Downtime Prevention Through Software Standardization

Vern Bolton, Program Manager, Industrial Solutions Center, Schneider Electric North American Operating Division

Time was, if a machine went down, it required a member of the maintenance staff to drop what he or she was doing and connect a laptop computer to the machine to troubleshoot what went wrong. But there is a faster way to accomplish that same task today—through root-cause diagnostic logic built into the software that operates the machine. Essentially, software logic can be written to stop a machine if there is a problem and simultaneously send a message to a human-machine interface (HMI) or Supervisory Control and Data Acquisition (SCADA) screen, noting what's wrong and how to address it. From there, the issue can be attended to by a line operator if it's a simple fix, or a maintenance staff member if it's more comprehensive.

While this feature (typically called integrated guaranteed root-cause diagnostics) helps reduce or even eliminate potentially long periods of downtime—and the financial consequences that could result—it is most effective when incorporated into the software of all machines. The only way that can be accomplished is through development and implementation of a software standard. There are many benefits:

- It allows a facility manager to reconfigure processes with less impact on time required to rewrite logic, meaning the software can easily evolve based on changing conditions as the years pass.
- OEMs are able to get machines to market faster because logic is pre-written and tested by the software supplier.
- If all OEMs are applying the same standard, the final destination for a given machine doesn't matter, because line operators need only understand the nuances of the software logic. Plus, machines can be moved from facility to facility with no learning curve for operators.

But accruing all the benefits of a software standard takes implementation, of course, which can't be endeavored without astute project management and communication. A successful approach requires buy-in from perhaps dozens of OEMs; a means of disseminating and teaching the standard; and perhaps most of all, the true leadership of the end-user to ensure accountability. In short, it's a team effort.

Standard Evolution

A software standard rarely, if ever, stays static. The reason is simple—the potential for downtime, and particularly a long period of unplanned downtime, is also never static, particularly when new projects or processes are added. There is simply too much at stake for the end-user manufacturer to stand pat with a standard.

Hence, one of the primary goals of a software standard is to reduce potential downtime by helping line operators and maintenance staff resolve issues quickly. A carefully considered standard should point a line operator directly to the device within a process that is causing problems and also give some indication as to what to do to resolve it. The value can be measured in time. Consider the following approaches when a process suddenly stalls:

Scenario A:

- Time an operator spends trying to self-diagnose the problem: 5 min
- Time to call maintenance and have a staff member arrive: 10 min
- Time for maintenance to diagnose using a PC: 15 min
- Time to rectify the problem: 5 min
- TOTAL: 35 minutes

Scenario B:

- Time a line operator takes to view an HMI screen, understand what has happened, and what steps to take: 1 min
- Time to rectify the problem: 5 min
- TOTAL: 6 minutes

In Scenario B, the software's logic was crafted to ensure a PLC published an appropriate message to the correct HMI, allowing the operator to know within seconds what the issue was, and how to rectify it.

Concerns about software degradation over time can be greatly diminished with standardization as well. A standard usually has a set of rules that must be followed for machine control and root-cause diagnostics to work. If the logic is written in such a way that when these rules are not applied, the end-user manufacturer can then be made aware of discrepancies, which can then be immediately addressed. Additionally, a control strategy can be designed that requires the diagnostic functionality to be in place to work, which eliminates the possibility of a staff member "jumpering" out a section of logic to avoid a component (e.g., a switch) to get a process running, and then never going back to fix the problem later. At the same time, it reduces the possibility of root-cause diagnostic messages being ignored.

For manufacturers that frequently change processes, a modular approach to software standard development can be a good tactic, because sections of the logic can be moved or reconfigured easily with no risk to the rest of the machine's logic. For example, if a software's logic was written in such a way that Task A is done before Task B, and an operator realizes it should be the other way around, those tasks can be swapped.

It's also important to note that development of a software standard, based on the needs and budget of an end-user customer, is only one part of the overall equation. The others are testing of the completed standard and dissemination to OEMs, which both require careful coordination.

Getting Started

A software standard may not make sense for all manufacturers. It requires an up-front investment for design, training, documentation and testing, and part of the payback is reduction of downtime by having better-written logic, quicker diagnosis of issues, and a more seamless project launch. But there is also the monetary ROI that should be considered from the outset; a manufacturer with a few machines in a single facility may not permit a reasonable ROI, if any.

But for those with hundreds of machines spread over multiple facilities, a software standard could mean the difference between meeting customer deadlines and expectations, or losing business to a competitor.