5

OTHER TROUBLESHOOTING METHODS

Substitution and fault insertion

"Remove and conquer"

"Circle the wagons"

Trapping

Consultation

Intuition and out-of-the-box thinking

5.1   WHY USE OTHER TROUBLESHOOTING METHODS?

The previous chapter discussed logical/analytical frameworks for troubleshooting. While these frameworks work most of the time, some problems require less systematic techniques to complement the logical frameworks. Normally, you will begin to use these other techniques only after the logical analysis has failed to suggest a viable solution. Defining the problem, gathering information, and performing analysis still take place when you use these methods.

You may need to approach troubleshooting from a different point of view because a system may be too complex or sophisticated to troubleshoot with the knowledge available to you. This can occur with microprocessor-based equipment consisting of multiple components (such as a PLC or a DCS). Sometimes manufacturers provide only limited information about what goes on inside the equipment. Maybe the problem is transient in nature, or is in a complex system with communication links between components and multiple power systems and grounds, as in a multiple variable-speed drive system.

5.2   SUBSTITUTION METHOD

The substitution method is troubleshooting by substituting a known good component for a suspected bad component. For modularized systems or those with easily replaceable components, substitution may reveal the component that is the cause of the problem. First, define the problem and gather and analyze as much information as you can. Note that these steps are no different than the initial steps in the structured framework methodology. Then select a likely replacement candidate and substitute a known good component for it. If the problem goes away, you have at least found a partial solution. Then evaluate to see if a more general solution is needed. For example, if the component can be repaired on-site, either troubleshoot it further to find the lower-level cause of the problem or return it to the manufacturer for analysis.

By substituting components until the problem is found, the substitution method may find problems where there is no likely candidate, a group of candidates, or even a vague area of suspicion. One potential problem with modular substitution, though, is that a higher-level cause can damage the replacement component as soon as you install it. This may confuse the issue if the failure is immediate as you will generally have the same symptoms after the replacement.   If the failure is not immediate, this will give you a clue that the real cause of the problem is external to the module. The use of this method can raise the overall maintenance cost due to extra module cost and the cost of inventory of replacement modules. Even more problematic are cases in which the higher-level cause does not damage the replacement right away.

Another form of this method is to substitute or insert a known good signal or value into a system to see where a problem comes from. If you insert a known good signal and the downstream part works properly, then the problem is upstream. The converse is also true.

5.3   FAULT INSERTION METHOD

Sometimes you can insert a fault instead of a known good signal or value and see how the system responds. For example, when a software interface keeps locking up, you may suspect that the interface is not responding to an I/O timeout properly. You can test this by inserting a fault—an I/O timeout. Another example of fault insertion would be inserting a bad value into a point in a computer program to see how the program responds. A third example might be inserting a transient into a system, such as a simulation of a voltage sag.

5.4 "REMOVE AND CONQUER" METHOD

For loosely coupled systems that have multiple independent devices, removing devices one at a time may help to find certain problems. For example, if a communication link with ten independent devices talking to a computer is not communicating properly, you might remove the boxes one at a time until the offending box is found. Once the problem device has been detected and repaired, the removed devices should be reinstalled one at a time to see if any other problems occur.

The "remove and conquer" technique is particularly useful when a communication system has been put together incorrectly or exceeds system design specifications. For example, there might be too many boxes on a communication link, cables that are too long, cable mismatches, wrong cables, impedance mismatches, or too many repeaters. In these situations, sections of the communication system can be disconnected to see what happens.

"Remove and conquer" can also work for grounding problems. To detect whether a system is grounded in two places, try lifting a ground to see if things get better. (If you are lifting a safety ground, take care that you are protected while doing this.) One common problem for which this method is useful is when a shield is grounded in two places, causing a ground loop; if the regular shield ground is disconnected and the system improves, then another ground connection on the same shield may be the problem.

A similar technique, "add back and conquer," means removing all the boxes and adding them back one by one until you find the cause of the problem. For example, on a new communication system like the one mentioned above, the boxes were removed one at a time and replaced, but no offending box was found. But when all the boxes were removed and added back one at a time, the troubleshooter found that there were too many devices for the computer's port to support. This could also have been detected if the devices were removed one at a time and not replaced and a point was found where the system worked.

5.5 "CIRCLE THE WAGONS" METHOD

When you believe that the cause of a problem is external to the device or system, try the "circle the wagons" technique. Draw an imaginary circle or boundary around the device or system; then see what interfaces (such as signals, power, grounding, environmental, and EMI) cross the circle. Then isolate and test each boundary crossing. Obviously, if you do not identify all the boundary crossings, you may miss the one causing the problem. Often this is just a mental exercise that helps you think about external influences, which then leads to a solution. Figures 5-1 and 5-2 illustrate this concept.

FIGURE 5-1
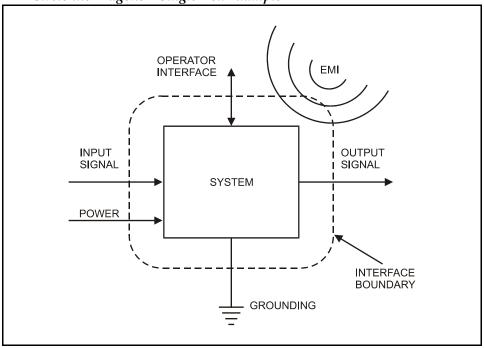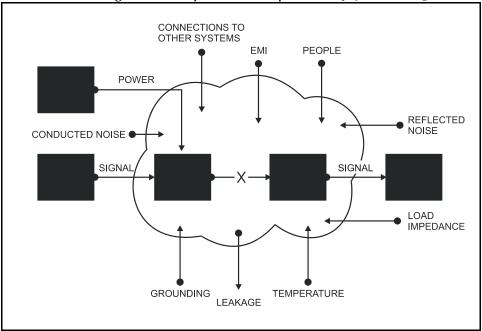   *Circle the Wagons—Single Box Example*

FIGURE 5-2
   Circle the Wagons—Multiple Box Example (courtesy of Control Magazine)

5.6 TRAPPING

Sometimes the event that causes the problem is not alarmed, or is a transient, or happens so fast the system cannot catch it. This is somewhat like having a mouse in your house. You generally cannot see it, but you can see what it has done. How do you catch the mouse? You set a trap.

In sophisticated systems, you may have the ability to set additional alarms or identify trends to help track down the cause of the problem. For less sophisticated systems, you may have to use external test equipment or build a trap (see Figure 5-3). Power monitors such as the Dranetz 658 (see Figure 5-4) are often used for power problems. A storage scope may also be used to trap transients. Portable data loggers can be connected to monitor variables over time and dump the results into a computer, where the information can be graphed or evaluated. Examples of these include the AEMC Simple Logger, which can monitor voltage, current, or temperature, or the HOBO series from Onset Computer Corporation, which can log several different variables.

FIGURE 5-3
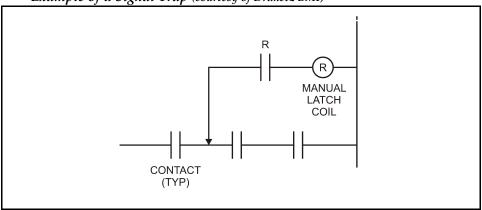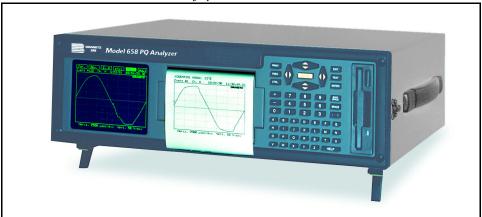*Example of a Signal Trap (courtesy of Dranetz BMI)*

FIGURE 5-4
Dranetz Model 658 (courtesy of Dranetz BMI)

If software is involved, you may have to build software traps that involve additional logic or code to detect the transient or bug. In programs that use languages like FORTRAN or BASIC, a debugger may be available, or you can add print statements to print out intermediate values to detect the problem. If you are the programmer, consider putting in diagnostic print statements and having a software switch or switches turn them on and then print the results to a log file. You can use multiple levels of diagnostics with different switches to trap in different places.

5.7   COMPLEX TO SIMPLE METHOD

Many control loops and systems may have different levels of operation or complexity with varying levels of sophistication. One troubleshooting method is to break systems down from complex to simple. This involves finding the simple parts that function to make the whole. Once you find the simplest non-functioning "part," you can evaluate the non-functioning part or, if necessary, you can start at a simple known good part and "rebuild" the system until you can find the problem.

A common example of this is a varying or oscillating control loop. By placing the control loop in manual (moving from automatic control [complex] to manual control [simple]), one can determine if the automatic part of the system (commonly the tuning) is causing the problem or if the problem is being caused by the process or other external inputs. Another example is a cascade loop where you have a master loop and a slave loop. Cascade loops are commonly used to isolate variations in the slave loop measured variable from causing variations in the master loop measured variable (the desired control variable). Breaking down this kind of loop involves breaking the cascade by placing the master loop in manual or breaking the loop at the slave controller to see if the problem goes away, which can tell you which loop or if the process is causing the problem.

Computer control, either cascade or direct digital control, can be troubleshot sometimes by breaking the computer link, though this may be done for you as it is typically the first thing an operator does when he has a control problem in this type of system.

Hierarchical systems are another type of system that can be sometimes troubleshot using this method by isolating the different hierarchical levels from each other and reconnecting to find the problem.

Another example of this method is breaking down a complex system into sub-units that have defined inputs and outputs and verifying each sub-unit's functionality. A sub-unit can typically be broken down into a black box representation as shown in Figure 5-5.