Troubleshooting Theory

# The importance of having a methodology.

Having a troubleshooting methodology is important, especially in the IT arena.

Due to the complexity of modern computing systems, a wise technician will have and follow a troubleshooting methodology. A formal methodology gives the technician a starting place and a logical sequence of steps to follow. If you do not have a methodology in place, you are much more likely to waste time and effort and create frustration—not only for yourself but also for your end user.

# Six-step troubleshooting methodology.

1. identify the problem
2. establish a theory of probable cause
3. test that theory of probable cause
4. develop an action plan and implement that plan
5. verify complete system functionality
6. document the process

## The six steps of troubleshooting.

### 1. Identify the problem.

The first thing that you need to be aware of when troubleshooting a problem is that the symptoms are not the problem. When troubleshooting, it is critical that you actually identify the underlying problem—what’s actually causing the symptoms to manifest themselves. To do that, you should question the user. Ask detailed questions about when the symptoms occurred and why they may have occurred. If the user can re-create the issue for you, this can be extremely helpful. Determine what, if anything, has changed, which may have caused the problem. Importantly, before you take any action, make a backup copy of the system, so that you can preserve everything as it is.

#### Highlights:

* Question the user.
* Remember that the symptoms are not the underlying problem.
* Determine what has changed.
* Make a backup of the system before moving on.

### 2. Establish a theory of probable cause.

Once you have identified the base problem that is causing the symptoms, you will establish a list of probable causes. Once you have a list of all of the probable causes, use your technical knowledge to prioritize that list. Your list should have the probable causes listed from most likely to least likely. Incidentally, be sure to question the obvious. For instance, if the symptom is a power situation at the workstation, is the power cord plugged in and, if it is plugged in, is the outlet actually getting power? Additionally, if no probable cause can be determined, you will need to escalate the problem to a higher level.

#### Highlights:

* Make a list of probable causes.
* Using your knowledge, prioritize the list.
* Question the obvious (i.e., if the symptom is a power issue at the workstation, first check to make sure that the power cord is plugged in).
* Escalate to a higher level if a probable cause cannot be determined.

### 3. Test probable cause theory to determine actual cause.

Once you have established your theory of probable cause, you should take a moment to consider whether or not you can troubleshoot the issue on your own, or if escalating it to a higher authority is called for. If it falls within your capabilities, you will need to test your theory to determine if it is, indeed, the actual cause. Your theory was created from the most likely probable cause, so you need to determine how best to test it. If your theory is confirmed, you will move on to the next step. If the theory is disproved, you will need to go back to step two or step one, as needed, and work your way through the troubleshooting methodology.

#### Highlights:

* Create your theory from the most likely probable cause.
* If the theory is confirmed, move on to the next step.
* If the theory is not confirmed, go back to step two or step one (if needed).

### 4. Establish an action plan and execute the plan.

Once you have determined the actual cause by testing your probable cause, you will need to establish an action plan and then execute that plan. Simple problems will probably only need simple plans. However, if it is a complex problem, you may need to write out the plan so that you can be sure to execute it correctly. This is another opportunity to escalate the problem to a more senior level if necessary.

#### Highlights:

* Simple problems probably just need simple plans.
* Complex problems may need written out action plans.
* Escalate to a higher level if required.

### 5. Verify full system functionality.

After you have executed your plan, you will need to verify that the system is fully functional. If everything works—that’s great. Based on your findings and the issue, you may find that you have the opportunity to implement preventative measures so that the problem does not occur again. If full system functionality has not occurred, you will need to go back to step one and continue to work through the troubleshooting methodology.

#### Highlights:

* If everything works, great! If applicable, use your findings to implement preventative measures.
* If not everything works, go back to step one.

### 6. Document the process.

Once everything is fully functional, documenting the process becomes important. This is where you document findings, actions, and outcomes. When the problem occurs again, there will be information available to walk someone through the means of troubleshooting and resolving the issue.

This documentation also captures a history of equipment and users so that perpetual issues become known and recorded. An important aspect of this is that both positive and negative outcomes should be documented. This can save time during future troubleshooting and prevent others from taking the same missteps you may have taken.

#### Highlights:

* Capture your findings, actions, and outcomes.
* Issues that need to be troubleshot may occur again.
* Documentation provides a history of equipment and users so that problem issues are known.

# What was covered.

## The importance of having a methodology.

A methodology gives a systematic approach to solving IT problems. It reduces wasted time and frustration for both the technician and the end user.

## Six-step troubleshooting methodology.

Identify the problem; establish theory of probable cause; test the theory; establish a plan of action and implement it; verify system functionality; and document everything.

# roubleshooting 101

As with many software nowadays, XenApp requires minimal configuration and installation decisions, and an experienced administrator can configure an infrastructure in a matter of hours.

Particularly because the installation is a simple process, it is the troubleshooting that sometimes becomes difficult.

It is important to note that a solid grasp of XenApp components, interaction, and workflow is needed before performing troubleshooting.

Most times troubleshooting can be easy, either the solution is straightforward, perhaps because the administrator has experienced this problem in the past, or a simple internet search for the particular error message will yield a Citrix knowledge-based article or blog post for that particular problem.

In all other cases, troubleshooting needs to be performed in an organized fashion so the solution is reached in the shortest amount of time possible since many times the problem could involve downtime for a large number of users.

Although seemingly unimportant, one of the most important aspects of troubleshooting is producing a comprehensible problem statement:

* How is the problem manifesting itself?
* Who is facing the issue?
* When did the issue start?

Without clear answers to these questions, an ambiguous problem can undermine efforts for a solution.

Consider the fact that most of the times an issue is generally logged by a service desk or call center (first line of support), who might escalate to a desktop support team (second line of support), and who will in turn escalate to a Citrix team (third line of support).

If any piece of information is misunderstood by the analyst logging the incident, this in turn can be propagated to the Citrix team with the information being completely irrelevant in the troubleshooting process or even incorrect.

Consider the following scenario: a user working in the [finance](https://subscription.packtpub.com/search?released=Available&concept=Financial%20Technology) department calls the helpdesk and complains that an accounting application stopped working in Citrix. The application was working fine last week. The help desk agent performs a series of basic troubleshooting steps and escalates the problem to the next line of support without requesting additional information.

Consider the following questions:

* How many users are affected? Has the application stopped working for other users?
* What is the expected behavior of the application?
* Are you in the same location as last week or a new office?
* Is the application being used by a small or large number of users?
* Can the issue be reproduced on a different machine or in a different office?

While each question in itself might not directly lead to a solution, it can narrow down the problem considerably.

For instance, a positive answer to the first question might indicate, this is a server or network issue as it affects multiple users.

A positive answer to the third question might indicate, this is a network error; the next logical step would be to check whether there are any networking restrictions applied to subnets or IP addresses in the current location.

The fifth question is meant to check whether the issue is specific to a user, machine, or location.

# Breaking down problems

When troubleshooting difficult cases, after making sure you have understood the problem (information provided is correct and relevant), one must ensure a systematic approach to problem solving.

One strategy that can be used is divide and conquer where you break down a problem into individual, easily solvable problems.

Considering the previous example where a user calls the helpdesk (see the previous case), one way of breaking down the problem is testing each sub-system individually, for example:

* Are the Citrix servers online and healthy? Check the monitoring systems.
* Is the network link reliable? Run a continuous ping and check whether websites load correctly.
* Is the problem easy to reproduce on any machine or does the problem follow the user?

For instance, in the case of XenApp 7.5/7.6, the following components can be considered:

* Server/desktop operating system machines and virtual delivery agents
* Delivery controller
* StoreFront
* Citrix receiver
* NetScaler Gateway

Going back to our example, one or more components can be causing a problem. For instance, there might be a problem with the **Virtual Delivery Agent** (**VDA**) on the server/servers hosting the [finance](https://subscription.packtpub.com/search?released=Available&concept=Financial%20Technology) application. This prevents the controller from being able to use the broker agent part of the VDA to communicate with the server.

Another possibility is that the issue is related to authentication. The StoreFront or the NetScaler Gateway (if the user is outside the corporate network) might have problems authenticating users to Site resources.

It is important to quickly rule out as many components as possible. For instance, we could quickly test if the Citrix web page is accessible internally (where only the StoreFront component is used) and externally (where we might be reaching a NetScaler Gateway first). If the webpage is accessible internally but not externally, we would need to focus our attention on the NetScaler Gateway.

Alternatively, if, in both scenarios, the webpage does not load, we might focus our attention on the actual servers and/or delivery controllers.

Let’s take another example: several users complain that during the day, applications published in XenApp start to become slow every morning.

The users mention that the slowness has been happening for some time, but it has only started to impact them recently.

Consider the following questions:

* How long has the initial slowness been observed (several weeks or months)?
* Around what hour is the impact noticeable?
* How long is the impact—several hours or the entire day?
* How often does the problem occur—on a daily basis or only on specific days?

Answers to the these questions can be tremendously important when dealing with performance-related issues. For example, it is important to establish whether the performance is affected during specific hours/days (help to isolate whether a scheduled operation is causing the issue) and whether it is consistent (for example, happens every day of the week or happens only on specific dates/days).

Further breaking down the problem could consist of:

* Determining whether there is any correlation between systems tasks (antivirus, backup, web filtering, and so on) and the start of the slowness
* Determining whether the impacted application(s) can be tied to a group of servers, users, or user locations
* Analyzing past monitoring data for any negative performance trends

*Use NetScaler Insight Center to collect information about traffic, performance data, and session information for NetScaler Gateway.*

# Resolution testing

Before describing how resolution testing should be done by administrators when troubleshooting a XenApp environment, there are two terms that need explaining: in software development terms, **resolution testing** is known as the process of retesting a bug once the development team has released a fix.

**Regression testing** is another methodology where test cases are re-executed for previously successful test cases.

Both testing methods are an important part of testing a software solution, as sometimes, fixing one bug can cause regressions in other parts of the solution leading to new bugs.

Citrix administrators need to think in the same manner as testers do. Once the problem has been understood and a fix has been identified, then the fix or workaround can be applied. Once the fix is applied, the next step is to attempt to reproduce the initial issue. If this is not successful, it would generally mean the initial issue is resolved and most of the time that is the case.

However, besides testing for the initial issue, a Citrix administrator should also perform a number of tests to ensure that the fix does not negatively affect the XenApp infrastructure in another manner, for example, another application might stop working.

# Root cause analysis

Once the problem has been correctly understood, a fix applied and tested, the next step would be to determine the root cause and apply corrective actions if needed.

The **Root Cause Analysis and Corrective Actions** (**RCCA**) is the final step in troubleshooting a problem and involves determining the root cause of the issue and outlining any suggestions and recommendations for actions that can be implemented to prevent the reoccurrence of the underlying issue.

Most of the problems encountered in the Citrix world can be grouped into three categories:

* Performance issues, for example, applications are slow to start, network is unreliable, and so on
* Incorrect configuration, for example, XenApp is not properly configured during initial installation or subsequent change
* Broken code leading to unexpected behavior from XenApp or underlying components—these are trickiest to debug and probably the least encountered.

Most root cause analysis reveal either a performance issue or an incorrect configuration.

Where a root cause is deemed to be performance related, usually tackling them requires improvements in the infrastructure—bigger bandwidth, more servers, faster disks, and so on. The real challenge is determining how much to scale the infrastructure so that performance falls back within acceptable parameters without spending a large amount of money.

Preventive steps for these types of problems could be:

* Ensuring a capacity management process is in place
* Monitoring Citrix infrastructures for active usage
* Creating an easily scalable Citrix architecture

Incorrect configurations are usually self-evident. An administrator performs a change that negatively affects the Citrix infrastructure, again usually almost immediately. The root cause analysis, therefore, focuses on the following questions:

* Has the change management process been followed?
* Have the risks been properly established and highlighted?
* Have actions been considered to minimize the risks?
* Is there a backup plan in place in case a rollback is needed?
* What is the impact of a failed change and how it will affect users or production environments?

Changes where the risks have been appropriately highlighted (“Changing X setting has the risk of bringing down the Citrix site for 15 minutes”), where the change is performed out of hours (minimizing risks) and has a proper rollback plan in place are perfectly acceptable.

Most changes have the potential of causing downtime but if the proper change management process is followed, the risks are minimized and the potential outage reduced.

Preventive steps for this type of problems could be:

* Ensuring the risks have been correctly identified and presented to the business
* Ensuring steps to minimize the risks have been identified
* Ensuring there is a clear backup plan in place

Finally, during troubleshooting, a number of changes might need to be done before the final fix is found. It is, therefore, a good idea to keep a track of these changes while the troubleshooting process is actively ongoing.

Once the correct fix has been identified, a retroactive change request should be logged in the IT system. Although, in this instance, the change hasn’t followed the standard change management approval process, it is still useful to have changes logged in the system in case they need to be looked up in the future as part of troubleshooting previous changes.

# Summary

In this article, we covered the basic methodologies of troubleshooting. We’ve described troubleshooting as first understanding the problem, breaking down the problem into their affected components, and finally, testing. The problems are solved once the fix or workaround is identified.

We highlighted the fact that sometimes, problems can be traced back to scheduled changes in the infrastructure and that keeping track of changes is important as it can help in identifying the problem and mitigating or resolving it.

Finally, we discussed the root cause analysis, the process of determining the root cause of the issue (not just the fix/workaround) and preventive steps to minimize the reoccurrence of the issue.