When Validation Proves It Works - But Not That It's Right

Why the Industry's Standard Validation Processes Are Systematically Blind to Data Architecture Quality

Jonathan Slavic | Circuit Check Inc.

Abstract

Validation is the industry's safety net — the formal process designed to catch problems before production. In hardware test programs, this takes two distinct forms depending on the organization. For contract manufacturers receiving development code directly, validation is often informal: development engineers review their own work, deem it acceptable, and hand it over. For OEMs who invest in a formal rewrite of development code into a stable, locked-down manufacturing environment, validation is the opposite extreme — extensive, documented, and expensive.

Both approaches share a common blind spot. Neither confirms whether the test software's data architecture supports manufacturing intelligence at scale. Validation, in both scenarios, confirms execution. It never asks whether the output can answer the questions manufacturing demands.

This paper examines how that blind spot develops, why it persists in both scenarios, and what it costs when production exposes what validation never examined.

1. The Safety Net That Has a Hole In It

In any hardware program, validation is supposed to be the moment where problems get caught. It exists precisely because software written under prototype conditions may not be ready for production demands. A proper validation process should surface gaps before yield targets, delivery schedules, and customer commitments are on the line.

The problem is that validation frameworks were built to answer a specific question: does the test system execute correctly and catch the failures it is designed to catch? That is a necessary question. It is not a sufficient one.

A second question goes almost universally unasked: does the data this system produces support the analytical demands of manufacturing at scale?

This paper tracks two scenarios through validation — from test code origin through factory deployment — and shows how both arrive at production with the same unexamined flaw, through entirely different paths.

> **The core problem:** *Validation proves the machine works. It does not prove the machine produces useful intelligence.*

2. Where the Code Comes From

To understand why validation misses architecture, it is necessary to understand how test software originates and how it reaches production. The path differs significantly between organizations, but the starting point is almost always the same.

Development engineers write test functions during DVT to validate early units under test. These are written for speed, flexibility, and real-time visibility. Loops handle multichannel iteration. Data is exported as it is generated — measurement executes, result appends to a file, loop continues. The structure mirrors execution flow, which is exactly what development needs.

What happens next is where the two scenarios diverge. But before they do, something else happens first — and it matters more than it appears.

The Decoder: A Workaround That Becomes Evidence

When development engineers live with flat file output long enough, they build a decoder. Typically an Excel workbook — sometimes a Python script — designed to load a single data file from a failing UUT and highlight problems, color-code anomalies, or point toward a root cause. It is clever, practical, and exactly the kind of tool a good engineer builds when the raw output is not immediately readable.

The decoder works beautifully in development. One file. One UUT. One engineer who built the tool and knows precisely what every column means. In that context it feels like a solved problem. The data is readable. Analysis is possible. The workflow makes sense.

What the decoder actually represents is an admission, written in Excel, that the raw output is not sufficient for analysis on its own. The engineers already knew the flat file was not enough — they solved it locally, for themselves, in a tool that was never designed to scale.

The decoder becomes dangerous not when it is used in development, but when it travels with the code. It gets handed to the CM as the analysis tool. It gets referenced during OEM validation as evidence that the data is analyzable. It delays — sometimes by months — the recognition that the underlying architecture cannot support production-scale intelligence. Because it works for one file, nobody asks whether it works for ten thousand.

> **The decoder test:** *If your analysis tool can only load one file at a time, it was built for debugging — not for manufacturing intelligence. The fact that it exists is proof the data architecture needs one.*

Contract Manufacturer (CM): The Direct Path

The development code is handed to a contract manufacturer with minimal or no architectural transition. The CM has zero control over the software's structure. They run what they are given, rely on the exported outputs, and escalate failures back to the OEM. The data architecture that was appropriate for 10 prototype units is now responsible for production volume.

OEM: The Rewrite Path

The development code is handed to test engineers who rewrite it in a stable, locked-down manufacturing environment — NI TestSTAND, LabVIEW, a controlled framework with version control, change management, and formal documentation. Real budget. Real engineering time. The result looks like hardened production code because it is, by every external measure, exactly that.

But the rewrite is a translation exercise, not an architectural redesign. The test engineers are focused on functional parity and tool correctness. The same execution-driven data structure from DVT is faithfully reproduced in a more disciplined environment. The tool changes. The data model does not.

> **The critical distinction:** *The CM inherits the structural flaw visibly. The OEM inherits it invisibly, behind the appearance of professional tooling. The flaw is identical. The risk of it going unexamined is higher at the OEM.*

3. Validation at the CM: Deemed Good

In the CM direct path, validation is informal by nature. There is no formal test plan for the test software itself. Development engineers — the same people who wrote the code — review it, run it against a known-good unit, confirm it catches expected failures, and deem it acceptable for handover.

This is not negligence. It is a rational response to schedule pressure, resource constraints, and the implicit assumption that working code is production-ready code. In many programs, the development engineers are the most qualified people available to evaluate the software. The review is genuine.

The problem is structural, not motivational. The bar was set by the people who built the system, using the criteria they had in mind when they built it. Those criteria were development criteria: does it run, does it catch failures, does it produce output. No one asked whether the output structure supports manufacturing intelligence because no one was thinking about manufacturing intelligence when the code was written.

What Gets Tested

The validation confirms that the test executes without errors, that known-good units pass and known-bad units fail, that output files are generated, and that results are readable. This is exactly the right validation for a development environment.

What it does not confirm: whether the output can answer which channel is driving failures across a population, whether results can be correlated to firmware revision without manual reconstruction, whether a test station's drift is visible in the data, or whether a CM with no engineering support can extract actionable intelligence from the logs without OEM involvement.

The Handover

Code passes informal validation. It transfers to the CM. The CM runs it because that is what they were given and they have no authority to redesign it. The decoder comes with it — handed over as the analysis tool, often without documentation, maintained only in the memory of the engineer who built it. The data architecture that was never examined in validation is now the foundation of production data management.

When volume increases and the decoder can no longer keep up — when a yield drop requires population-level analysis across hundreds of files and the workbook was only ever designed for

one — the CM has no path forward. They escalate to the OEM. The OEM engineering team that wrote the decoder is now doing production support instead of development work. The cost of the unexamined architecture arrives, on schedule, exactly when it is least convenient.

4. Validation at the OEM: Rigorous, Expensive, and Aimed at the Wrong Target

The OEM rewrite path produces a very different validation experience. Because the software now lives in a controlled manufacturing environment, changes require formal processes. Validation is not optional — it is mandated by the quality system. And it is done with genuine rigor.

The Validation Apparatus

A typical OEM rewrite validation involves multiple units under test run across the full test sequence, Gauge R&R studies to confirm measurement repeatability and reproducibility, MSA analyses to quantify measurement system variation, formal test reports documenting results, and sometimes material consumption for destructive test validation or environmental cycling. This represents significant engineering time, resource allocation, and in some cases actual hardware cost.

The reports are thorough. They document which units ran, what results were observed, whether the new sections executed correctly, and whether the system performed without introducing new failures. The conclusion — almost universally — is that the system ran cleanly and the new sections worked.

What the Reports Do Not Ask

The validation framework is built around a binary question: did it work? The reports confirm execution. They confirm measurement stability. They confirm that the rewrite introduced no regressions.

The decoder gets rewritten too — or at minimum validated alongside the test software. It loads a file. The results look right. The highlights appear in the correct cells. The validation report notes that data analysis was confirmed. What it does not note is that the analysis tool was designed for one file, one engineer, and one Tuesday afternoon debugging session. The fact that it still works in the validation environment is not evidence that it supports manufacturing intelligence. It is evidence that it was faithfully reproduced alongside the architecture it was built to compensate for.

They do not confirm whether the data structure supports yield trend analysis across a production population. They do not confirm whether channel-level failures are distinguishable in the output without post-processing. They do not confirm whether a quality engineer can extract a Pareto of failure modes from six months of production data in under an hour.

These questions were not part of the original development criteria, so they were not part of the rewrite criteria, and they are not part of the validation criteria. The validation is internally consistent. It simply validates the wrong scope.

> **The hard truth:** *A Gauge R&R proves the measurement system is repeatable. It says nothing about whether the data model allows that measurement to be correlated across 10,000 units, 6 test stations, and 4 firmware revisions.*

5. The Revalidation Trap

In the CM path, fixing the data architecture after production exposes the gap is painful. It requires code changes, CM coordination, and a new informal validation cycle. Disruptive, but manageable.

In the OEM rewrite path, the same fix is a controlled change event. The manufacturing environment that made the rewrite look professional is the same environment that makes correcting it expensive.

The Cost of Fixing What Validation Missed

A data architecture improvement in a locked-down manufacturing environment triggers the full quality apparatus. The change must be documented, reviewed, and approved. The updated software must be revalidated — which means re-running the Gauge R&R, regenerating the MSA reports, consuming engineering time, and in some cases consuming hardware. The reports must be archived. The CM must be notified and may require their own acceptance process.

None of this effort is directed at the root cause. It is the overhead cost of correcting something that should have been designed correctly in the first place — now multiplied by the formal rigor of the environment it lives in.

The Compounding Effect

The longer the flawed architecture runs in production, the more expensive the correction becomes. Historical data accumulated under the old structure may not be compatible with the new model. Migration requires engineering time. Trend continuity is broken. Reports generated under the old structure cannot be directly compared to reports generated under the new one.

The blind spot in validation does not just allow the problem to enter production. It locks the problem in place behind a quality system that was designed to ensure stability — and does exactly that, for the wrong architecture.

6. The Government-Locked Scenario

A third variant exists beyond the two primary scenarios, and it represents the most extreme version of the constraint. In some programs — particularly defense and government contracts — test software is not simply inherited from a development team. It is delivered as a program artifact, subject to contractual configuration control.

In this environment, the CM has no authority to modify the test software at all. Changes require program office involvement. Revalidation may require government witness or formal data package submission. The data architecture delivered with the program is, for practical purposes, permanent for the life of the contract.

If that architecture does not support manufacturing intelligence, the factory operates without it indefinitely. There is no informal fix, no internal change process, no budget line for improvement. The validation that accepted the original software — conducted by the program office or the prime contractor — determined the data architecture for the entire production run, whether or not data architecture was ever on the validation checklist.

7. What a Data Architecture Review Gate Would Look Like

The validation frameworks in both scenarios are not broken. They answer the questions they were designed to answer. The gap is that no one added the right questions.

A data architecture review gate does not require replacing existing validation processes. It requires adding an explicit checkpoint, separate from functional confirmation, that asks whether the test output supports production intelligence.

The Questions the Gate Should Answer

Before test software is approved for production — whether through informal handover in Scenario A or formal validation in Scenario B — the following should be answerable from the data alone, without post-processing or manual reconstruction:

1. Can a failure be attributed to a specific channel, test step, and unit identity without ambiguity?
2. Can production yield be trended by firmware revision, test station, operator, and date without joining multiple files?
3. Can a test station's measurement drift be identified from historical data without manual reconstruction?
4. Can a Pareto of failure modes across a production population be generated in under an hour by someone who did not write the test software?
5. Can all of the above be answered by the CM without OEM engineering involvement?

If the answer to any of these questions is no, the data architecture is not production-ready, regardless of whether the test executes correctly.

> **The standard:** *Working test code and production-ready data architecture are not the same thing. Validation should confirm both.*

8. The Cost of the Blind Spot at Scale

The consequences of unexamined data architecture are not hypothetical. They appear predictably, at predictable stages of production, in predictable forms.

Early production yield drops require root cause analysis. With execution-driven flat file logs, that analysis requires engineering time to reconstruct what the data should have organized automatically. Hours become days. In a CM environment, the burden escalates back to OEM engineering, consuming resources that were not budgeted for production support.

Process changes — fixture updates, firmware revisions, line rebalancing — require before-and-after comparison. Without structured data that captures the context of each test result, that comparison requires manual data extraction and reconstruction. The comparison that should take a query takes a project.

Sustained production surfaces measurement drift, station-to-station variation, and operator effects. These are normal manufacturing phenomena, manageable when visible in the data. With execution-driven logs, they are invisible until they cause failures — and by then, the population of affected units may already be significant.

Every one of these costs was introduced at validation, when the data architecture was accepted without examination.

9. Conclusion

The validation processes the industry uses — informal at the CM, rigorous at the OEM — are not the problem. They do what they were designed to do. The problem is the scope they were designed to cover.

Confirming execution is necessary. Confirming that a test catches failures is necessary. Neither is sufficient if the data those tests produce cannot support the analytical demands of manufacturing at scale.

At the CM, the informality of validation means the blind spot is easy to understand, if not easy to fix. At the OEM, the rigor of validation creates a more dangerous condition: an expensive, well-documented, quality-system-backed confirmation that the wrong architecture is production-ready. The cost of correcting it is then compounded by the very apparatus that certified it.

A data architecture review gate — a small addition to existing validation processes — closes the blind spot before it enters production. The questions it asks are not complex. They require no new tools, no new processes, no new headcount. They require only the recognition that working code and intelligent data are different standards, and that production deserves both.

The transition from prototype to production does not begin at the factory. It begins in the structure of the first line of test code — and it must be confirmed before that code is ever called production-ready.

About the Author

Jonathan Slavic is a Customer Solutions and Analytics Architect with over 25 years of experience in hardware test engineering, manufacturing systems, and production data intelligence. He has implemented WATS-based platforms at global scale and across complex mixed environments — greenfield, brownfield, and government-controlled programs — giving him a perspective on manufacturing data architecture that is grounded in real factory conditions, not theory.

In his role at Circuit Check Inc., Jonathan works directly with customers to design the bridge between test infrastructure and actionable production intelligence — helping organizations move from data that records what happened to data that drives decisions. Circuit Check Inc. is the leading fixture and test system manufacturer in the United States, and partners with WATS to bring structured data architecture to production environments that need more than execution confirmation.

For inquiries about test data architecture, WATS integration, or manufacturing intelligence strategy, contact Circuit Check Inc.