



ConvergeOne

3 CRUCIAL STEPS TO ACHIEVING MONITORING SUCCESS IN A COMPLEX IT ENVIRONMENT



Monitoring systems are overloaded with complex data that can be difficult to parse. IT teams must act as detectives to weed out the noise, identify that a genuine problem exists, and diagnose and remediate it. Further complicating matters is the fact that many problems and issues can't be detected and diagnosed based upon the data provided by applications. As a result, IT teams must employ a different set of methodologies to avert system outages. This article will share the steps to achieving monitoring success in spite of these challenges.

REACHING A STATE OF MONITORING NIRVANA

Mark Langanki, Chief Technology Officer, ConvergeOne



Monitoring has proven to be one of the more difficult components of IT for many years, perhaps because we make too many assumptions about how monitoring works and what the monitoring tools we use can actually do. We've all been mesmerized by demos at trade shows that boldly claim to have finally "cracked the code"! Unfortunately, after we purchase the tool, take it home, and install it, we begin the long, winding road to disappointment when it doesn't do what we thought it would do. Alas, the tool just showed well on the trade show floor.

Why is it such a struggle to find the holy grail of monitoring systems that can fix all of our problems? It's simple: One tool alone can't possibly fully monitor today's complex IT environments—oh, and monitoring doesn't work unless we have a full understanding of what's really going on within our environment.

In this article, I will review the monitoring issues around the data presented from our systems and applications and the steps to reaching a state of monitoring nirvana.

Step #1: Receive Comprehensive Data

One of the main issues with monitoring comes from a lack of understanding about the flow and value of data. First, let's think about the initial source of the event we wish to monitor. Where does it come from? What does it say? Most tools are betting on the developers of software, hardware, and operating systems being able to provide clear, concise data that describes exactly what is going on with the code when something goes wrong.

If you ask software developers and engineers what percentage of their code base is for exception (error) handling, you will likely hear anywhere from 50 to 80 percent. Your initial reaction may be that this is a good number, but the key to handling exceptions in code is understanding the output.

Problems arise when we are unable to identify the full definition of the issue as soon as it arises. There is a subliminal miss when developers fail to consider that the error generated by their code is not being read by a fellow developer, but rather a managed services engineer who is keen on finding the needle in the haystack.

So, how significant of an issue is this? Well, let's think back to the fateful December when Target got hacked. There were over 40 monitoring tools in place looking for security vulnerabilities or malicious behavior. Only one of the 40 found the issue, but it identified the behavior as "malware.binary." What, exactly, does that mean? [According to Quora](#), malware binary is "an executable program code which could also contain data/instruction for malicious malware program to run in your OS."

Now, in hindsight, we understand what this means, but let's think about the issue from when the Target hack occurred. What if the developer of the software spent another 12 seconds to type out something to the effect of, "malware.binary detected, found activity on system [insert the IP address] which indicates an infected system with malware, time to quarantine and take off the network"? If that didn't work, the tool should have noticed the behavior was taking place on the point-of-sale systems—which are the kind of systems you *really* don't want people to hack into, considering they contain credit card and other personal financial information.

At this point, you may be reaching the conclusion that the fix is a simple one: all we need is for developers to provide us with more comprehensive data. Not so fast. That is just the first step.

Step #2: Remove the Noise (and Solve Mr. Boddy's Murder)

Even if we have perfect data that defines exactly what's going on, we need to take the next step to remove the noise. In this case, noise is defined as things that are immaterial to the ability to understand what's going on and what's important. In other words, what's important is defining, understanding, and resolving an issue—and if something isn't contributing to doing any of these things, it's noise.

Remember the statistic we shared earlier that in the course of developing code, programmers dedicate a large percentage to exception handling? Well, some of the exceptions that are encountered and reported are not truly problems that a monitoring application needs to further act upon. The true art is being able to identify which data points require further action within the context of the environment.

Think of it this way: In the game of Clue, we are tasked with solving Mr. Boddy's murder, but the search for the killer is made simpler because we have a narrowed-down list of suspects, murder weapons, and locations to investigate. In the complex world of IT troubleshooting, a.k.a. detective work, it's not that simple! Sometimes we don't even know what we're investigating. Chasing down false leads is a big part of the problem we need to resolve.

In summary, we know that within our complex unified communications (UC) and contact center (CC) IT environments, there is a wealth of information that becomes exposed via monitoring applications—but the quality of the information that our service engineers receive can sometimes leave much to be desired. Because of the network-based nature of our UC and CC applications, we require an end-to-end, multi-vendor approach to monitoring and management that is adept at weeding out the noise.

Step #3: Diagnose the Asymptomatic

Imagine that we have great data that tells us everything we need to know with no noise. Have we solved all of our issues? Not quite—and the next issue is a doozy.

Monitoring that's based on waiting for data to be sent, captured, and analyzed accounts for a small percent of actual problems that occur. There are many issue types that cannot be caught by the standard method of what we call "traps" in monitoring.

Here are a few examples that can't be monitored via standard Simple Network Management Protocol (SNMP) tools:

1. **Logic errors:** We configured our voice response system to be closed on Thanksgiving by date, but we forgot to update that configuration for the new year, and now we will be closed when we need to be open and vice versa. No monitoring application can monitor this since technically nothing is broken.
2. **Bugs:** Oh, bugs, you get us every time! We can't monitor for something when we don't know how it will perform. This is not referring to application failures that have known errors. We are talking about bugs that result in things just not working as expected.
3. **Configuration errors:** Yes, we could look for these, but they would not be represented in a failure in an error log/SNMP trap.
4. **Compatibility:** Whoa, now we are talking outside of a single system. How can we know if two or more things are compatible just by waiting for errors to be generated from an application?

The list keeps going, but you get the point. There is no single easy way to identify that something has failed and determine exactly what should be done about it.

I know what you're thinking: Great, is there *anything* we can do to achieve true monitoring success (which we were fooled into thinking was such an easy feat when we witnessed that awesome demo on the trade show floor)? The answer is yes, but to get to that point, you need to think about monitoring differently.

1. Adopt the philosophy that there is a difference between good data and data that looks good.
2. Understand that a manual process to manage monitoring systems will create a cost that far outweighs the value of the things it can find—and that's *if* you do it right.
3. Acknowledge that we can't trust the data from developers and need to find a way to normalize the default messages to more meaningful instruction sets that describe the specific actions that must be taken.
4. Adopt another philosophy that if something is not "actionable," then it shouldn't be done. Graphs are pretty, but if you can't make a business decision based on the data or measure the positive impact, then don't bother creating or referencing them.
5. Accept that no one wants to keep their eyes glued to a screen, waiting for status updates. Skip the wasted cost of having humans sit around and wait for something to go red.
6. Recognize that no single tool, software, or method will work for all of your systems. You need a platform that can integrate and leverage the value of multiple tools and software.

We are all searching for ways to make the complicated easier to understand. You can achieve your monitoring vision—but you first need to understand the reasons why your current monitoring system doesn't work and identify the steps to move into a mode that allows you to find the issues in your systems quickly and effectively.



About the Author

As CTO, Mark spearheads ConvergeOne's innovations in technology solutions and services. He spent 25 years as the Chief Operating Officer at Spanlink Communications, which ConvergeOne acquired in 2014. A graduate of the University of Minnesota, Mark has been an instructor there since 1996 and in 2014 assumed the role of Associate Program Director.



TAKE THE FIRST STEP TO IMPROVING YOUR MONITORING CAPABILITIES

ConvergeOne's OnGuard platform monitors and manages over 250 customer environments in order to maximize system availability by alerting our engineers and our customers' engineers of problems as soon as they occur—or, in many cases, before they occur. Discover how ConvergeOne can assist you with reducing downtime and improving customer service through enhanced monitoring capabilities: convergeone.com/onguard