Troubleshooting Basics: A Practical Approach to Problem Solving

White Paper

Published: September 2005

Physicians are fond of saying "*Treat the problem, not the symptom*." The same is true for Information Technology.

Contents

Preface

Effective troubleshooting is an important aspect to any job, but it is especially important in Information Technology (IT) where problems are not always clear, and the impact of a "fix" could have unforeseen ramifications. Typically it is a skill that is developed and honed over time.

The goal of this paper is to describe a simple, easy to apply, and generic approach that should be invaluable to many IT workers. This was initially developed as an internal document, but due to the lack of good troubleshooting information available we decided to publish it as a public service.

Overview

Troubleshooting. Although it is one of the most important skills for an IT professional to master it is often the most neglected. What can be done to improve this skill? What can be done to expedite that learning process? The process described within this white paper has been proven effective and successful by our organization time and time again. While the approach is not revolutionary (many would argue that it is "common sense") it is one not used enough in business.

Root Cause Analysis – Identify the *Real Problem*

This is the single most important step in troubleshooting. A proper Root Cause Analysis (RCA) ensures the ***real problem*** has been identified and fixed – as opposed to just fixing a symptom. Moreover, it helps ensure that the corrective action has been taken to prevent the problem from occurring again. Unfortunately, in many organizations this is not done properly for one reason or another. Lack of time or skilled personnel are often cited as reasons in these cases. We believe that the true cause is often related to the lack of a defined process.

So what constitutes an RCA? Properly completed, a Root Cause Analysis asks the following questions, which are discussed in the text below:

- What is the problem?
- What are the symptoms of the problem?
- Why is the problem occurring?
- How did you become aware of the problem?
- Can the problem be recreated?
- How will you fix the problem?
- How will you prove that the fix works, and that it did not introduce new problems?
- How are you going to prevent the problem from occurring again?

Gather the Necessary Data

To start with you need to determine what the problem is. Unfortunately, this question can be more complex than it seems because multiple problems and symptoms can arise from a single failure incident. Take for instance the crash of a server that supports multiple network services, such as LDAP, DNS, SMTP, etc. Depending on the redundancy of the environment, the symptoms of this problem may be subtle - timeouts on DNS and LDAP queries, odd delays in email, etc. However, upon investigation you will find that the problems can be traced back to the one server that suffered the failure.

Of course, there are times when multiple, unrelated failures occur. In the example above, a failing content switch or firewall could account for some of the observed problems, seen along with the problems generated by the server. Here, there would be two root causes for the problems experienced – the first would be the server and the second would be the network switch/firewall. The trick is being able to methodically analyze the data in order to make that determination.

As a starting point, first check the systems documentation to determine if this is a known problem or if similar symptoms have been seen in the past. If the problem has been encountered before there may be a set procedure to clear it. Here, you would just need to execute the procedure and validate that the corrective action worked as intended. In addition, you will want to document this problem occurrence for tracking purposes and trend analysis.

If this is a new problem then begin your quest. The first phase of RCA is essentially a process of fully documenting the problem. This doesn't need to be formal (unless your organization requires it), but at the very least it should contain all currently known "facts" about the problem. What is happening? What is the scope? Is it restricted to one user, multiple users, or the entire enterprise? How long has the problem been occurring? **Identify what you think is going on based on the symptoms and other available information.** The bullets below summarize the key questions that need to be answered in this phase:

- Collate all the symptoms.
  - Which components of the application(s) are being used when the problem occurs?
  - Does it always occur with this component?
  - Are there multiple problems or just one problem?
  - Identify the sequence of events that lead to the problem, e.g., values entered, keys pressed, etc.

- Identify the affected users by type and location.
  - The problem may be specific to a group of users performing the same function, users in a single office/location, or all users.
  - Understanding the scope will allow you to isolate and focus on specific potential problem areas.

- When did this problem start happening?
  - This is not necessarily the same as when it was reported. The user may be reporting a pre-existing problem. It is important to define the timeline.
  - Is this the first instance of this problem for the reporting user(s)?

During the creation of this documentation care should be taken to capture any recent changes that have been made in the environment – no matter how small, seemingly harmless, or unrelated they may be. Often, problems are ***triggered*** by a recent change, but the change itself may not be the cause. That change may simply be exposing another problem with the implementation of the hardware, software, etc. (This brings up the subject of change control & management which although related to our topic is outside the scope of this paper).

The second phase of RCA is isolating the faulty component(s). The exact steps performed here are almost entirely dependent on the nature of the problem, the technology involved, and the complexity of the environment. However, as a rule of thumb it is best to start with a basic scan of the hierarchy of components involved, starting from the largest components first. For example, if a user is reporting a slow running application you should start by looking at the database server (computer) itself and the network. Next, move on to the database, then the client platform, and finally the user's own desktop computer.

Often you will identify the source of the problem as being something that is affecting many people, but only a few are complaining. This is especially true in environments with a lot of problems where people just become accustomed to issues. Other times you will find that specific programs, modules, or queries are involved that are only executed as part of a specific logic path. Either way, this isolation helps provide the focus necessary to quickly resolve the problem.

The final phase of RCA is confirming that you have identified the faulty component(s). For some issues this may just as simple as reviewing and saving the log files generated during the problem event. Other situations may require a test to be developed that will expose the faulty component.

Take the case of poorly performing SQL query. By performing a query analysis you might find that there is disk contention that appear to be causing the problem. But further analysis might that the result set of data is much greater than expected, or that a desired index is not being used. This could lead to changes in the storage structure, indexing, the optimization strategy, or even the query design. By developing this test case you can eliminate assumptions and later focus on the root cause that has been demonstrated to be true. Your efforts will now be directed to working on a true fix, as opposed to a work-around.

Determine A Course Of Action

Once the initial documentation has been completed and the fault identified, the next step is to determine the course of action. This can be as simple as one person listing out the changes they will make to a configuration file, or as complex as several developers determining how best to troubleshoot the flow of an order through a system. It's important to remember that in instances where more than one person is working on a problem there has to be coordination between team members. It would not be productive to have one person modifying the configuration on a system while another person reboots it! **Also keep in mind one of the basic tenets of corrective action – make only one change at a time and then prove that it works or undo it.**

While planning it is important to keep the following points in mind:

- Don't make assumptions - when in doubt you should investigate to validate that the information you have is correct. For example, user testimony is notoriously unreliable - so if a user states that a process is taking 10 minutes to run, ask to see a demonstration to make sure the time isn't being exaggerated through frustration by the user. Quantify as much as possible.

- View things as a system and not as individual components. The resolution of a database problem may very well be related to the operating system of the platform the DBMS software is running on, the network connections, or the client front end. It would be a mistake to immediately exclude these other components from the troubleshooting process just because the problem presented itself as a database issue.

- Don't be stubborn – Some problems are due to changes in the environment – applications, network, OS, etc. Instead of trying to make the system work with the new modifications, a more prudent solution may be to get the system back to a "known state" by backing out the changes. This allows the system to return to production while the IT staff reviews the problems encountered and has time to come up with a new implementation plan to address those known issues.

It is important to note that this step and the "Implementation and Testing" step below can and often are an iterative process. Although it is possible to solve a problem on the "first pass", it is very likely you will need to take the results from testing a change, and use them as input to determine a new course of action if the problem is not resolved. Take the extra step to truly validate (and document) the fix.

Implement And Test

At this point you have a plan of action and are ready to begin implementing it. During this process, it is important to follow some simple guidelines.

- **Only change one thing at a time!** Make and document the change, and then document the impact of that change. Even if the problem is not solved it is important to record this fact. This saves time if it is necessary to bring new people into the problem resolution process. It stops them from "reinventing the wheel."

- **Don't treat symptoms!** It's important when implementing a fix that you go back and review the initial problem documentation. Does this fix the problem, or does it just fix a symptom of a larger problem? For example, users are reporting problems sending and receiving email. IT begins troubleshooting the mail server, only to discover after several hours that the problem was related to a problem with name resolution due to one of the organization's DNS servers being down.

- **Know what constitutes normal!** A surprising number of IT departments do not know what constitutes a "normal" running system. Frequently, the department uses a tool such as Quest's "Foglight" or HP's "OpenView" to collect "hard" system metrics (such as CPU, memory, and disk statistics). These are important, however it is also necessary to know details such as the average amount of time to perform key functions (complete a sales transaction, complete a shipping transaction), the normal amount of time to generate a report, what "normal" system response looks like to a user, etc. This understanding will provide context for the system metrics and will help you to see the "big picture". Without this understanding it is easy to make mistakes based on a misunderstanding of the data.

Besides using the guidelines above, it cannot be stressed enough that the actions taken during any troubleshooting process need to be documented. Documentation provides a "road-map" of what changes have been made, helps to ensure the problem does not happen again, and makes it easier to transition the troubleshooting to a co-worker, or to explain the problem to a support resource. It is a good historical record for future problems, as the problems may differ, but your approach might provide insight for resolution.

Finding Closure – Documentation is Key

At some point in the troubleshooting process, you will find the problem and implement a fix (or series of fixes). Rather than move to the next issue it is important the current problem is closed out properly. This can vary by organization, but at the very least the following should be done:

- Documentation should be completed and signed by the responsible parties, with a detailed description of the problem, the corrective action, and any other pertinent information. This documentation should be stored in a place accessible to the entire IT department, such as a shared network folder, a corporate document repository (such as Basecamp or eRoom), printed and filed, or some other form of storage. **Remember, documentation is most valuable when it is accessible.**

- If the problem appears to be one that can recur – for example, a query issued by an application that is known to crash the DBMS server – then the resolution that has been implemented might just be a work-around. A plan of corrective action should be defined and tasked out to implement a fix. Detailed information on this task should be attached to the problem documentation, and it should be referenced as both part of the Design and QA work for the implementation of the fix.

Summary

The ability to address a problem properly - define it, investigate it, correct it, and close it out - is a key skill in today's IT department. Downsizing and consolidation has made it necessary for workers to be responsible for multiple areas of overlapping technology, making root cause analysis an extremely important aspect of the quick and complete remediation of problems.

The creation and maintenance of proper documentation can help with the prevention of future problems as well since it builds a knowledge base to work from. It can also provide the ability to produce a library of systems procedures and specific tests for future use. This can translate to the ability to rapidly transition tasks and knowledge to other team members or support personnel. Although there are exceptions to every rule, the application of the guidelines presented above should help in solving a majority of these types of problems.

About the Author

Jason Schmidt, PMP is a Principal Consultant at Comprehensive Solutions. He has over ten years experience in Information Technology, ranging from staff level work such as Programming, Database Administration, and System Administration, to key Leadership roles such as Project Management of several client projects, Supervision of a group of technical consultants, and Business Development. Effective Troubleshooting is one of the many skills that our consultants often teach our clients during the course of a consulting engagement.

Let Us Help You Succeed!

Call today to discuss ways that Comprehensive Solutions can help your organization save money and achieve better results with your IT projects. We provide the ***confidence*** that you want and deliver the ***results*** that you need.

Back to White Papers
Back to Services

Comprehensive Solutions
4040 N. Calhoun Road
Suite 105
Brookfield, WI 53005
U.S.A.

Phone: (262) 544-9954
Fax: (262) 544-1236