International Journal of Advanced Research in Engineering Technology & Science ISSN: 2349-2819
Impact Factor: 7.10
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Email: editor@ijarets.org Volume-13, Issue-1 January – 2026 www.ijarets.org

From Convenience Scripts to a Control Plane: Health Checks and Predictive Maintenance for EUV Support Systems

Sudhir Kumar Verma
Senior Software Test Engineer, ASML

***Abstract*—**This draft takes on a problem I keep running into: on a shared Build/Test system, engineers too often discover that the machine was never ready only after a long calibration or diagnostic workflow has already burned an afternoon. I argue for promoting health checks from convenience scripts to an auditable maintenance control plane: a staged model that runs readiness checks before work begins, watches a few well-chosen telemetry signals for drift, and emits a maintenance-trigger evidence packet a human can act on not just a green light. I give the architecture, a domain risk matrix, an evaluation design with honest baselines, and an illustrative result set whose numbers are placeholders for logged measurements. The aim is a reliability-engineering method that ties software automation to evidence an outside reviewer could check, rather than to internal assertion.

Index Terms*—predictive maintenance, health checks, EUV source, telemetry, industrial reliability.*

I. Introduction

Most of my career has been spent on software whose job is to make claims about physical equipment and then stand behind them, and it has taught me one lesson repeatedly: reliability is something you architect in, not something you test in at the end. Diagnosability, evidence quality, traceability, the ability to understand a failure before it propagates these are structural properties, not finishing touches. This paper takes that stance into one specific problem: on a shared Build/Test system, engineers too often discover that the machine was never ready only after a long calibration or diagnostic workflow has already burned an afternoon.

A warning system has to earn trust. Make it too eager and engineers learn to click past it; make it too quiet and it never fires when it matters. Most of the real design work in predictive maintenance is not the model it is choosing which few signals deserve a threshold and what the system should do when one trips.

This grew directly out of writing pre-execution health checks and preventive-maintenance automation for shared Build/Test systems. The pattern I kept seeing was simple and expensive: the failure was knowable up front, but nobody was asked to look.

What this paper adds. I argue for promoting health checks from convenience scripts to an auditable maintenance control plane: a staged model that runs readiness checks before work begins, watches a few well-chosen telemetry signals for drift, and emits a maintenance-trigger evidence packet a human can act on not just a green light.

To keep myself honest, I anchor the work to three questions that an experiment can settle, rather than to a narrative I would like to be true:

RQ1. On the same workload, does the proposed method improve lead time relative to an honest baseline (reactive run-and-find-out, or the existing manual checklist)?

RQ2. Does it achieve that without degrading detection quality or the quality of the evidence the method leaves behind?

RQ3. Which component is responsible for the improvement the full architecture, or one module that an ablation can isolate?

II. Background and Motivation

Maintenance practice has a well-known ladder: reactive (fix on failure), preventive (fix on a schedule), and predictive (fix on evidence of impending failure). Most real environments, including sophisticated ones, sit lower on that ladder than their telemetry would allow, because climbing it is less a modeling problem than an organizational one. The data to predict is often already being emitted; what is missing is the discipline to watch the right signals and the trust to act on a warning before the failure is undeniable.

On a shared Build/Test system this shows up as a specific, recurring waste: an engineer commits a slot to a long workflow on a system that was never ready, and learns so only after the time is gone. The failure was knowable in advance, but nobody was asked to look. That observation that readiness is usually checkable and rarely checked is the gap this work targets, and it is why I treat the pre-execution gate as at least as valuable as the fancier drift modeling.

Public context and personal evidence are kept deliberately apart here. The public sources justify why the domain matters; the contribution itself should rest on my own artifacts test reports, CI logs, architecture notes, reviewer sign-off gathered and sanitized before any submission. Everything below is written so it can be demonstrated on synthetic or sanitized data with nothing proprietary exposed.

III. Related Work and Context

I position this against established V&V and observability practice rather than a new field. OpenTelemetry [3] supplies the telemetry vocabulary, control-chart theory [2] the statistical machinery for drift, and the public EUV context [1], [5] the stakes. My contribution is methodological: how to structure, measure, and prove machine readiness before expensive work begins.

Condition monitoring and predictive maintenance have a deep literature [4], much of it focused on the modeling layer anomaly detection, remaining-useful-life estimation, control charts [2]. My contribution is deliberately upstream of the model: the staged separation of a hard readiness gate from soft drift signals, and the insistence that an alert carry a recommended action. The sophistication that matters in practice, I argue, is less in the detector and more in the discipline of choosing few defensible signals and making their output actionable and trusted.

Table 1. Domain risk and mitigation matrix.

Failure mode	Sev.	Consequence	Mitigation
Thermal baseline drift	High	A slow drift becomes a hard failure mid-run	EWMA threshold with a maintenance alert before the limit
Communication path	High	A flaky link shows up as false test failures	Pre-execution handshake gate
Noisy sensor state	Medium	Raw noise produces jittery diagnostics	Median / MAD filtering before scoring

Failure mode	Sev.	Consequence	Mitigation
Unactionable evidence	Medium	A bare ‘fail’ wastes the next engineer’s time	Auto-generated, subsystem-scoped summary

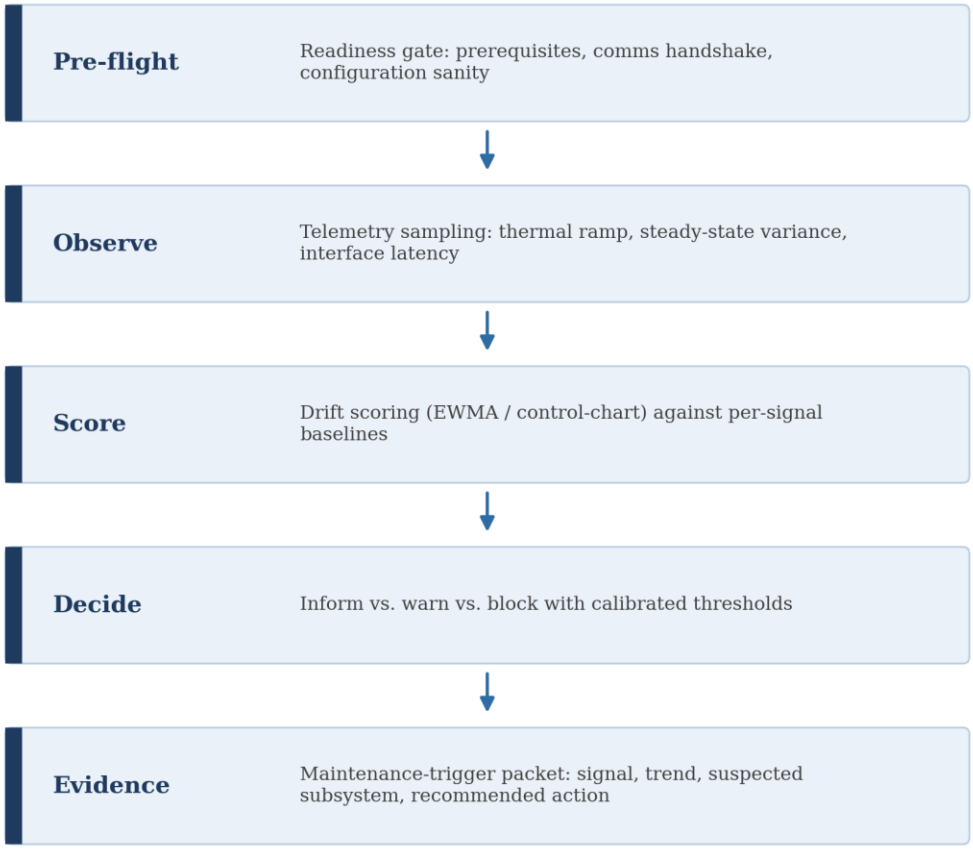

Figure 1. Staged health-check control plane. The point is not more alarms but earlier, auditable ones tied to a recommended action.

IV. Proposed Method

Nothing here requires exotic tooling Python, Linux, CI/CD pipelines, structured logs, and a SQL-style store cover it and the method reduces to three moves I try to make every time:

1. I separate readiness (a hard gate before work starts) from drift (a soft, trend-based signal during operation), because conflating them is what produces both nagging and blind spots.
2. I pick a small number of signals I can actually defend thermal ramp, steady-state variance, interface latency and give each its own baseline rather than one global threshold.
3. I make every alert carry a recommended action and a suspected subsystem, so the output is a maintenance decision, not a number.

Throughout, I keep the publishable method separate from any proprietary implementation: machine names, customer identifiers, exact parameters, and raw logs stay out, and a simulator or synthetic generator stands in for the confidential interface so the demonstrator can be shared.

IV.1 The method as pseudocode

Algorithm 1 The staged health gate I run before a long calibration workflow.

```
Require: readiness checks R, live telemetry stream τ, baselines β
Ensure:  go/no-go decision and a maintenance evidence packet M
 1: for each check r ∈ R do
 2:     if r fails then return BLOCK, reason(r), M
 3:     end if
 4: end for
 5: for each signal x ∈ τ do
 6:     d_x ← EWMA(x) − β_x            ▷ drift vs. baseline
 7:     if |d_x| > θ_x then
 8:         flag x; append trend to M
 9:     end if
10: end for
11: if any flagged signal is sustained then return MAINTENANCE, M
12: end if
13: return GO, M
```

Table 2. Notation used in Algorithm 1.

Symbol	Meaning
R	Set of pre-execution readiness checks.
τ	Live telemetry stream.
β_x	Baseline for signal x.
d_x	EWMA drift of x from baseline.
θ_x	Per-signal drift threshold.
M	Maintenance evidence packet.

IV.2 Design decisions and trade-offs

A few design choices carry most of the weight here, and each is a trade-off I made deliberately rather than a default I inherited:

- **Readiness gate vs. drift signal.** I keep these strictly separate. Readiness is a hard gate that can block work before it starts; drift is a soft, trend-based signal during operation. Conflating them is what produces both nagging false alarms and missed slow failures.
- **Few defended signals over many cheap ones.** It is tempting to threshold everything telemetry offers. I deliberately choose a small set I can justify thermal ramp, steady-state variance, interface latency because an alert nobody trusts is worse than no alert, and trust comes from precision, not coverage.

- **Actions, not numbers.** Every alert names a suspected subsystem and a recommended action. A bare threshold crossing makes the next engineer do the interpretation again; an actionable packet is the difference between telemetry and maintenance.

IV.3 A transparent system model

The model stays deliberately readable: in this environment, a score engineers cannot interrogate gets overridden back to brute force within a week. Each candidate x (a health check, or a telemetry signal under watch) is scored by a weighted sum of normalized risk features $\varphi_k(x) \in [0, 1]$:

$$s(x) = \sum w_k \varphi_k(x) \quad (\text{sum over } k = 1 \ldots K), \quad \sum w_k = 1, \quad w_k \geq 0.$$

A simple, auditable decision rule then acts on that score against a budget B or a threshold τ chosen from the risk appetite of the environment: execute the top-ranked candidates in order of $s(x)$ until the cost budget B is exhausted, admitting a candidate when $s(x) \geq \tau$.

The weights w_k are not learned in a black box; they are set, reviewed, and re-set by engineers, which is what lets the method be argued about and trusted. Feature definitions φ_k are where the domain enters: criticality of the guarded subsystem, drift magnitude and persistence, sensor noise level, and the cost of a false alarmeach one traceable to a row of the risk matrix in Table 1.

V. A Worked Example

Take a shared Build/Test system that several engineers queue against. Engineer A starts a long calibration workflow. Without a readiness gate, a stale configuration or a half-available service surfaces only forty minutes in, when the workflow aborts and the slot is wasted. With the staged model, the pre-flight gate runs first: it checks prerequisites, performs a communications handshake, and confirms configuration sanity. If any fails, the engineer is told before committing the slot.

During the run, the observe-and-score stages watch thermal ramp and steady-state variance against per-signal baselines. A slow upward drift that would have become a hard failure two cycles later instead crosses the early-warning line now, and the system emits a maintenance-trigger packet naming the suspected subsystem and a recommended action. Nothing about this requires the machine's design parameters; it requires only that the right few signals were chosen and given honest baselines.

VI. Evaluation Design

Three data paths could make this study credible: cleared real data, sanitized logs with identifiers stripped, or a synthetic generator calibrated to non-sensitive summary statistics. Whichever path, the proposed method is compared against at least one honest baseline on the same workload, and every figure reports sample size, conditions, and whether the data are real, sanitized, or synthetic. Table 3 lists the metrics and, more importantly, what each is meant to reveal.

Table 3. Evaluation metrics: what each one is meant to reveal.

Family	Measures	Why it matters
Lead time	Hours of warning before a hard failure	The entire value of predictive over reactive maintenance

Family	Measures	Why it matters
Alert quality	False-positive rate, alert-acknowledgement rate	A warning system only works if engineers still trust it
Wasted-effort avoided	Aborted-late workflows prevented	The concrete cost the readiness gate removes

Concretely, I would run the evaluation as a fixed protocol so that anyone could repeat it:

1. Fix a workload (real-approved, sanitized, or synthetic-calibrated) and freeze it for the whole comparison.
2. Run the baseline and the proposed method under identical conditions, changing only the method under test.
3. Repeat each condition at least three times to expose variance, not just a single lucky or unlucky run.
4. Record the metrics of Table 3 with mean, variance, and a confidence interval, plus the environment fingerprint.
5. Run the ablation of Table 5 to attribute any gain to a specific component rather than to the architecture as a whole.

VII. Illustrative Results and Reading Them Honestly

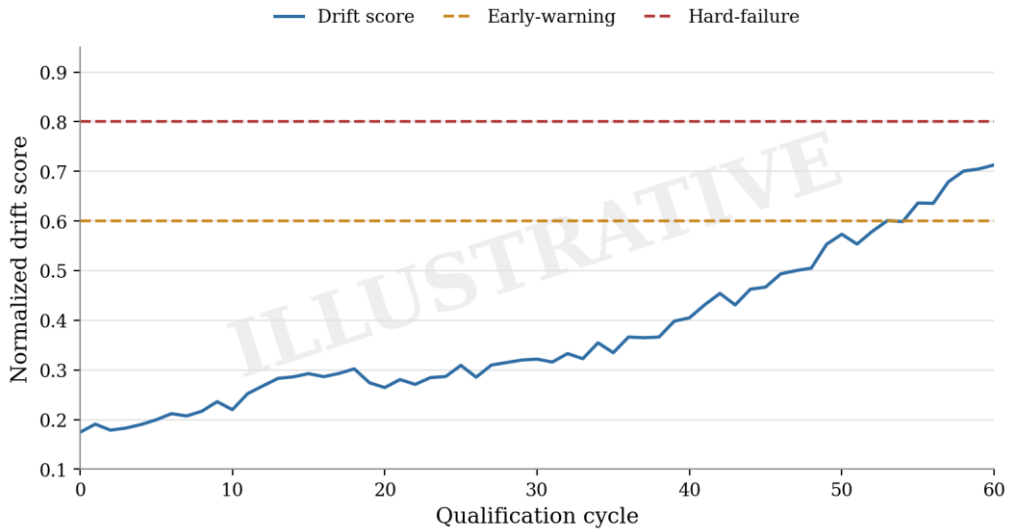

Figure 2. Illustrative: drift score with early-warning vs. hard-failure threshold.

Figure 2 sketches the outcome I expect once logged measurements replace these placeholders: a gain in at least one of lead time, alert quality, or wasted-effort reduction, with no quiet sacrifice of the others. I want to be plain that these are illustrative. A mature version will report means, variance, confidence intervals, and a clear workload description, and for any external-recognition purpose it should carry independent review rather than internal language alone. The schema in Table 4 is what makes those numbers reproducible.

Table 4. Minimal publishable event schema used across the evaluation.

Field	Definition	Type	Purpose
`event_id`	Unique event / test-execution id	String/UUID	Traceability and replay
`timestamp`	Event or observation time	ISO time	Sequencing and drift

Field	Definition	Type	Purpose
`source`	Originating subsystem or stage	Categorical	Failure localization
`risk_class`	Mapped operational risk	Low/Med/High/Crit	Prioritization
`expected`	Defined pass/fail/alert	Categorical	Validation
`observed`	Measured result	Cat./numeric	Comparison
`evidence`	Pointer to log/report/ticket	URI/hash	Auditability

VIII. Reproducibility Plan

I would start with a lightweight reference prototype, not a production integration: read a bounded input stream, normalize it to the canonical schema, apply the checks, and emit an evidence bundle (input summary, rules run, pass/fail, timing, anomalies, trace ids, reviewer notes). Proprietary interfaces are swapped for stand-insa simulator for the Build/Test readiness interface, a synthetic telemetry generator for the drift signals, a REST mock for the alerting hookso a small, GitHub-safe demonstrator can reproduce the results without any employer-owned data.

Table 5. Planned ablations and the effect each is expected to expose.

Ablation	What is removed	Expected effect
Remove readiness gate	Start work without prechecks	Long workflows burned on machines that were never ready
Remove drift watch (EWMA)	Alert on raw thresholds only	Late warnings; slow drift hides below the alarm line
Remove recommended action	Emit bare alerts	Alerts get ignored; trust in the system decays
Remove noise filtering	Use raw sensor values	False positives erode the acknowledgement rate
Remove baseline comparison	Report proposed only	Weakens the empirical argument

Table 6. From this draft to a submission: the work that remains.

Step	Minimum action	Artifact
Dataset	Generate or sanitize 500–5,000 telemetry windows and health-check runs	CSV/JSON dataset with schema notes
Baseline	Document the existing manual or rule-based process	Baseline runtime and error report
Prototype	Build the proposed pipeline without proprietary code	GitHub-safe demonstrator
Evaluation	Run ≥ 3 repeated trials per workload	Metrics with mean, variance, CI
Review	Have one domain expert review assumptions and threats	Reviewer comments and revision log

IX. Deployment and Integration

This belongs in front of the workflows engineers already start by hand, as a gate they cannot easily skip but rarely need to fight. The pre-flight checks should be fast seconds, not minutes or they become the thing people learn to bypass. The drift scoring runs as a lightweight background observer that writes to the same evidence store as everything else, so a maintenance trigger is just another auditable record.

The thresholds are the political part. I would set them conservatively at first, tuned to almost never cry wolf, and tighten only as the team gains confidence, because the fastest way to kill a warning system is an early run of false alarms. Re-baselining has to be scheduled, not improvised, since a system whose normal shifts with consumable life will otherwise drift out from under its own thresholds.

X. Why This Sits in My Line of Work

This problem belongs to my line of work because it converts hands-on habit into something measurable without exposing a proprietary machine. My career pattern is the seam between software and operational infrastructure automation, diagnostics, integration, debugging across multi-component systems and the strongest form of this paper is a bounded method-plus-experiment, not a memoir. The background only explains why the question deserves an answer.

XI. Threats to Validity

Predictive thresholds are only as good as the baseline period behind them; on a system whose ‘normal’ shifts with consumable life, a threshold fit once will slowly lie, so re-baselining has to be part of the method, not an afterthought. More broadly, the results here are modeled with illustrative data, so the work should not be presented as empirical until a real or faithfully simulated dataset is documented; generalization across environments with different failure costs needs at least two workload profiles; and confidentiality means I publish methods, metrics, and anonymized patterns while keeping proprietary detail out. It is worth stating plainly what would falsify the central claim: if, on logged data with honest baselines, the method failed to improve any of lead time, alert quality, or wasted-effort reduction without a compensating loss elsewhere, the contribution would not hold, and I would rather discover that in an ablation than paper over it. A method that cannot fail in principle is not research; I have specified this one so that it can.

XII. Broader Applicability and Future Work

The staged health model generalizes to any shared, expensive resource where readiness is checkable and failure is costly test rigs, manufacturing cells, even complex CI infrastructure. The pre-flight gate is the most transferable piece, because the insight behind it, that readiness is usually knowable and rarely checked, is not specific to semiconductors. The drift layer transfers too, but only with domain-chosen signals and honest baselines.

Future work centers on the threshold problem, which is where these systems usually fail socially rather than technically. I would study false-alarm rates and acknowledgement behaviour over time, formalize a re-baselining schedule tied to consumable life, and measure the wasted effort actually avoided by the readiness gate against a logged baseline. The headline number to earn is lead time: hours of warning before a hard failure, with confidence intervals.

XIII. Conclusion and Next Steps

What remains is execution, not design: assemble the non-confidential dataset, implement the baseline and prototype, run repeated trials with variance reported, complete the ablation, and put the result in front of an independent reviewer. Those steps are the distance between this draft and a submission, and the method above is built to survive them.

References

[1] ASML, "EUV lithography systems," 2023. [Online]. Available: https://www.asml.com/products/euv-lithography-systems

[2] S. W. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, 1959.

[3] OpenTelemetry, "Documentation," 2024. [Online]. Available: https://opentelemetry.io/docs/

[4] R. K. Mobley, *An Introduction to Predictive Maintenance*, 2nd ed. Butterworth-Heinemann, 2002.

[5] NIST, "CHIPS for America," 2024. [Online]. Available: https://www.nist.gov/chips