OPERATIONS RISK MANAGEMENT

When Your AI Logic Fails: Managing the New Physical Dependency

A guide for making operational risk management account for software dependencies.

Artificial Intelligence · Supply Chain Resilience · Operational Risk · Digital Twin Failure

> The newsletter for executives who prefer understanding systemic fragility over optimizing isolated processes.

Issue Summary: The New Failure Mode

In traditional manufacturing, we are used to physical failures. A motor burns out because of a bearing failure; a sensor fails because of environmental wear; a batch is scrapped because a chemical concentration drifted outside of the spec. These are tangible problems with traceable roots. We have tools for these—preventive maintenance schedules, spare parts inventories, and standard operating procedures (SOPs) to handle them when they occur.

Today, we are introducing a new type of failure into our production lines: logic-based dependency. When you integrate an AI model to perform tasks like visual inspection, demand forecasting, or predictive maintenance signaling, that model becomes a "moving part" in your process. However, unlike a physical gear, it doesn't wear out—it drifts. It can change because of a remote update from a vendor, or its accuracy can degrade as the real-world data environment shifts away from what it was trained on.

We are moving from a world where we manage mechanical reliability to one where we must manage logical reliability. If an AI model is making decisions about whether a part passes inspection or how

much raw material to order, that model is no longer just "software." It is a mission-critical component of your production line. If it fails, the line stops, even if every physical machine on the floor is running perfectly. We must begin treating these digital components with the same rigor we apply to our most critical hardware.

Name of the Problem: Abstraction Collapse

In many organizations today, there is a dangerous assumption that because "the AI works," it doesn't need an operational safety net. This leads to what I call **The Abstraction Collapse**.

This happens when leadership treats a sophisticated software tool as a magic box rather than a complex component in the supply chain. When the model performs well during the pilot phase, it is accepted without any secondary verification layers. The "abstraction" of the technology hides the underlying fragility of the logic. Because the engineers and managers don't see the math or the data weights behind the output, they assume the result is a constant truth rather than a variable calculation.

I have seen this play out on the floor when an AI-driven vision system suddenly begins flagging good parts as scrap because of a slight change in ambient lighting or a new batch of packaging material. Because there was no "fallback" logic—no manual override path, no secondary sensor check, and no local validation step—the line stayed down for hours while someone waited for a remote software update to fix the "logic."

The failure isn't that the AI made a mistake; it’s that the system was designed with zero tolerance for a logic error. We cannot allow our operations to be held hostage by an abstraction we don't have the tools to troubleshoot in real-time. If you can't fix it at the machine with a wrench or a manual override, then the risk of that component is not yet fully managed.

Why It Persists: Mistaking Performance for Ownership

The reason this persists is simple: it is easier to trust the "black box" than it is to build a robust, multi-layered process around it. Most teams treat AI as an "all-or-nothing" solution. If the model works today, they assume it will work tomorrow. This is not just a technical oversight; it's a failure of operational discipline.

We often mistake high performance for system ownership. Just because a tool provides 98% accuracy doesn't mean you own that result; you are merely borrowing it from the provider’s model. When we don't build in our own checks, we aren't "trusting" the technology—we are simply ignoring the risk of its failure.

The Comfort_able Rationalization	**The Underlying Reality**
"The AI is highly accurate; it doesn't need a manual check."	We have no way to verify *why* it's right or wrong today, making us unable to react when it fails.

The Comfort_able Rationalization	The Underlying Reality
"It’s just software; IT will handle the updates and fixes."	A logic failure on the floor is an operational stoppage that requires immediate local resolution.
"The vendor guarantees the model's performance."	The vendor's guarantee doesn't stop a production line from stalling at 3:00 AM on a Tuesday.

We have to move away from seeking perfection in the first layer of logic and instead demand resilience in the total system. If we want to run a professional operation, we cannot rely on someone else’s "black" box to be our only point of failure.

What it Costs: The Cost of Blind Optimization

When we fail to account for these dependencies, the costs aren't just "bugs"—they are tangible losses in production and safety.

First, there is the **cost of lost time**. When a logic-based system fails without an immediate local fallback, your operators become helpless. They cannot troubleshoot the code; they can only wait for a remote specialist to intervene. Every minute spent waiting for a "patch" or a "reboot" is a failure in our ability to maintain autonomy over our own production line.

Second, there is the **cost of hidden risk**. If an AI-driven system begins to drift—perhaps slightly lowering its standards for what constitutes a "pass"—and we don't have a secondary verification layer (like a periodic manual audit or a secondary sensor), that defect flows into the next stage of production. We only find out it was a problem when the customer complains, not when the part leaves the station.

Finally, there is the **cost of eroded trust**. When an automated system fails and the "fix" takes hours because no local workaround exists, the operators on the floor lose confidence in the technology. They begin to view the AI as a nuisance rather than a tool, leading them to find their own workarounds—often manual ones that bypass our safety protocols entirely.

The Framework: Building Operational Immunity in AI-Dependent Systems

To manage this, we must treat these models like any other high-risk component. We need to build "Operational Immunity" by wrapping the logic in physical and procedural safeguards.

1. Establish a Buffer Layer

Never allow an AI output to be the sole trigger for a critical action. If an AI flags a part as "Good," it should pass through a secondary, simpler validation gate—this could be a mechanical check, a different sensor type, or a human sign-off on a sampled basis. We are creating a buffer so that if the logic drifts, the defect is caught by the next layer of the process before it leaves our control.

2. Implement Multi-Model Redundancy

If your operation depends on a specific AI model for a critical function (like sorting or path planning), you must have a secondary "primitive" method ready to go. This isn't a second AI; it’s a simplified, rules-based logic system that can be toggled on if the primary complex system fails. Think of it as a manual transmission: if the automatic shifts fail, we need a way to keep moving forward using simpler mechanics.

3. Localized Fallback Protocols

Every station utilizing an AI component must have a "Degraded Mode" procedure. If the software hangs or the output becomes inconsistent, there must be a physical switch or a clear SOP that allows the operator to bypass the automated logic and complete the task using manual steps. This ensures that even if the "brain" of the machine is confused, the hands on the floor can still keep the line moving.

Practical Takeaways: Three Things to Audit Next Week

Don't wait for a system failure to see if your team knows what to do. Walk the floor and audit these three specific areas:

1. **Identify the "Single Points of Logic":** Map out every area where an automated decision (AI or otherwise) is the *only* thing determining the flow of material. If there is no second check, that is a high-risk point of failure.
2. **Verify the Local Fallback:** Pick one AI-integrated station and ask the operator: *"If this screen freezes or gives you an error right now, what do you do to keep moving?"* If their answer involves calling a remote help desk or waiting for an IT ticket, your process is not currently resilient.
3. **Audit the "Why" of Success:** During your next review of automated results (like vision system logs), don't just look at the hit rate. Ask to see the samples that were flagged as errors. If you can’t tell why the machine made a mistake, you have no way of knowing when it will start making those same mistakes on "good" parts.

The Last Checkpoint

We are entering an era where our machines aren't just failing because they break; they are failing because they get confused. Our job as leaders is to ensure that even when the logic fails, the operation continues. We don't have to understand every line of code to know how to manage a risk—we just need to make sure we never let an abstraction become our only point of failure.

Call to Action

Where is your organization most dependent on unverified logic? Share this diagnosis with a colleague who needs it. newsletter@sixsigmakaizen.com

Newsletter replies: newsletter@sixsigmakaizen.com · X: @kaizen_6sigma

Stay curious, stay improving,

David Rodgers

SixSigmaKaizen.com

References: Source: When AI becomes a supply chain component.