A Guide to the Art of Troubleshooting

By Geoff Halprin, The SysAdmin Group P/L

Background

This paper is a discussion paper on the art of troubleshooting. The intention is to impart some ideas, hints and tools to assist less experienced systems administrators in gaining valuable insight into the attitude and mind-set of senior systems administrators.

What is Systems Administration? System Administration is a combination of proactive and reactive duties which includes planning, problem avoidance and problem resolution. (This is a vastly reduced definition in view of the discussion at hand.)

Systems administrators deal with the vast complexity of interrelated components of a system as they attempt to isolate and resolve problems based on symptoms that are not always repeatable.

What is troubleshooting? Troubleshooting is the art of taking what a user sees as a problem, determining the real underlying nature of the problem, and then resolving it (if possible). I used the word “art” to describe this practice, as much of what is done is based upon experience, and difficult to articulate or quantify. It is this ability to define a process which turns an art into a science. This paper is my best efforts at that goal.

Our role as troubleshooter is similar to a doctor’s role in dealing with the intricacies of the human body. He must obtain information about symptoms from the patient then, armed with a large base of knowledge about the body, probe further and isolate the real underlying cause of the symptoms, and then treat the real problem.

Rules of the Game

1. You are NOT expected to *know* the answer; you ARE expected to *find* ***an*** answer.
2. You are NOT expected to justify your existance by explaining exactly how you do your job to the customer. They have hired you to the job of troubleshooter; show them you can by your results. It is only when dealing with technically competent customers, such as other systems administration staff, that you will need to explain how you fixed the problem.
3. Speak to the customer in their terms, not ours. Do not speak jargon or provide information that the customer is not interested in. If all they want to hear is “you are better,” then that’s all you should tell them. Let the customer guide you as to how much information they are after.
4. Do not make assumptions about a customer’s knowledge or technical abilities. Wherever you have asked a customer to perform a task for you, verify that they did this with specific “yes / no” questions. e.g. Instead of asking, “did you check that the cable is plugged in?” ask, “is the cable plugged into the port marked ‘a’ on the back of the Sun?”

The Process

The process described below is intended (as is this entire document) as a guideline. The process is iterative - you don’t finish a step and move onto the next step never to return; you will often uncover information in one step which requires that you return to a previous step. The intention here is to provide an overall process which must be followed, however formally, informally or even subconciously, which guides us towards the desired end point.

What is presented below, then, represents my understanding of the steps a systems administrator must perform in order to confidently resolve a problem.

PHASE I - PROBLEM IDENTIFICATION

Step 1. Interview The Customer

You must obtain as much information as possible about the symptoms that the customer sees in order to diagnose the real, actual problem. Note that at this point, we are only dealing with a customer *perceived* problem. We are attempting to obtain a clear description of that problem.

In this step we are attempting to uncover the nature and symptoms of the problem, what its impact on the user base is, and what actions have been taken thus far by the customer as a result.

- Do not give an immediate answer. Take the symptoms on board and, no matter how sure you are about the actual problem, do not state this as fact. You can suggest that you have suspicions and, if these are confirmed, then it will be fixed in ***x*** hours / days / whatever. Use phrases like “this will take time,” and, “I need to do some research” to defer estimating the solution until after you have performed the diagnosis step.
- You must identify and compensate for assumptions made by the customer in providing or filtering information.
- Be sure to get complete details as to what steps the customer has already taken to resolve the perceived problem. He may have (a) made things worse, (b) obfuscated the original problem, or (c) created a new problem.
- We are trying to attain consistency of observable symptoms. Without this consistency, we cannot isolate the problem. If necessary, have the customer perform specific tests or defer the problem resolution until it occurs again, this time with the customer armed to observe specific aspects of the system in order to attain this consistency.

Important Questions:

1. “What are you trying to do?” It may be that they are requesting the impossible, and that already exists a simple way to achieve their goal that they are not aware of.
2. “What has changed?” This is the single, most important question, as 99% of all problems are the result of a change. (If something didn’t change, then the problem must have always been there!) Note that the user will often discount changes that they feel are unrelated, so you will need to develop skills for obtaining apparently irrelevant information, perhaps as a result of some initial diagnosis work (see step 2).
3. Symptom isolation questions such as;

   “Does it happen every time?”
   “Does it happen for all users?”
   “Is it time specific?”
   “Is it file specific?”
   “Is it location / hardware specific?”

Step 2. Determine the Nature of the REAL Problem

At this point, we still only have the customer’s word that a problem exists. (We have hopefully had them demonstrate it to us in step 1.) Before we begin work on diagnosing the problem, we must satisfy ourselves that a real problem does exist, and preferably in a repeatable form.

This step is closely related to step 1, as you will most likely need to ask the customer further questions as you uncover the nature of the problem. This step has been separated from the previous step because, where step 1 was primarily communication with the customer, this step is primarily examination of the system.

In this step, we are attempting to interpret the customer’s symptoms, and identify the underlying technical systemic symptoms of the problem.

Step 3. Verify/Replicate the Problem

Hopefully in step 2 we were able to verify the existance of a problem, but we now have to attempt to make the problem readily replicable. Basically, this means identifying a sequence of commands which will consistently cause the problem to occur.

Of course, if the problem is a system crash, or something equally painful, then it may be appropriate to skip this step. :-)

PHASE II - DIAGNOSIS

Diagnosis is similar in concept to the combination of systems analysis and systems testing.

You are hoping to quantify the bounds of the problem space and the complex interplays within the problem space. You must gain an insight into the various components which form the system in question and how they interrelate, how data flows around the system, and also the temporal aspects of these interactions. This is systems analysis.

Having done this, you wish to identify where within this system the problem occurs, such that we can resolve it. This is systems testing. We use the same basic techniques as we do for testing in software development; black-box, white-box, top-down, bottom-up, etc.

What is unique is the way we must combine these two disciplines. We use systems testing to gain better insight into the makeup of the system, which in turn guides the analysis process.

Step 4. Research the Problem

Having accurately replicated the symptoms of the problem, there is a fair likelyhood that you are not the first person to have encountered this problem. As with other fields of diagnosis, such as medicine, heavy use should be made of research materials.

There are many available sources of knowledge in which to research the problem;

- Someone may have already described this problem and documented a solution in the a local site knowledge base.
- This solution (or even just a bug report) may be found in SunSolve, the Library, the various Usenet newsgroups, Internet mailing lists, etc.

Effective searching of databases is a useful tool for isolating and eliminating components.

Step 5. Isolate the Problem

One definition of ‘isolate’ is “to reduce the number of variables.” I think this definition is especially appropriate as it implies the previous step; “identify the variables.”

So, there are two distinct steps to problem isolation;

1. Identify the components of the problem space
2. Isolate the fault by testing individual components or groups of components, eliminating components that work correcty.

TOOL: The Diagram

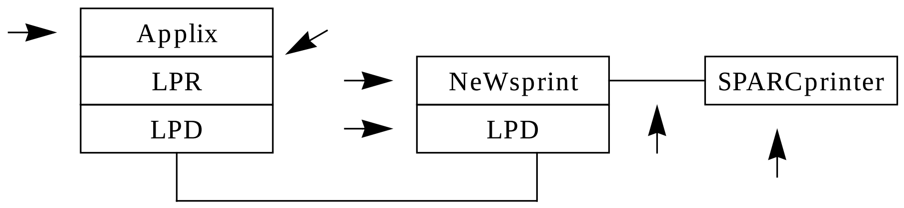
A useful tool for this phase is a diagram. In this step we will first draw the components, both hardware and software, and show the connectivity of these components. We will then annotate this diagram to direct our efforts in isolation of the problem.

In drawing the system diagram, the following guidelines are useful;

1. Draw all hardware that falls within the bounds of the system. Show all cables, both client and server hosts, comms boxes, etc. i.e. Draw ALL appropriate hardware.
2. Use a large empty box for each host that is involved in the problem space.
3. Now draw the software layers in each host box as appropriate. We are aiming to show the exact path that the data must flow in order to get from beginning to end. The hardware connection points should clearly emerge from the appropriate software layer.
4. We do not have to draw a diagram down to the ***n***'th detail. As a first run, we are attempting to identify the subsystems that are involved in the problem space. Once we have isolated the sybsystem that is at fault, we will "break open" that box, and draw a like diagram for that subsystem. We will continue to peel the onion until we find the fault. More on this later.
5. Having drawn the components, we have almost accomplished part one. Now we must annotate this diagram by drawing an arrow at every point of potential failure. We have now identified the components which we must test for correct behaviour.
6. As we work through these components, using the techniques described below, we can cross off arrows, until we are left with a single point of failure having been identified. Note that some of the techniques described below will effectively halve the problem space, eliminating half of the identified components from even needing testing.
7. As with everything else in this document, the process is iterative. This tool is meant to guide the process.
8. This diagram also serves as a useful tool for discussing the problem with other people to obtain their input and ideas on how to proceed.

To assist in the explanation of this tool, here are 2 examples;

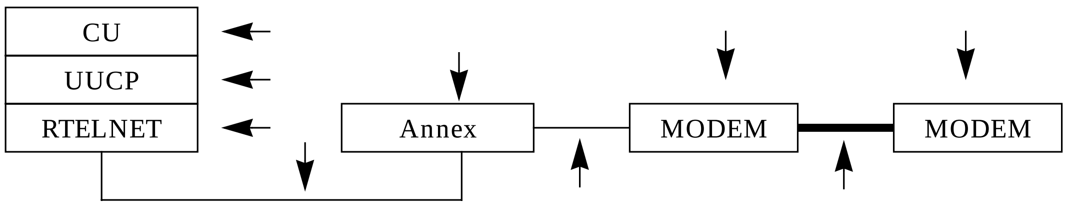
Example A: Occasionally, a NeWSprint job is appearing shifted down the paper when printed from the Applix application.



In the above diagram, we can see that the major sub-systems that are involved in the problem space are; (1) Applix, (2) LPR/LPD, (3) NeWSprint, and (4) the SPARCprinter. We can now start our testing.

As an example of the iterative nature of this process, in this example we may find that we need to break open the NeWsprint box to reveal the presence of OpenWindows.

Example B: A complex modem interaction is failing;



In the above diagram it becomes clear that major points of potential failure include; (1) CU / UUCP config, (2) RTELNET invocation, (3) annex port configuration, (4) annex / modem cable, (5) source modem profile, and (6) destination modem profile. As with the previous example, we may need to peel the onion to reveal that the annex / modem cable consists of an RJ-45 to RJ-45 cable, and an RJ-45 to DB-25 headshell.

Testing: Techniques

When testing a system attempting to isolate a problem, we need to draw on as many tools as we can. To paraphrase P. J. Plauger [Programming on Purpose], “If you had to build a wooden table, would you use a saw, a lathe, a hammer or a screw driver?” Of course, the answer is you would use different tools at different times, as appropriate. The same goes for testing. There are many techniques, and we should use these in combination to achieve our goal with maximum effectiveness. Below are some of the tools you should have in your kit;

1. **Top Down Testing**. In top down testing, we are attempting to split the problem space in half. We do this by choosing a point somewhere in the middle of the system (approximately the same number of potential failure points on each side), then test one side for correct operation.

   For example, we could attempt to just use **lpr** to print a saved PostScript file (one that printed badly) from the machine which has the printer. This reduces the problem space. Likewise, we could swap physical printers and see if the problem re-occured.

   Obviously, the number of potential failure points on each side is only a guide, as it is more important to choose a point where we can quickly and simply ascertain which side the problem is on.

2. **Bottom Up Testing**. This is, as it sounds, the opposite of top down testing. Here we start at one point and prove “outwards” from that point. In the simplest case, we start at one end, so that we are only proving in one direction.

   This technique might not seem very helpful, but we can use this technique in conjunction with top down testing to remove specific points of potential failure from the problem space, with possibly great impact on reducing the overall complexity of the problem.

3. **Black Box Testing**. As the name implies, here we pretend we are dealing with a black box which we cannot see inside. We prove to our satisfaction that the box is functioning correctly, then use this new fact to test other boxes.

   Black box testing means comparing input to expected output in a test suite to prove that the box is functioning correctly.

4. **White Box Testing**. Here we are looking inside the box to ensure that it is behaving properly. Having found a problem with a component of the system (a black box), we must now break open that box and look inside - repeating the process of identifying the components within the bounds of this box, and then testing these sub-components.

5. **Coverage Testing**. Here we test a white box which has alternate paths through it, so that we can be sure it works correctly under all circumstances. Thus, we are testing that all paths through the box work as expected . Once we are happy with it, we can close the box and it becomes a black box.

6. **Iterative Testing**. As with everything else, we use the above techniques to successively home in on the problem. We peel the layers of the onion. Start by breaking the problem down into a few key subsystems, identify which subsystem is broken, then troubleshoot that subsystem.

7. **Regression Testing**. Just because our initial test suite suggested that a component was fine, doesn't mean we assume it is faultless from then on. Always examine new test results with an open mind. This may require re-testing components that you thought were fine (at least, until that last inconsistency).

HINTS

- What diagnostic tools are available? Always think about what tools the system provides to assist you in your task. Examples of some very handy tools are; **trace**/**truss**, **netstat**, **ps**, **proctool**, **find** and **pff**. Remember you are only trying to prove correct or incorrect behaviour of a component.
- Record all results. A common mistake is to think you can remember the test results from a series of 8 tests. You then move onto the next component, and suddenly need to recall whether test 3 failed or not, and exactly what the test conditions were, as you have just uncovered a hunch while investigating the next component. Always write down the test conditions and the results. If appropriate, save the input data and output. Remember, this is a scientific process. In the case of GUI driven software, you may need to record the mouse movements and menu selections the user is using to cause the problem. (We needed to do this at a customer site when diagnosing their performance problems.)

Step 6. Resolve the Problem

Our immediate reaction, as technically capable people, is to solve the problem, whatever it takes. This may not actually be in the customer's best interests! It may take too long, or cost too much to actually solve, and an alternative solution or workaround may be called for.

More likely still, is that there will be a number of ways in which the problem can be solved. There will be multiple technical solutions to the problem, each with their own pros and cons. There may also be non-technical solutions or procedural solutions.

Here we need to identify the possible solutions, both technical and non-technical, including work-arounds. Along with each of these we need to define the parameters for the choice of solution. Sometimes this will require customer approval of the particular recommended solution.

PHASE III - CLEANUP

Step 7. Test the Solution

At this point, we believe we have solved the problem. We must now attempt to duplicate the problem using the sequence we discoved in step 3. If this test fails, then it's back to the drawing board!

- Be aware of cascading errors. Fixing one problem, especially if you are using a workaround, may cause other problems in the system being worked on, or in surrounding or interrelated systems. Be sure to test all systems which may be affected by the changes you are making. Use a series of regression tests to ensure that the system still exhibits correct behaviour under a variety of circumstances.

  More subtle that this, is that fixing one problem may ***uncover*** another, previously hidden, problem. Thus, there may be a series of problems that must be resolved.

Step 8. Get Customer Acceptance of the Solution

As with any problem resolution, we need to do more than satisfy ourselves that we have done our job. The customer must also be happy. Too often I have seen a systems administrator who is happy with the job they have done and closed the call, only to find out later that they never solved the customer's problem, only theirs.

Step 9. Followup and Outstanding or Arising Issues

The solution that we implemented may have been a workaround, or a short-term solution, and a more rigourous solution must follow. There could be any number of issues that arose from the process we have just undertaken. The failure could have pointed to the need to drastically overhaul some process, system or procedure.

These need to be written up, and then followed up in an appropriate timeframe, by the appropriate people.