until solution is found
- Heavily driven by knowledge of the meaning of each diagnostic, and of the internal operation of each component of the system
- Many specialized analysis tools available
- In practice, you often need to iterate to collect more specialized data
  - For example, enable special tracing, collect heap dump, and so on
- The low-level, detailed timeline of events created earlier, is often a good tool to help focus that analysis
  - Keep adding more detail in the timeline

The analysis approach is heavily driven by the knowledge of the meaning of each diagnostic, using specialized analysis tools and the low-level timeline created earlier to increase the available diagnostic data that is relevant to the problem. This approach focuses on the details until the solution is found.

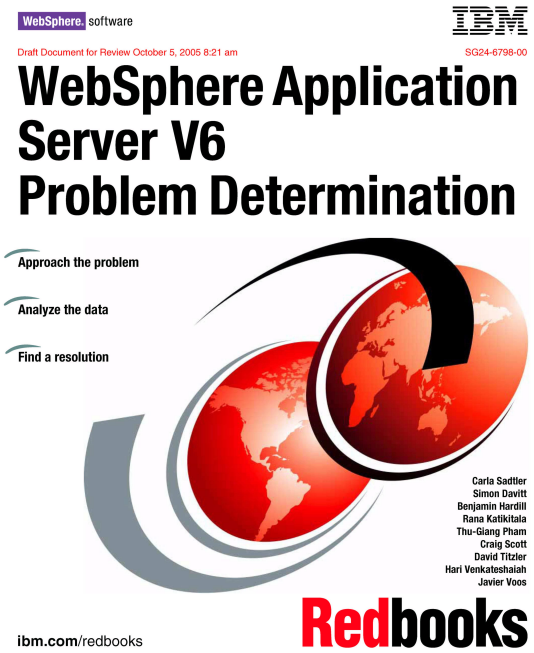
Isolation approach

- Isolate and simplify the problem to the smallest, simplest possible sub-unit of the original system, to reduce the amount of data to analyze
- Heavily driven by knowledge of the structure of the system and its flows, but can often treat each individual component as a black box
  - Use the topology diagram
- Monitor flows at various control points in the topology
- Inject “pings” at various control points to observe results
- Eliminate one component at a time from the topology, rerun test to see if problem re-occurs

The isolation approach treats each component as a black box instead of focusing on the details. This apparoch is driven by knowledge of the structure of the system and its flows, and monitors flows at various control points to decide how to isolate the problematic component(s) for further analysis.

For more information

IBM Redbook
- SG24-6798-00
- www.redbooks.ibm.com



Please read the SG24-6798-00 Redbook on Problem Determination for more useful guidelines.

Unit summary

Having completed this unit, you should be able to:

- Understand a problem
- Devise a plan of action
- Carry out the plan
- Examine the solution

Having completed this unit, you should be able to understand a problem, devise a plan of action, carry out the plan, and examine the solution.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject= Feedback about SW5706G03_Methodology.ppt

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

Approach IBM Perform WebSphere

Java, JMX, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.