A TECHNOLOGICAL AND INTELLECTUAL HISTORY

History of Hardware Diagnostics, Observability, and Debugging

From sensory inspection, checking circuits, and test points to built-in self-test, remote management, digital twins, and AI-assisted fault isolation

— evidence, causality, and the changing craft of explanation —

Concept and direction by Dmitry Vostokov, DumpAnalysis.org

Developed with AI assistance (GPT-5.6 Sol Max and Opus 5 Extra) and checked against the sources listed in this article.

July 2026

Historical synthesis with 111 references, chronology, glossary, and evidence taxonomy

Abstract

Hardware has always failed through mechanisms that exceed what an operator can see directly. A stopped shaft, an open cable, a drifting vacuum tube, a marginal memory cell, a cracked solder joint, and a poisoned telemetry path produce different symptoms, yet each creates the same investigative problem: how can hidden physical condition be converted into trustworthy evidence? The history of hardware diagnostics is therefore a history of instruments, checking structures, service practices, and models that make failure observable without pretending that observation is explanation.

This history follows those lineages from steam-engine indicators, electrical bridge measurements, test points, and maintenance logs through checking circuits, redundancy, automatic test equipment, scan design, built-in self-test, firmware error records, vehicle and aerospace fault management, condition monitoring, out-of-band management, and fleet-scale telemetry. It treats debugging as controlled interaction with a device, diagnostics as inference from evidence, observability as a designed capability, and prognosis as a distinct claim about future condition.

The central argument is that hardware observability did not replace inspection, testing, or fault isolation. It expanded an older evidence system across scales: from a mechanic's senses to calibrated instruments; from an external probe to on-chip monitors; from a single failed unit to population statistics; and from manual troubleshooting to model- and AI-assisted diagnosis. At every stage, the decisive questions remain physical and epistemic: what changed, where did it change, when did it become detectable, which layer reported it, and what evidence would distinguish cause from consequence?

How to read this history

This article is a historical synthesis rather than a catalogue of products. Dates identify documented instruments, publications, standards, systems, and investigations, but many practices developed in parallel and were transmitted through manuals, field service, military and aerospace programs, manufacturing, and laboratory craft. Claims of priority are therefore used sparingly. The aim is to explain changing diagnostic capabilities and investigative habits.

The chapters alternate between technologies and practices because hardware evidence is always mediated. An oscilloscope trace depends on bandwidth, grounding, triggering, and probe loading. An ECC syndrome depends on code design and address mapping. A baseboard-management alert depends on sensors, firmware, thresholds, time synchronization, and identity. The same artifact can be decisive, misleading, or silent according to how it was produced.

References in square brackets point to the source list. Primary standards, manuals, government reports, archival material, and peer-reviewed studies are emphasized. The case studies are not offered as

monocausal morality tales; they show how component behavior, instrumentation, maintenance history, operating context, and organizational interpretation combine in consequential investigations.

Neighboring practices and their primary questions

Practice	Primary question	Characteristic evidence
Inspection	What physical condition is visible or measurable now?	Wear, cracks, corrosion, discoloration, sound, heat, vibration, dimensions, images
Testing	Does the artifact conform or fail under specified stimuli?	Test vectors, limits, pass/fail records, environmental and stress results
Debugging	What changes when the device is probed, stopped, stimulated, or reconfigured?	Waveforms, traces, breakpoints, register state, controlled substitutions
Monitoring	Which selected conditions require awareness or action?	Counters, alarms, thresholds, trends, heartbeats, health states
Diagnostics	Which failure mechanisms and contributing conditions best explain the evidence?	Syndromes, logs, measurements, topology, configuration, comparative tests
Prognostics	How is condition likely to evolve, and how much useful life remains?	Degradation trends, load history, uncertainty bounds, physics or learned models
Forensics	What happened, in what sequence, with defensible provenance?	Preserved artifacts, timestamps, chain of custody, calibration and identity records

Table 1. Working distinctions used throughout this article; practices overlap in real investigations.

SEVEN RECURRING TRANSITIONS

- Sensory signs → quantified physical signals
- External instruments → embedded and on-chip self-observation
- Repair after failure → detection of degradation before failure
- Single-component fault finding → cross-layer system diagnosis
- Manual stimulus and substitution → generated test patterns and built-in self-test
- Local service records → fleet-scale population evidence
- Human fault isolation → model- and AI-assisted diagnosis with governed action

Contents

Seven parts · 35 chapters · three reference appendices

Table 2. Article contents and opening pages.

PART I

Foundations: making physical condition knowable

1. Vocabulary, scope, and historical method

Hardware diagnostics is easiest to misunderstand when inspection, testing, monitoring, debugging, prognostics, and forensics are treated as synonyms. Testing usually begins with a specification and a stimulus: does the unit conform? Debugging is interactive and interventionist: what changes if a signal is forced, a clock is slowed, a register is read, or a module is replaced? Monitoring watches selected variables during operation. Diagnostics interprets evidence to identify mechanisms and contributing conditions. Prognostics estimates future condition. These practices overlap, but their primary questions and standards of proof differ.

Three lineages organize this history. The measurement lineage turns hidden physical variables into signals: pressure into a diagram, current into a deflection, vibration into a spectrum, junction temperature into telemetry. The checking lineage introduces redundancy, test patterns, assertions, codes, and self-tests that distinguish acceptable from anomalous behavior. The service lineage preserves configuration, symptoms, repair actions, and population experience so that a fault can be reproduced, localized, and prevented. Modern observability combines all three.

The familiar chain fault → error → failure remains useful if applied carefully. A fault is a hypothesized or established cause, such as a crack, an alpha-particle strike, a missing termination, or a design defect. An error is an incorrect internal state, such as a flipped bit or corrupted transaction. A failure is externally visible loss or deviation of service. The dependability taxonomy developed by Avizienis and colleagues formalized these distinctions and also warned that faults may be physical, design-related, malicious, transient, or permanent.[19] Hardware diagnosis therefore reasons across both matter and representation.

Historical evidence is uneven. Instruments survive in museums; field notebooks and service bulletins often do not. Standards document intended interfaces, not necessarily operational use. Accident reports reconstruct a selected sequence under legal and institutional constraints. A responsible history separates contemporary documentation from retrospective interpretation and treats every diagnostic artifact as a partial observation shaped by its capture path.

2. Before electronics: sensory inspection, indicator diagrams, and maintenance records

Long before electronic instrumentation, maintainers diagnosed machines through heat, odor, sound, touch, motion, leakage, wear, and the appearance of fracture surfaces. These were not merely intuitive impressions. Skilled observation depended on a learned baseline: the normal cadence of a pump, the expected temperature of a bearing housing, the color of combustion, or the feel of backlash. The human body was the first multi-modal sensor suite, but its readings were difficult to compare across observers, time, and sites.

The steam-engine indicator converted one hidden variable into a durable record. A piston exposed the instrument to changing cylinder pressure while a moving surface recorded pressure against piston travel. The Watt indicator was introduced by Boulton and Watt in the late eighteenth century to assist adjustment and evaluate engine work; later spring indicators, including the Richards design, improved portability and response.[1, 2, 3] The diagram mattered because it preserved a cycle that the eye could not follow directly and made differences between cylinders, loads, and valve timing comparable.

Indicator practice established several durable ideas. First, measurement needs a reference axis and calibration. Second, a useful trace relates an internal variable to system phase, not merely to clock time. Third, the instrument has dynamics of its own: friction, spring response, tubing volume, and mounting can distort the event. Fourth, interpretation depends on a model of normal operation. A pressure loop did not announce “late valve timing” by itself; an engineer inferred that condition from shape, phase, construction, and context.

Maintenance records supplied temporal depth. Running hours, lubrication, inspections, replacements, and recurring symptoms formed an early event history. Such records were often local and inconsistent, yet they introduced a principle that later fleet analytics would rediscover: a component’s present measurement is far more informative when joined to its identity, age, load history, environment, and prior interventions.

3. Telegraphy and electrical fault localization: bridges, continuity, and distance to fault

Telegraph and cable systems made diagnosis a problem of hidden condition over distance. An open conductor, short to ground, cross, insulation leak, or intermittent contact might lie kilometers from the test station and be inaccessible until its approximate location was known. Simple continuity and insulation tests classified the fault; bridge methods used resistance ratios and known cable properties to estimate

distance. Murray- and Varley-loop techniques became enduring examples of converting an inaccessible geometric location into an electrical measurement.[4, 5]

The reasoning pattern was explicit. Reconfigure the faulty conductor with a sound return path, apply a controlled source, balance a bridge, and infer position from resistance. The method depended on assumptions: uniform conductor resistance, known cable length, stable contact, a suitable sound conductor, and a fault resistance compatible with the technique. Temperature, joints, parallel paths, and multiple faults could bias the estimate. Diagnosis advanced not only through better meters but through recognizing when a measurement model did not fit the physical plant.

Electrical maintenance also normalized sectionalization. Technicians isolated spans, exchanged pairs, compared ends, applied tones, and performed tests from multiple points. The resulting workflow—classify, localize, expose, repair, and retest—became a template for electronics and computing. Modern time-domain reflectometry changes the stimulus and inference, locating impedance discontinuities from reflected pulses, but it preserves the same core move: create a propagating event whose return encodes distance and fault character.

Cable work highlights the importance of negative evidence. A clean test at one voltage, frequency, or temperature did not prove a line healthy under weather, mechanical stress, or service load. Intermittent faults forced investigators to correlate electrical readings with time, environment, and traffic. The lesson persists in buses and high-speed links: absence of an error during a short bench test is evidence about that test condition, not a universal property of the interconnect.

4. Meters, oscilloscopes, triggering, and the visibility of signals

Electrical meters made amplitude, resistance, and continuity portable. Oscillographs and cathode-ray oscilloscopes added time, exposing transient and periodic behavior that a pointer averaged away. Tektronix's postwar instruments helped make triggered sweep and calibrated laboratory oscilloscopes widely practical; the company's first products emerged from design work begun in 1946.[6] Hewlett-Packard similarly built a measurement culture around calibrated oscillators, analyzers, and later automated instruments.[8]

The oscilloscope did more than draw voltage. Triggering selected which repeated event would anchor the display; bandwidth and sampling limited what could appear; coupling removed or preserved DC content; probes changed capacitance, impedance, and ground paths. The instrument therefore taught a discipline of observation: every trace is a joint product of device, probe, acquisition settings, and display transformation. Modern primers still emphasize bandwidth, rise time, sample rate, record length, and

trigger configuration because incorrect settings can make a healthy signal look faulty—or erase the fault altogether.[7]

Test points and schematics linked measurement to design intent. A node label, expected voltage, waveform sketch, and tolerance transformed an anonymous conductor into an interpretable boundary. Troubleshooting manuals commonly proceeded from power and clock toward increasingly local signals. That order reflects causal reach: a failed rail can invalidate every downstream observation, while a wrong output at one logic stage may be only a consequence.

Instrument comparison introduced another diagnostic habit. Observe the same signal with different ranges, probes, time bases, loads, or reference channels. If the anomaly moves with the instrument configuration, the measurement path becomes a suspect. This is the physical analogue of changing logging level or sampler in software: observability mechanisms must themselves be debugged.

5. Early electronic computing: checkout, marginal testing, panels, and cross-layer faults

Early computers were rooms of exposed physical state. Lamps displayed registers and control conditions; switches deposited words and selected operations; oscilloscopes examined pulses; test programs exercised memory and arithmetic; maintenance teams listened for characteristic rhythms and inspected tubes, relays, connectors, and power supplies. The celebrated 1947 moth taped into the Harvard Mark II logbook documents a literal relay obstruction, but its significance lies in the surrounding practice: operators recorded the symptom, the found condition, and the restoration of service.[9]

Because hardware and program behavior were tightly coupled, checkout crossed layers. An incorrect sum might come from a program, a wiring error, a marginal tube, a timing path, or an unreliable memory location. Teams used known patterns, repeated operations, comparison with hand calculations, signal tracing, and module substitution. The Whirlwind computer's surviving history shows an engineering culture in which maintenance and operation were continuous, and even sound could reveal the machine's repetitive internal activity.[10, 11]

Marginal checking deliberately changed voltages, timing, temperature, or other operating conditions to expose components approaching failure. A machine that passed at nominal settings but failed under a small controlled margin was not yet broken in service, but it had revealed reduced tolerance. This is an early form of accelerated stress and robustness testing. It also exposes the ethical boundary of active diagnosis: the test must create information without causing the very damage it is meant to predict.

The era established four persistent requirements: make internal state visible; provide known stimuli and expected results; preserve a first-failure record before repair changes the system; and connect logical symptoms to physical replaceable units. Later systems would automate each requirement, but they would not eliminate the need to understand the path from physical fault to recorded evidence.

PART II

Reliability becomes an engineering discipline

6. Error-detecting codes, checking circuits, and self-checking machines

As computing moved from experimental apparatus to operational infrastructure, correctness could no longer depend only on periodic checkout. The machine needed checking structures that operated with the work. Parity added a redundant bit to detect selected data errors. Checksums, residue codes, duplicated logic, and comparison circuits extended the idea: transform the expected relation into hardware that can signal when the relation is violated.

Richard Hamming's 1950 paper showed how codes could not only detect but correct errors by arranging parity checks so that a syndrome identifies the erroneous position.[12] The conceptual advance was diagnostic as much as mathematical. Instead of one undifferentiated alarm, the pattern of failed checks carried location information. Contemporary ECC memory still follows this structure: data and check bits produce a syndrome, policy classifies the event as correctable or uncorrectable, and a handler records address and context—if the platform preserves them.

Self-checking circuits raised a harder question: who checks the checker? Redundant encoding, complementary rails, duplicated computation, and periodic self-test reduce the chance that a single latent defect produces an undetected wrong result, but coverage depends on the fault model. A checker can share power, clock, design, environmental, or specification faults with the logic it observes. John von Neumann's analysis of reliable organisms built from unreliable components and later work on modular redundancy made this dependence explicit.[13, 14]

Checking changed service behavior. Corrected errors allowed continued operation, but they also created pre-failure evidence. If corrections are counted only as success, degradation is hidden; if they are recorded with component identity and time, they can reveal a worsening channel or memory device. The distinction between correction and diagnosis remains fundamental: a code can restore a value without establishing why it was corrupted.

7. Redundancy, voting, fault containment, and graceful degradation

Redundancy provides alternative paths for service and alternative observations for diagnosis. Dual comparison detects disagreement but cannot always decide which side is correct. Triple modular redundancy adds a voter and can mask one discrepant result under its assumed fault model.[14] Standby

units, replicated channels, spare rows, redundant power, and mirrored storage apply the same idea at different scales.

The diagnostic value of redundancy is conditional. Two replicas that disagree create a symptom, but shared design, maintenance, environment, input, or power can make them fail together. A voter can be a single point of failure. Redundant sensors may be co-located in the same hazard. The important engineering concept became fault containment: define boundaries within which a fault can propagate, and arrange detection, isolation, and recovery so that damage does not cross them.

SAGE and later aerospace computers linked redundancy to operational continuity. Large systems used checking, duplicate paths, service procedures, and modular replacement because intermittent faults and maintenance time were system properties, not merely component statistics.[15, 16] Apollo guidance-computer documentation likewise treated self-checking and alarm behavior as part of the flight system, where a correct response could include restarting work while preserving essential state.[17, 18]

Graceful degradation broadened the objective from “never fail” to “retain the most important service under fault.” That requires a priority model: which function may be shed, which evidence must remain available, and which recovery action is safe? Redundancy is therefore not only duplicated hardware. It is an architecture of independence, detection, decision, and controlled reconfiguration.

8. Reliability engineering: FMEA, fault trees, burn-in, and environmental stress

Reliability engineering formalized failure as something to analyze before, during, and after operation. Failure modes and effects analysis proceeds component or function by component, asking how each mode could occur and what local and system effects would follow. Fault-tree analysis reasons from an undesired top event through combinations of causes. NASA guidance helped institutionalize these methods in complex programs, alongside reliability allocation, parts control, qualification, and corrective-action systems.[20, 21, 22, 23]

These methods are diagnostic designs on paper. They identify where detectors are needed, which faults must be distinguished, what evidence recovery requires, and where a common cause defeats redundancy. They also reveal an asymmetry: enumerating plausible faults is easier than proving completeness. Real failures frequently arrive through interactions, manufacturing variation, maintenance error, aging, or environmental combinations absent from the model.

Burn-in and environmental stress screening attempt to precipitate latent manufacturing defects before field deployment. Temperature cycling, vibration, humidity, voltage margin, and load expose weak bonds,

contamination, marginal timing, and other vulnerabilities. The familiar bathtub curve—early failures, a lower-rate useful-life region, and wear-out—is a population model, not a law for every technology; NIST's reliability guidance explicitly treats hazard shapes as model choices rather than universal truth.[24]

What physically fails

The phrase failure mechanism should name a physical process, not merely restate a symptom. Important families include fabrication defects and contamination; opens, shorts, cracked vias, weak bonds, solder fatigue, delamination, corrosion, tin whiskers, and connector fretting; electromigration, stress migration, dielectric breakdown, hot-carrier degradation, and bias-temperature instability; radiation upset and latch-up; retention, read-disturb, and wear mechanisms in memories; and mechanical fatigue, creep, wear, vibration, shock, humidity, thermal cycling, and overload. Electrical margin can also be lost through power droop, ground bounce, crosstalk, impedance discontinuity, clock jitter, skew, or marginal setup and hold timing.[107, 109]

Those mechanisms can converge on the same mode—an open, short, delay, wrong bit, reset, or intermittent loss of service—and one mechanism can produce several modes as stress and damage evolve. FMEA organizes modes and effects; physics-of-failure work connects material, geometry, load, environment, and time; laboratory failure analysis looks for the physical signature. Treating those layers separately prevents an error code or replaced FRU from being mistaken for root cause.

The discipline also shifted emphasis from a failed part to a closed corrective-action loop. Evidence from test and field returns should update design rules, screening, suppliers, thresholds, maintenance, and training. Historical surveys of fault-tolerant computing show how architecture, validation, and service practice evolved together rather than as independent specialties.[25] Without that loop, diagnostics merely accelerates replacement. With it, every investigated failure becomes information about the process that created and operated the hardware.

9. Field service culture: schematics, FRUs, substitution, and preventive maintenance

Field service translated design knowledge into repeatable action. Manuals provided block diagrams, signal expectations, adjustment procedures, fault trees, error codes, test points, and parts lists. The field-replaceable unit reorganized diagnosis around service economics: isolate far enough to replace a board, module, drive, fan, or supply within a target time, then repair the removed unit elsewhere if worthwhile.

Substitution with a known-good unit is powerful because it changes one boundary while holding the surrounding system approximately constant. Yet it can mislead. A replacement may differ in revision or

firmware, a connector may be disturbed, a marginal supply may tolerate one load but not another, and the act of reseating may temporarily cure corrosion or fretting. Good service records therefore distinguish the observed symptom, the diagnostic action, the removed part, the verification test, and the eventual confirmed failure mechanism.

Preventive maintenance assumed that cleaning, lubrication, calibration, adjustment, and scheduled replacement could reduce failure probability. Condition-based maintenance later challenged fixed intervals by asking whether measured degradation could justify action more precisely. Both approaches depend on an asset's identity and service history. A threshold without knowledge of age, duty cycle, environment, and sensor calibration can turn either strategy into ritual.

Field culture also produced informal knowledge: signature sounds, recurring connector faults, serial-number ranges, and repair folklore. Fleet databases and service bulletins attempted to make that tacit experience portable. The historical tension remains: structured codes scale, while free text preserves nuance; parts replacement restores service, while failure analysis establishes knowledge.

10. Mainframe RAS: machine checks, logout records, error logs, and service processors

Mainframes made reliability, availability, and serviceability a system architecture. IBM System/360 documented machine-check interruption as a controlled response to detected malfunction, with status that software could inspect.[26] Later systems expanded checking, retry, sparing, partitioning, error logging, and service access. The goal was not simply a more reliable component but a machine that could detect, contain, record, recover, and be repaired with bounded disruption.

A machine check is an interface between hardware evidence and system policy. Detection logic raises a condition; registers and logout areas capture syndrome, address, unit, and validity; firmware or the operating system decides whether to retry, offline a resource, terminate work, or stop. The record must say not only what was observed but which fields are trustworthy after the fault. IBM's RAS literature emphasized these interacting mechanisms and the importance of service information.[27, 28]

First-failure data capture became a formal objective. Once a handler retries a transaction, resets a channel, clears a latch, or replaces state, decisive evidence may disappear. Service processors created an increasingly independent observation and control plane that could preserve logs even when the host processor or operating system was compromised. This separation foreshadowed the BMC.

RAS also made maintenance a software-mediated workflow. Error logs could be analyzed for recurrence and thresholds; diagnostics could run online or during maintenance; units could be deconfigured; remote

support could inspect state. Tandem and Stratus systems demonstrated different combinations of process pairs, replicated components, failover, and online service.[30, 31] The same power introduced new risks: service interfaces needed versioning, authorization, and trustworthy identity. An observation path that can alter configuration is also a control path.

11. Diagnostic programs and automatic test equipment

Diagnostic programs apply known workloads or patterns while observing internal checks and outputs. They may target arithmetic, memory, channels, peripherals, buses, or complete configurations. Coverage comes from exercising sensitized paths and comparing against an oracle. A passing result means that the tested behaviors under those conditions matched expectations; it does not establish the absence of untested faults.

Automatic test equipment industrialized stimulus, measurement, limits, and reporting. In manufacturing it could contact a board or device, apply vectors, measure analog parameters, bin results, and preserve yield data. In service depots it reduced reliance on individual probing skill. Modular instruments and programmable control created an early measurement pipeline: configure, stimulate, acquire, compare, classify, and archive.

ATE forced several questions into engineering form. What fault model justifies the vector set? How are fixtures calibrated? Which failures are caused by contact resistance or the test socket? What traceability links a result to device, lot, program version, limits, and instrument state? False fails waste product; false passes export risk. Diagnostic quality therefore includes both coverage and measurement-system integrity.

This era joined manufacturing and field evidence. Yield excursions could identify process changes; returned-unit diagnosis could expose escapes; recurring no-fault-found outcomes could reveal intermittent conditions or inadequate tests. The long-term trend was already visible: hardware diagnosis improves when evidence from design, production, qualification, service, and operation shares identity and vocabulary.

PART III

Diagnosing digital hardware

12. Logic analyzers, in-circuit emulation, and pre-silicon debugging

As digital systems acquired wider buses and faster state transitions, a two-channel oscilloscope could show electrical shape but not enough simultaneous logical context. The logic analyzer sampled many channels, converted voltages into logic states, and displayed timing or decoded state. Hewlett-Packard's 1973 logic analyzer and later state analyzers exemplified this shift from viewing individual waveforms toward reconstructing transactions and instruction-related events.[32, 33]

Triggering became a diagnostic language. Investigators could capture around a rare address, data pattern, sequence, or control condition and preserve events before and after it. Timing mode asked when edges occurred relative to a clock; state mode sampled according to the target's own clock and asked which logical state appeared. Both depended on thresholds, pod loading, skew, sample depth, and correct connection. A perfectly captured trace from the wrong clock domain could still be a confident fiction.

In-circuit emulation replaced or intercepted the processor so that a development system could control execution and expose registers, memory, and buses while the rest of the target operated. Intel's ICE-80, introduced for the 8080, made this approach a recognizable microprocessor-development instrument.[34] It connected software-like debugging operations—break, step, inspect—to physical pins and timing.

Increasing integration eventually removed external access to many internal buses. Emulation migrated into on-chip debug logic, while trace compression and high-speed ports carried selected events outward. The transition illustrates a recurring observability tradeoff: integration improves performance and cost while reducing natural visibility, so designers must deliberately spend gates, pins, bandwidth, memory, power, and security budget to restore it.

As hardware design moved upward into electronic-system-level models, debugging moved earlier than fabrication. Rogin and Drechsler's 2010 synthesis treats SystemC-based ESL debugging as a hierarchy spanning early error detection, high-level exploration, design understanding, and automatic failure isolation.[111] This extends hardware diagnosis from observing a manufactured device to reasoning about executable design models, properties, program slices, and structural views before silicon exists.

13. Semiconductor test: fault models, ATPG, scan chains, and design for testability

Integrated circuits made physical defects harder to reach and logical state more numerous. Production test therefore relied on fault models: simplified representations such as a node stuck at zero or one, bridging between lines, delay faults, or memory-cell coupling. A model does not claim every physical defect literally behaves that way; it provides a tractable target for generating patterns and estimating coverage.

The D-algorithm, published by J. Paul Roth in 1966, was an influential systematic method for propagating the effect of a modeled fault to an observable output.[35] Automatic test-pattern generation expanded that idea. But sequential logic presented a controllability and observability problem: reaching an internal state through normal operation could require long, uncertain sequences, and the effect might never reach a pin.

Scan design addressed the problem by connecting storage elements into shift paths during test. Internal state could be loaded, functional clocks applied, and the resulting state shifted out. Design for testability made observation an architectural property rather than an afterthought. It improved manufacturing coverage and failure analysis but incurred area, timing, routing, power, test-time, and security costs.

Fault coverage is evidence relative to a model and implementation. High stuck-at coverage does not prove equal coverage of timing, analog, power-integrity, or systematic design faults. Pattern effectiveness also depends on tester timing, voltage conditions, compression, masking, and the quality of physical diagnosis that maps failing patterns back toward candidate structures. Semiconductor test made diagnostic claims quantitative, but also made their assumptions easier to overlook.

14. Built-in self-test and power-on self-test

Built-in self-test moves stimulus generation, response compaction, or both into the device. Logic BIST may use pseudorandom patterns and signature registers; memory BIST applies algorithms designed to expose cell, addressing, transition, and coupling faults; analog and mixed-signal BIST use loopback, references, or embedded measurement. A classic tutorial literature organized BIST as a design response to the rising cost and reduced external access of complex circuits.[36]

Power-on self-test brought a user-visible version of the idea to personal computers. The IBM PC Guide to Operations described the system's automatic power-on self-test, audible indications, and displayed error codes, while the technical reference exposed low-level behavior and diagnostics.[38, 39] POST answered

a practical question before loading an operating system: are enough processor, memory, video, and peripheral functions working to continue?

Self-test creates a bootstrap problem. The tester relies on some of the hardware it tests, and a failure in clock, reset, power, ROM, or output path can prevent a useful report. Beep codes, LEDs, seven-segment displays, redundant status paths, and staged tests are responses to that problem. Early stages should use minimal trusted state; later stages may become richer as memory and firmware services are validated.

A self-test result is strongest when it records version, configuration, environmental conditions, subtest, expected and actual signatures, and whether the test was destructive. A generic “failed” lamp is actionable only if the service path can localize further. Conversely, an exhaustive-looking code can be misleading when the failing observation path itself is untrusted.

15. Boundary scan and standardized access: JTAG and successors

Surface-mount packages and multilayer boards made many interconnects physically inaccessible to probes. IEEE 1149.1 boundary scan placed controllable and observable cells near digital pins and standardized test access so board interconnects and component behavior could be examined without direct contact at every node. The standard first appeared in 1990 and has continued to evolve.[37]

Boundary scan can shift instructions and data through a serial path, sample pins, drive test values, and bypass devices. At board level it can reveal opens, shorts, and assembly faults that are invisible after packaging. It also became a gateway to device programming, core debug, manufacturing configuration, and internal instruments. What began as test access grew into a general diagnostic infrastructure.

Successors and extensions

Later standards addressed structures that board-level boundary scan could not describe by itself. IEEE 1500 defines wrappers and test access for reusable embedded cores inside a system on chip. IEEE 1687—Internal JTAG, or IJTAG—defines access to networks of embedded instruments and description languages that let procedures be retargeted through a changing scan network. IEEE 1838 defines test access for three-dimensional stacked integrated circuits, including die wrappers and serial access across a stack.[103, 104, 105]

These are successors in scope rather than replacements for IEEE 1149.1. A board TAP may lead into an IEEE 1687 instrument network; an IEEE 1500-wrapped core may be reached through a chip-level architecture; IEEE 1838 carries test access across die boundaries. Together they show the historical movement of the probe point—from package pins, to embedded cores and sensors, to heterogeneous multi-die assemblies.

Standard access does not guarantee semantic uniformity. Device chains require accurate descriptions; reset behavior matters; mixed-voltage and powered-off domains complicate interpretation; test modes may alter normal pin behavior. A chain failure near the first device can hide everything downstream. Engineers therefore test the test path itself and distinguish physical connectivity from the correctness of the boundary-scan description.

JTAG also demonstrates the security duality of observability. An interface able to halt cores, read memory, alter registers, or reprogram nonvolatile storage is invaluable in development and repair and dangerous in an adversarial setting. Lifecycle states, authentication, irreversible locks, audit, and controlled service procedures became part of hardware diagnostic design.

16. Memory diagnostics: parity, ECC, scrubbing, soft errors, and RowHammer

Memory failures range from manufacturing defects and wear to timing margin, power noise, temperature, radiation-induced upset, retention loss, and interaction between cells. Walking patterns, address tests, March algorithms, and stress conditions target different mechanisms. Because a memory array is large and regular, a small number of recorded errors can support localization—if physical address mapping, interleaving, sparing, and controller transformations are known.

Parity detects selected errors; ECC can correct common patterns and report syndromes. Scrubbing periodically reads and corrects memory so that a second error does not accumulate in the same codeword. Correctable-error counts are therefore operational evidence, not harmless noise. They need component, channel, address, temperature, workload, and time context. The absence of correctable reports can mean good health, disabled reporting, aggressive sparing, or a fault that bypasses the code.

Radiation studies showed that terrestrial electronics can experience soft errors. Ziegler and Lanford linked cosmic rays to electronic error mechanisms, and later measurements refined the contribution at sea level.[46, 47] Large field studies then found substantial correctable and uncorrectable DRAM error behavior in production fleets, challenging simple independent-error assumptions.[49] JEDEC standards provided a vocabulary and procedures for reporting soft-error rates.[48]

RowHammer revealed a different class: repeated activation of DRAM rows can induce disturbance errors in nearby rows, crossing reliability and security boundaries. The 2014 study demonstrated the phenomenon in contemporary modules and showed why abstract fault models must be revised when device physics and access patterns interact.[50] Memory diagnosis today spans cell physics, controller policy, address translation, firmware records, operating-system handling, and fleet statistics.

17. Storage diagnostics: bad-block maps, SMART, RAID, and latent faults

Storage devices have long hidden physical irregularity behind logical interfaces. Bad-sector maps, remapping, retries, ECC, wear leveling, and firmware recovery can preserve apparently successful I/O while the medium or mechanism degrades. Diagnostics therefore needs both host-visible errors and device-internal health evidence. The boundary is difficult because internal policies differ by model and firmware.

SMART standardized a way for drives to expose health attributes and self-tests, and management profiles attempted to normalize diagnostic functions.[53] Yet attribute semantics and thresholds can be vendor-specific. Large field studies found that some SMART signals correlate with failure while many failed drives showed no strong pre-failure signal. Google's population study and related work at Carnegie Mellon emphasized both the value and the limits of drive telemetry.[51, 52]

RAID added redundancy and reconstruction but also new latent-fault interactions. A failed drive may be replaced successfully only if every required sector on surviving members can be read. Patrol reads, scrubs, checksums, and end-to-end validation seek to discover latent errors before degraded operation depends on them. A controller that silently remaps or retries can improve availability while concealing the rate at which margin is being consumed.

Solid-state storage changes mechanisms but not the diagnostic logic. Program/erase wear, retention, read disturb, controller firmware, power-loss behavior, and flash translation complicate the relation between logical address and physical cell. Useful evidence includes media-error counters, spare consumption, temperature, workload, unsafe shutdowns, firmware identity, and the history of corrected and retried operations—not a single synthetic "health percentage."

18. Processor and interconnect diagnostics: machine-check architecture, AER, and containment

Modern processors detect internal and external errors through parity, ECC, protocol checks, watchdogs, timeout logic, and machine-check architecture. Intel's architecture, for example, defines banks of status, address, and miscellaneous information with validity and severity semantics.[44, 45] The diagnostic record is produced close to the detection point but may be consumed by firmware, an operating system, a hypervisor, a BMC, or a management service.

Interconnects add layered detection. A link can retrain, replay a packet, correct an error, report a malformed transaction, or declare a component unreachable. Advanced Error Reporting in PCI Express and comparable mechanisms distinguish correctable, nonfatal, and fatal conditions and preserve source identifiers. The categories guide containment, but they are not root-cause labels: a receiver may report corruption created by a transmitter, channel, connector, clock, power event, or protocol violation.

Silent data corruption and mercurial cores

The sharpest limit of this detection architecture is a processor that computes the wrong result without raising a machine check, producing an ECC syndrome, or incrementing a useful counter. Google called rare cores with intermittent, pattern-sensitive erroneous computation mercurial cores. Meta reported silent data corruption across hundreds of thousands of machines and found CPUs whose defects were not captured by hardware error-reporting mechanisms.[101, 102] A successful instruction retirement is not proof that the instruction was computed correctly.

Because the fault creates no native alarm, evidence has to be manufactured above and around it: recurring per-core test corpora, end-to-end invariants and checksums, duplicate or diverse recomputation, application-level semantic checks, fleet cohort analysis, quarantine, and replacement. The Google and Meta reports also change the time model. A core that passed manufacturing test may fail only for a particular instruction sequence, voltage, temperature, aging state, or physical instance, so one clean qualification result cannot establish lifetime correctness.

Signal-integrity margining

A link can operate correctly while its timing or voltage eye is narrowing. Oscilloscope eye diagrams, jitter decomposition, bit-error-rate testing, stressed-receiver tests, voltage and timing sweeps, equalization experiments, and crosstalk or power-noise correlation measure how far operation is from the decision boundary. They answer a different question from a protocol counter: not only whether errors occurred, but how much tolerance remains before they become likely.

PCI Express 4.0 introduced standardized Lane Margining at the Receiver so software can step the receiver sampling point and measure available time and voltage margin under an operating link. PCI-SIG describes the feature as a way to determine how close each lane is to the edge under real conditions.[106] Margin results still require context—lane identity, speed, equalization state, channel topology, temperature, workload, firmware, and repeatability—and a healthy eye at one observation point does not rule out common-mode power or clock faults.

Containment determines whether diagnosis remains possible. Poisoned data, transaction aborts, link reset, page retirement, core offlining, and partition isolation can prevent propagation. But every recovery

action changes the system. If the original syndrome, topology, timing, and configuration are not captured first, successful recovery can erase causal evidence.

Processor diagnosis is therefore cross-layer. A user process sees a crash; the kernel records a machine check; firmware stores a persistent record; the BMC reports a sensor excursion; board telemetry shows voltage droop; fleet data identifies a revision-specific pattern. None of these alone is the hardware story. Their correlation depends on shared time, component identity, topology, and trustworthy decoding tables.

19. The physical failure-analysis laboratory: X-ray, emission microscopy, SEM, and FIB

Operational diagnostics usually stop at a suspect component, path, or condition. Physical failure analysis continues from that localization toward the material defect and the process that created it. The laboratory closes the return path promised by reliability engineering and fleet observability: symptom to failing site, site to mechanism, mechanism to manufacturing, design, screening, handling, or operating change. The ASM Handbook frames failure analysis as a systematic investigation that connects evidence, mechanism, root cause, and corrective action.[108]

Preserve first; remove material last

A good sequence begins with chain of custody, photographs, configuration and event history, electrical verification, and a written hypothesis matrix. Noninvasive inspection precedes decapsulation, polishing, delayering, or sectioning because every preparation step can destroy evidence or create artifacts. The investigator repeatedly correlates physical coordinates with schematics, layout, scan diagnosis, thermal or photon maps, and failing vectors.[108, 109]

Seeing through packages and assemblies

Radiography and X-ray computed tomography reveal buried structure without opening the part: solder-joint voids and bridges, cracked vias, wire-bond geometry, delamination, inclusions, and misalignment. Acoustic microscopy and infrared or optical methods add sensitivity to interfaces and materials. Multi-scale X-ray work on three-dimensional integrated circuits showed how nondestructive tomography can support process development, failure analysis, and production monitoring without sample-preparation artifacts.[110] Resolution, material density, orientation, reconstruction assumptions, and reference specimens bound what an image can prove.

Localizing electrical activity

Emission microscopy detects photons from electrically active defect sites such as leaky junctions, gate-oxide shorts, latch-up, or saturated devices. Lock-in thermography, optical-beam-induced current, light- or thermally-induced voltage alteration, and related stimulation methods map a measured response back to a position. The ASM microelectronics failure-analysis desk reference treats photon emission, thermal detection, and laser- and beam-assisted techniques as parts of a coordinated fault-isolation workflow.[109] A bright or hot spot is a localization result, not yet a mechanism.

Opening the site

Scanning electron microscopy supplies high-resolution surface and cross-section images; backscattered-electron contrast and energy-dispersive X-ray spectroscopy add compositional evidence. A focused ion beam can mill a site-specific cross-section, expose buried interconnect, prepare a transmission-electron-microscopy lamella, deposit material, or perform a circuit edit that tests a causal hypothesis. These capabilities can reveal voids, cracks, corrosion products, electromigration damage, dielectric breakdown, process residues, or malformed geometry, but beam damage, redeposition, charging, curtaining, and preparation bias must be documented.[109]

The strongest conclusion correlates independent evidence: the failing electrical condition, a localized site, a physical anomaly, a plausible stress and mechanism, and reproduction or corrective action. A striking micrograph without that chain may be incidental damage. Conversely, a fleet correlation without returned-part analysis may identify a vulnerable cohort while leaving the causal process unknown. The physical laboratory and the observability pipeline are therefore one diagnostic system operating at different scales.

PART IV

Embedded, vehicle, aerospace, and physical-asset health

20. Firmware as diagnostic mediator: BIOS, UEFI, ACPI, and persistent error records

Firmware occupies a privileged diagnostic position. It initializes hardware before an operating system exists, discovers topology, runs early tests, configures error reporting, and may continue to handle platform events after boot. This makes it a mediator between silicon-specific evidence and higher-level tools—and a possible source of translation loss.

UEFI Platform Initialization status codes formalize progress, error, and debug reporting during firmware execution.[40] The UEFI specification defines a common boot and runtime environment, while ACPI's Platform Error Interfaces describe structures such as the Hardware Error Source Table and mechanisms for error serialization.[41, 42] Microsoft's Windows Hardware Error Architecture consumes standardized error records so applications and management tools can reason about hardware conditions without knowing every chipset register.[43]

Persistent records matter when the machine cannot continue. Firmware or platform storage can preserve evidence across reset, allowing a later operating system or service tool to retrieve it. But persistence also creates lifecycle questions: when is a record cleared, how are repeated events counted, how is time represented before the real-time clock is valid, and how is a component identified after replacement?

Firmware expands observability while adding version dependence. A silicon event may be masked, remapped, summarized, or rate-limited before software sees it. Diagnostic reports should therefore preserve firmware, microcode, board, and schema versions. When possible, raw status and decoded interpretation should travel together so that improved decoders can revisit old evidence.

21. Embedded debug and trace: Nexus, CoreSight, and RISC-V

Embedded systems need visibility without the luxury of stopping time-critical control. On-chip debug modules provide halt, step, breakpoint, watchpoint, register, and memory access. Trace systems record instruction flow, branches, data accesses, timestamps, exceptions, or system events with less intrusion than print-based instrumentation. Compression is essential because internal event rates exceed practical output bandwidth.

The Nexus 5001 forum standard defined a scalable interface for embedded processor debug, while Arm CoreSight organized trace sources, links, funnels, buffers, and sinks into a system-wide infrastructure.[54, 56] Arm's program-flow trace specifications show how execution can be reconstructed from encoded control-flow information rather than every fetched instruction.[55] RISC-V's ratified Debug Specification 1.0 similarly standardizes a path for external debuggers across implementations.[57]

Trace is selective history. Filters, trigger windows, buffer depth, timestamp sources, synchronization packets, and dropped-data behavior determine which past can be reconstructed. A trace buffer that freezes after a trigger may preserve the lead-up to failure; a streaming port may capture longer history but fail when the transport is overloaded. Analysts need explicit loss markers and configuration alongside the payload.

These interfaces also cross trust boundaries. Production devices may contain keys, personal data, control authority, or safety-critical state. Secure debug requires lifecycle governance: who may enable it, under what physical and cryptographic conditions, what is logged, what can be read or changed, and how recovery works when the main processor is not trusted.

22. Automotive diagnostics: warning lamps, OBD-II, CAN, UDS, and freeze frames

Automotive diagnostics evolved from local mechanical observation and proprietary service connectors into regulated, networked evidence. Engine-control units could detect sensor, actuator, circuit, and emissions-control conditions, store diagnostic trouble codes, illuminate a warning lamp, and expose data to a scan tool. U.S. regulation and California requirements made OBD-II broadly mandatory for 1996 and later light-duty vehicles.[58, 59]

SAE J1979 defined standardized diagnostic test modes and parameter access for emissions-related systems, including trouble codes and freeze-frame information.[60] Freeze frames preserve selected operating conditions when a fault is recorded: speed, load, temperature, fuel-control state, and other parameters. The evidence is contextual but bounded. It may show what the controller believed, not what the physical sensor actually experienced, and later events may overwrite limited storage.

Vehicle networks expanded diagnosis beyond the engine. CAN carries messages among controllers; Unified Diagnostic Services support sessions, data identifiers, routines, security access, and reprogramming. The industry's transition toward OBD on UDS is reflected in SAE J1979-2.[61] A modern symptom can involve physical hardware, sensor calibration, wiring, bus behavior, control software, gateway policy, and cloud services.

Trouble codes are detectors, not verdicts. An oxygen-sensor-related code may arise from the sensor, wiring, exhaust leak, fuel delivery, or upstream combustion. Good diagnosis combines code definitions with topology, live data, bidirectional tests, electrical measurements, service history, and confirmed repair. Standardization made evidence portable; it did not make causal reasoning automatic.

23. Aerospace built-in test, FDIR, and integrated vehicle health management

Spacecraft and aircraft cannot assume immediate physical access. Built-in test, redundancy management, and fault detection, isolation, and recovery must operate under tight constraints on mass, power, computation, communication, and allowable intervention. NASA guidance treats fault management as a lifecycle activity connecting requirements, architecture, analysis, testing, operations, and anomaly response.[62, 63]

FDIR separates functions that are often conflated. Detection establishes that behavior is anomalous. Isolation narrows the responsible component, function, or region. Response reconfigures, safes, retries, or sheds load. Diagnosis may continue on the ground using richer telemetry and models. A recovery that preserves the mission but destroys evidence can make the next occurrence harder to understand, so snapshot buffers and event histories matter.

Integrated vehicle health management sought to combine sensors, models, maintenance information, and decision support across subsystems. NASA frameworks described IVHM as more than a collection of alarms: it integrates data and reasoning to support vehicle and fleet health.[64, 65] The integration challenge is semantic. Temperatures, residuals, mode states, maintenance actions, and structural measurements need common identity, time, units, and configuration.

Aerospace experience also disciplines autonomy. False positives can trigger hazardous reconfiguration; missed detections can lose the vehicle; ambiguous faults can propagate while the system debates. Thresholds, persistence logic, voting, inhibit conditions, confidence, and safe-state design are therefore part of the diagnostic argument. An autonomous response must be justified by the evidence it can obtain in the time available.

24. Industrial condition monitoring: vibration, acoustics, oil, current, and thermography

Rotating and process equipment made diagnosis a problem of degradation under load. Vibration transducers reveal imbalance, misalignment, looseness, bearing defects, resonance, and gear-mesh

behavior through amplitude, phase, waveform, and spectrum. Acoustic emission, motor-current signature, lubricant debris, temperature, pressure, and infrared imaging provide complementary views. Each modality is sensitive to different mechanisms and operating conditions.

A spectrum is not a fault label. Sensor mounting, orientation, bandwidth, windowing, sampling, rotational speed, load, and structural transmission shape the result. Vibration guidance emphasizes baseline and trend comparison because absolute thresholds can miss machine-specific behavior.[68] ISO 13374 organized condition-monitoring data processing into blocks from acquisition through state detection, health assessment, prognosis, and advisory generation.[66]

Condition-based maintenance changes the trigger for intervention from calendar time to evidence of health. That can reduce unnecessary work and catch degradation earlier, but it also shifts risk into sensors, thresholds, communications, and models. A failed sensor can imitate a healthy machine through a flat line or imitate a fault through drift. Sensor plausibility, calibration, redundancy, and missing-data detection are part of machinery health.

NIST reviews of industrial condition-monitoring technology emphasize evaluation across sensing, data, and decision layers.[67] The general lesson is multi-modal: a bearing diagnosis is stronger when vibration harmonics, temperature, lubricant evidence, speed, load, and inspection agree. Disagreement is not noise to discard; it may identify a sensor problem, a changing operating regime, or an incomplete mechanism model.

25. Nondestructive evaluation and structural health monitoring

Nondestructive evaluation asks what can be learned about material or structural condition without impairing the artifact. Visual, ultrasonic, radiographic, magnetic-particle, eddy-current, penetrant, thermographic, acoustic, and impact-based methods expose cracks, voids, corrosion, delamination, thickness loss, and other discontinuities. NIST defined NDE as obtaining information about an object’s properties without altering it and surveyed methods for in-situ construction materials.[69, 70]

An indication is not automatically a defect. Instrument response must be interpreted against geometry, material, surface condition, calibration standards, probability of detection, and acceptance criteria. False positives can cause unnecessary removal; false negatives can leave a critical flaw. The examiner’s procedure, qualification, equipment calibration, and record become part of the evidence chain.

Structural health monitoring extends occasional inspection with embedded or repeated sensing. Strain, acceleration, acoustic, corrosion, displacement, and environmental data can track change over time. Models connect local measurements to system condition, but sparse sensors and variable loading create

non-uniqueness: many damage states can produce similar responses. Baseline drift and sensor aging can be mistaken for structural change.

Additive manufacturing has brought process monitoring and in-process NDE closer to production. NIST distinguishes observation of process signatures from evaluation of flaws in the solidified part and warns that an in-process indication does not map automatically to a final defect.[71] The distinction generalizes: a diagnostic signal becomes a defect claim only through an explicit model and acceptance rule.

PART V

Remote and fleet-scale hardware observability

26. Network and appliance management: SNMP, RMON, counters, and streaming telemetry

Network equipment turned hardware monitoring into a remote protocol problem. Interfaces, fans, power supplies, temperatures, link state, packet errors, discards, and configuration needed names that a management station could retrieve. SNMP's 1990 architecture deliberately kept agents simple and modeled management largely as inspection or alteration of managed variables, with traps guiding attention.[72]

Counters gave durable operational memory but introduced interpretation traps. Wraparound, reset, discontinuity, sampling interval, aggregation, and changing interface identity can produce false trends. A rising error counter says that a detector fired; it does not by itself distinguish cable, optic, connector, peer, receiver, congestion, or configuration. Rates require two trustworthy samples, consistent time, and knowledge of counter width and reset behavior.

Remote Monitoring extended the idea toward traffic statistics, history, alarms, and captured events at network segments.[73] Later model-driven streaming telemetry and gNMI made subscription, typed paths, timestamps, and continuous updates central.[74] The architectural shift was from periodic polling of loosely typed objects toward a structured stream from the device, but the basic evidentiary obligations remained: identity, schema, units, loss behavior, and configuration.

Network observability also exposes observer cost. Polling consumes control-plane work; high-cardinality streams consume bandwidth and storage; detailed packet capture may contain sensitive content; an overloaded agent may report late or not at all. A robust system treats missing telemetry, stale sequence numbers, and collection backpressure as first-class hardware-health signals rather than silently converting them to zero.

27. Out-of-band server management: IPMI, BMCs, Redfish, and OpenBMC

The baseboard management controller created a computer beside the computer. Powered from standby rails, it can monitor sensors, read event logs, control fans, inspect power, provide a remote console, and

reset or power-cycle the host even when the host operating system is unavailable. IPMI standardized many of these functions and data records; DMTF now maintains the specification archive.[75]

Out-of-band access improves survivability of evidence. A host crash need not erase platform temperatures, voltages, boot progress, power events, or system-event-log entries. Yet independence is relative: the BMC can share board power, clock sources, buses, sensors, and firmware update infrastructure with the host. Its view may remain available during some host failures and disappear during others.

Redfish reframed management as a modern, resource-oriented interface with standard schemas for systems, chassis, managers, telemetry, events, and update services.[76] OpenBMC brought an open-source implementation ecosystem and documented telemetry architecture for exposing sensors, metrics, triggers, and reports.[77] These systems make machine-readable hardware state easier to aggregate across vendors while preserving OEM extensions where standards do not reach.

Management power brings security and epistemic risk. A compromised BMC can falsify observations, alter firmware, or control the host. Diagnostic trust therefore depends on secure boot, signed updates, authentication, authorization, audit, network isolation, and a clear statement of which controller produced each record. Out-of-band does not mean outside the system's threat model.

28. Data-center hardware telemetry: power, thermals, fans, links, and fleet health

At data-center scale, hardware condition becomes a population and topology problem. A single server exposes dozens or hundreds of sensors and counters; a fleet adds racks, power domains, cooling loops, networks, firmware cohorts, suppliers, and workload placement. Diagnosis must preserve the path from a fleet aggregate back to a physical unit and from that unit back to replaceable components.

Power and thermal evidence are coupled. A hot accelerator may result from workload, fan control, obstructed airflow, inlet temperature, thermal-interface degradation, voltage, firmware policy, or a neighboring device. Correlation across chassis sensors, facility telemetry, fan commands, power capping, workload, and ambient conditions helps separate these possibilities. Absolute thresholds catch extremes; spatial and peer comparison often reveal subtle anomalies.

Fleet systems turn corrected errors, link retries, fan excursions, throttling, and component replacement into rates and cohorts. Useful grouping includes model, lot, board revision, firmware, rack, environment, and workload. The danger is ecological inference: a cohort association can prioritize investigation but does not prove the mechanism in a specific machine.

Identity is the backbone. Serial number, physical slot, FRU identifier, topology address, firmware version, service event, and logical host name must remain joinable across replacement and redeployment. If a board moves but its historical counters appear to belong to the new chassis—or if a slot label is reused—the observability system manufactures a false history.

29. Performance counters and on-chip trace as silicon observability

Hardware performance counters count selected microarchitectural events: cycles, instructions, cache activity, branch outcomes, stalls, memory transactions, interconnect traffic, and power or thermal conditions. Intel's event guides document large, model-specific sets and the constraints on programming them.[79] These counters made silicon behavior available to operating systems and profilers without external probes.

A counter is a compressed observation. It loses event order and often attribution. Multiplexing estimates events that cannot be counted simultaneously; speculative execution may count work later discarded; skid moves an interrupt away from the precise instruction; privilege filtering and virtualization change scope. Event names that sound stable can differ between microarchitectures. Performance diagnosis requires model-specific definitions and validation with controlled workloads.

On-chip trace restores selected sequence. CoreSight program trace, timestamped system trace, fabric monitors, and vendor-specific flight recorders can connect instruction flow to interrupts, transactions, and external events.[54, 55] Buffer depth and bandwidth remain finite, so the decisive design questions are what to trigger on, what prehistory to retain, and how to align clocks across sources.

Silicon observability sits at the hardware-software boundary. Software configures counters, schedules work, and interprets symbols; hardware defines event semantics and capture precision. This cooperation is powerful but fragile. An analysis that omits microcode, firmware, kernel, virtualization, sampling, and processor model can turn an accurate count into an incorrect story.

30. Population studies: disks, DRAM, HBM, and the limits of health scores

Fleet-scale studies changed hardware reliability from a laboratory estimate into an empirical distribution under real workloads. Disk and DRAM studies used large populations to measure error incidence, recurrence, correlations, and predictive signals.[49, 51, 52] They revealed heavy-tailed behavior, concentration in subsets of devices, and limits to simple independence assumptions.

High-bandwidth memory and accelerators extend the challenge. Large 2024 field studies of HBM errors analyzed production behavior where memory stacks, package thermals, workloads, and error-management policy interact.[86] Such evidence can expose mechanisms invisible in qualification while also reflecting deployment-specific software, screening, and repair practices.

Health scores compress many signals into one value. Compression helps triage but hides assumptions about weighting, censoring, replacement policy, and cost. A score trained on removed drives learns the operator's removal decisions as well as device physics. If replaced units disappear from the risk set, apparent survivor behavior can be biased. Confidence, calibration, and feature-level explanation matter.

Population evidence is strongest when it supports a return path to mechanism. A rising cohort risk can trigger targeted bench analysis, supplier review, microscopy, firmware experiments, environmental checks, or test changes. Without that return path, fleet analytics can rank victims while leaving the causal process untouched.

PART VI

From evidence to prediction and automation

31. Model-based diagnosis, fault injection, and causal isolation

Model-based diagnosis represents expected component behavior and searches for sets of assumptions whose failure would explain the observations. Raymond Reiter's 1987 formulation gave a general logical account of diagnosis from a system description and inconsistent observations.[85] In engineering practice, models may be Boolean, qualitative, graph-based, probabilistic, physical, or hybrid.

Residuals compare measured behavior with model prediction. Isolation improves when candidate faults produce distinguishable residual patterns under available operating modes. This connects diagnosis directly to sensor placement and test design: if two mechanisms are observationally equivalent, no classifier can reliably separate them without new evidence or intervention.

Fault injection tests the diagnostic system by deliberately introducing controlled faults or errors in hardware, simulation, emulation, or software-mediated interfaces. NASA work has used injection and monitoring to characterize fault effects in embedded and aerospace systems.[83, 84] Injection can measure detection latency, coverage, containment, and recovery, but realism matters. A forced register bit may not reproduce the timing, analog behavior, spatial correlation, or progressive degradation of the physical mechanism it represents.

Causal isolation often requires active experiment: swap channels, vary load, change temperature, reroute traffic, disable a feature, or compare a known-good unit. The intervention is valuable because it breaks correlations, but it can also reset state or damage hardware. A disciplined diagnostic plan states the hypothesis, predicted observation, safety bound, rollback, and evidence to preserve before the change.

32. Prognostics, remaining useful life, digital twins, and AI-assisted maintenance

Prognostics asks a different question from fault detection: given current condition, operating plan, and uncertainty, how will degradation evolve and when will a functional limit be crossed? Remaining useful life is therefore not a property stored inside a component. It is a conditional estimate tied to a failure criterion, future load, environment, maintenance, and model.

Physics-based models use crack growth, wear, thermal cycling, electrochemistry, or other mechanisms. Data-driven models learn patterns from populations. Hybrid approaches constrain learning with physical

relationships. NASA's prognostics repository and C-MAPSS datasets helped make run-to-failure and simulated engine-degradation data widely available for method comparison.[81, 82] Their popularity also illustrates a warning: benchmark performance may not transfer to different fleets, sensors, maintenance policies, or failure modes.

Digital twins connect a physical asset to models and observations across its lifecycle. NIST describes digital twins for advanced manufacturing as representations that use data and models to support analysis and decisions.[80] A useful diagnostic twin is not merely a 3D model. It needs current configuration, parameter uncertainty, synchronization with measured state, provenance, validation, and a declared decision scope.

AI can summarize event streams, detect anomalies, rank hypotheses, recommend tests, and retrieve similar cases. Its strongest role is evidence orchestration: joining topology, manuals, sensor history, prior repairs, and mechanism models. It should expose supporting artifacts, uncertainty, and conflicts rather than emit an untraceable root-cause sentence. Maintenance action should remain bounded by authorization, safety, and verification.

33. Accelerators and AI infrastructure: GPU, HBM, and fabric telemetry

AI infrastructure concentrates power, heat, memory bandwidth, high-speed links, and tightly synchronized distributed work. A single degraded accelerator, HBM stack, optical path, PCIe link, or cooling zone can waste an entire training job without producing a simple hard failure. Diagnosis must connect device-level errors to topology, collective communication, workload phase, software stack, and facility conditions.

NVIDIA's Data Center GPU Manager exposes health, diagnostics, telemetry, profiling, and policy for GPU fleets, while Xid messages report classes of driver-detected GPU conditions.[87, 88] These interfaces are valuable starting points, not self-contained explanations. The same Xid can have multiple causes, and a job failure can be the first visible consequence of a link, memory, power, or software problem elsewhere.

Recent hyperscale reports describe automated testing and repair workflows for AI hardware. Meta has discussed fleet reliability practices across manufacturing, data centers, and repair. Alibaba Cloud's Aegis diagnoses failures in a production AI model-training service by combining job, GPU, host, and network evidence with targeted tests and operational workflows.[89, 90] The emerging pattern combines continuous telemetry, workload-aware health checks, cohort analysis, and controlled decommissioning or repair.

AI systems also diagnose the hardware that trains them, creating a governance loop. A learned model may recommend removing devices, rerouting work, or adjusting thresholds that alter its future data. Operators need counterfactual evaluation, audit, protected test populations, and human review for high-impact actions. Otherwise automation can optimize the visibility of its own policy rather than the reliability of the underlying fleet.

PART VII

Case studies, synthesis, and future directions

34. Diagnostic lessons from consequential hardware failures

Major investigations show that hardware evidence is rarely a single decisive sensor. The Apollo 13 review reconstructed how an oxygen-tank sequence involving prior damage, procedural change, thermostatic switches, wiring, and heating led to the in-flight explosion; telemetry and testing were interpreted alongside manufacturing and maintenance history.[93] The lesson is lifecycle continuity: a component's diagnostic identity begins before installation.

The Challenger investigation connected O-ring sealing behavior to cold conditions and examined how test and flight evidence had been interpreted before launch.[94] Aloha Airlines Flight 243 showed the interaction of fatigue, corrosion, inspection assumptions, and multi-site damage in an aging structure.[95] Columbia's investigation integrated imagery, telemetry, debris, testing, and organizational decisions to reconstruct foam impact and wing failure.[96] In each case, the investigation depended on evidence that crossed physical scale and institutional boundary.

Three Mile Island demonstrated how misleading or incomplete indications can shape operator understanding during a fast-moving event.[97] The Fukushima Daiichi report emphasized that loss of power, instrumentation, and control left operators with little reliable indication of plant condition.[98] Deepwater Horizon investigations joined pressure tests, alarms, control systems, physical components, maintenance, and operational decisions.[99] Observability is therefore a safety function: sensors must survive relevant hazards, express uncertainty, and reveal when their own path is unavailable.

The Pentium floating-point division defect showed a different dynamic. A rare hardware design error became publicly reproducible through a compact numerical example, turning a disputed symptom into a test any owner could run.[100] Reproduction did not explain the complete design mechanism, but it transformed accountability. Across all these cases, the durable lesson is not "add more sensors." It is to design evidence that remains interpretable under the failure, preserve anomalies before intervention, and allow independent tests to challenge the official model.

35. Recurring principles and future directions

Establish a baseline before interpreting deviation

Hardware measurements are comparative. Normal vibration, leakage, corrected-error rate, temperature, voltage margin, and link retry behavior depend on unit, configuration, environment, workload, and age. A universal threshold may catch catastrophe; a trustworthy baseline reveals change.

Preserve identity, topology, configuration, and time

A syndrome without physical address mapping, a temperature without sensor location, or a counter without reset history is weak evidence. Serial numbers, slots, lots, firmware, calibration, replacement events, and clock quality are diagnostic data, not administrative decoration.

Separate detection, isolation, diagnosis, recovery, and prognosis

A detector can be accurate while its label is causally wrong. Recovery can restore service without finding the mechanism. Prognosis adds assumptions about the future. Systems should record which claim each stage is making and the uncertainty it carries.

Treat corrected and retried events as evidence

ECC correction, link replay, remapping, sparing, and restart consume margin while hiding user-visible failure. Their rates and spatial patterns can reveal degradation, but only if they are retained before counters reset or components move.

Combine passive telemetry with controlled tests

Operational data shows behavior under real conditions; active tests create distinctions between hypotheses. Neither is sufficient alone. Tests should state their fault model, safety boundary, expected outcome, and observer effects.

Preserve first-failure data before changing state

Reset, retry, failover, reseat, and replacement may be necessary, but they alter the evidence. Flight recorders, freeze frames, persistent error records, trace buffers, and event logs should capture the earliest trustworthy state and the recovery sequence.

Make telemetry loss visible

A flat signal can mean stability, a stuck sensor, a dead collector, or exhausted bandwidth. Sequence numbers, freshness, confidence, calibration, and explicit missingness are part of the measurement.

Diagnose the observation path

Probes load circuits; sensors drift; firmware decodes registers; BMCs aggregate thresholds; schemas change. Independent channels, self-tests, calibration, raw evidence, and versioned decoders help separate device failure from observer failure.

Design for containment and degraded operation

Detection is most useful when the architecture can isolate a domain, retire a page, shed a function, reroute a link, or enter a safe state. Recovery policy should preserve enough evidence to learn from the event.

Close the loop from field evidence to design and manufacturing

The end of an investigation is not a replaced part. Confirmed mechanisms should change fault models, test patterns, screening, suppliers, firmware, thresholds, manuals, and training. No-fault-found returns are themselves evidence about the diagnostic system.

Govern powerful diagnostic interfaces

JTAG, debug modules, service processors, Redfish, and autonomous repair can expose secrets or control safety-critical hardware. Authentication, lifecycle state, least privilege, audit, and physical policy are part of observability design.

Instrument chiplets, packages, and materials

The next visibility boundary is inside heterogeneous packages. UCIe 2.0 and 3.0 add manageability, test, telemetry, and debug concepts across chiplets, acknowledging that system-in-package integration must be designed for diagnosis.[91] Embedded thermal, aging, link, and structural monitors will move evidence closer to failure mechanisms.

Use digital twins and AI as auditable hypotheses

Models and agents can correlate more evidence than a human can hold at once, but their outputs should link back to measurements, configuration, similar cases, and mechanism assumptions. NIST’s AI Risk

Management Framework supplies a broader governance vocabulary for validity, transparency, and accountable use.[92]

Keep the final explanation physical

The future may deliver autonomous fault isolation and repair, yet hardware still fails through energy, material, geometry, timing, environment, manufacture, and use. A useful explanation ultimately connects the alarm to a mechanism and to an intervention that changes the probability of recurrence.

Conclusion

The history of hardware diagnostics is not a smooth replacement of craft by automation. Sensory inspection survives beside high-rate telemetry; oscilloscopes remain essential beside on-chip trace; field substitution coexists with model-based inference; post-failure examination complements prognosis. Each new layer extends the evidence system while introducing its own assumptions and failure modes.

Across two centuries, progress came from making previously hidden distinctions observable: pressure over a cycle, resistance over distance, logic state across many channels, a syndrome within a codeword, a fault within a containment domain, a temperature within a topology, or a degradation trend across a fleet. The key innovation was often not the sensor alone but the surrounding structure of calibration, identity, stimulus, recording, interpretation, and corrective action.

Modern hardware observability should therefore be judged by more than the number of counters or endpoints. It should preserve first-failure evidence, express validity and missingness, survive relevant fault domains, connect physical and logical identity, support safe intervention, and allow raw artifacts to be reinterpreted as models improve. A dashboard can report that a detector fired; diagnosis must still explain what physical process made the detector right.

AI-assisted diagnosis will be most valuable when it strengthens that discipline. It can retrieve manuals, correlate fleet cohorts, propose distinguishing tests, and maintain competing hypotheses. It should not collapse uncertainty into a convenient cause label or act beyond verified authority. The enduring craft is the construction of an evidence-backed explanation that another investigator can challenge, reproduce, and use to prevent recurrence.

Appendix A. Selected chronology

Dates mark documented instruments, publications, standards, systems, and investigations. They indicate converging lineages rather than a single invention sequence. Rows are ordered by the beginning of the stated period; when an exact boundary year and a broader period share a start, the exact-year event appears first.

PERIOD	DEVELOPMENT	DIAGNOSTIC SIGNIFICANCE
Antiquity–1700s	Sensory inspection and maintenance craft	Sound, heat, motion, leakage, wear, and fracture are compared with learned normal condition.
1790s	Watt steam-engine indicator enters practical use.[1]	Hidden cylinder pressure becomes a recorded cycle related to piston travel.
1830s–1860s	Electrical telegraph maintenance develops continuity, insulation, bridge, and sectional tests.	Faults in inaccessible lines are classified and localized from terminal measurements.
1860s	Portable spring indicators such as the Richards design spread.[3]	Dynamic mechanical condition becomes easier to compare across engines and sites.
Late 1800s	Wheatstone, Murray, and Varley bridge practice matures for cable testing.[4]	A physical fault location is inferred from resistance ratios and cable models.
1897	Cathode-ray tube demonstrated as an instrument for electrical phenomena.	Fast electrical variation gains a direct visual display.
1920s–1930s	Electronic oscillographs and oscilloscopes improve time-domain measurement.	Transient behavior becomes visible rather than averaged by meters.
1940s	Wartime radar and electronics accelerate modular test and maintenance methods.	Calibrated instruments, test points, and replaceable assemblies become systematic.
1946	Tektronix begins work on its first triggered oscilloscope products.[6]	Stable display of repeated waveforms strengthens comparative diagnosis.
1947	Harvard Mark II relay moth is preserved in the operator log.[9]	A physical cause is documented with the symptom and corrective discovery.
Late 1940s	Whirlwind and other early computers use panels, scopes, test programs, and continuous maintenance.[10]	Logical and physical evidence are investigated together.
1950	Hamming publishes error-detecting and error-correcting codes.[12]	Syndromes encode location information, not only an alarm.
1950s	Marginal checking and diagnostic programs are used on operational computers.	Controlled stress exposes declining tolerance before nominal failure.
1950s–1960s	SAGE combines checking, redundancy, modular service, and continuous operation.[15]	Availability and maintainability become system-level properties.

PERIOD	DEVELOPMENT	DIAGNOSTIC SIGNIFICANCE
1956	von Neumann analyzes reliable systems built from unreliable components.[13]	Redundancy and fault models become formal design concerns.
1960s–1970s	FMEA, fault trees, qualification, and corrective-action systems mature.[22, 23]	Diagnostic coverage and sensor needs are analyzed before operation.
1962	Lyons and Vanderkulk describe triple modular redundancy for electronic computers.[14]	Voting can mask one discrepant module under explicit assumptions.
1964	IBM System/360 architecture documents machine-check interruption.[26]	Detected hardware malfunction becomes structured software-visible evidence.
1966	Roth publishes the D-algorithm.[35]	Automatic generation of tests for modeled logic faults becomes systematic.
1973	HP introduces a multi-channel logic analyzer.[32]	Digital timing and state across many signals become jointly observable.
1975	Intel introduces the ICE-80 in-circuit emulator.[34]	Processor execution can be controlled and inspected within the target system.
Late 1970s	BIST research formalizes on-chip stimulus and response compaction.[36]	Test resources migrate into increasingly inaccessible integrated circuits.
1979	Terrestrial cosmic-ray soft-error mechanisms are reported.[46]	Transient memory faults are connected to environment and device physics.
Late 20th century	Microelectronics laboratories combine emission localization, SEM, FIB, and nondestructive imaging.[108, 109]	Electrical symptoms gain a return path to a physical site and mechanism.
1980s	Mainframe and fault-tolerant systems deepen online error handling and serviceability.[27, 29]	Retry, logging, sparing, failover, and repair are designed as one workflow.
1981	IBM PC power-on self-test exposes coded boot diagnostics.[38]	Automatic startup checks become familiar to mass-market users.
1990	IEEE 1149.1 boundary scan is standardized.[37]	Board interconnects become testable without physical probe access to every node.
1990	SNMP is standardized in RFC 1157.[72]	Remote devices expose named management variables and traps.
Early 1990s	SMART emerges for drive self-monitoring and reporting.	Storage devices expose internal indicators beyond host-visible I/O failure.
1994	Pentium FDIV defect becomes publicly reproducible.[100]	A compact external test turns a rare design fault into accountable evidence.
1996	OBD-II becomes broadly required for U.S. light-duty vehicles.[59]	Standard codes, scan access, and freeze-frame evidence scale service diagnosis.
1998	IPMI begins standardizing platform-management functions.	A controller outside the host operating system observes and controls the server.

PERIOD	DEVELOPMENT	DIAGNOSTIC SIGNIFICANCE
1999–2003	Nexus and CoreSight-era embedded debug architectures emerge.[54, 56]	Trace and debug move on chip as external buses disappear.
2000s	ACPI error interfaces and OS hardware-error architectures mature.[42, 43]	Firmware, operating systems, and applications exchange structured error records.
2007	Large disk-fleet studies quantify failure trends and SMART limits.[51, 52]	Population evidence challenges simple vendor health scores.
2009	Large-scale production DRAM errors are reported.[49]	Correctable and uncorrectable errors show clustering and operational context.
2010	Rogin and Drechsler systematize electronic-system-level debugging around SystemC and hierarchical failure isolation.[111]	Hardware debugging moves into executable design models before fabrication.
2010s	Digital-twin and prognostics programs connect models with live asset data.[80, 81]	Diagnosis extends toward future condition and decision support.
2010s	Streaming telemetry and gNMI supplement polling.[74]	Typed, subscription-based device state becomes a continuous evidence stream.
2010s	OpenBMC develops an open platform-management stack.[77]	BMC sensors, events, and telemetry become inspectable and extensible.
2014	RowHammer disturbance errors are demonstrated in commodity DRAM.[50]	Access patterns interact with device physics across cell boundaries.
2015	DMTF publishes the initial Redfish standard generation.[76]	Hardware management moves toward schema-driven web APIs.
2020s	Accelerator telemetry and health tooling become fleet infrastructure.[87, 88]	GPU, HBM, fabric, workload, and thermal evidence must be correlated.
2021	Google and Meta report silent data corruption from rare, defective processor cores at fleet scale.[101, 102]	Wrong computation can occur without a machine check, syndrome, or useful hardware counter.
2024	Production HBM error behavior is studied at scale.[86]	Package memory reliability becomes an operational population problem.
2024	NIST distinguishes in-process monitoring from in-process NDE for additive manufacturing.[71]	Process anomalies and material defects are separated as diagnostic claims.
2024	UCIe 2.0 adds a manageability and DFx architecture for chiplets.[91]	Test, telemetry, and debug are designed across heterogeneous dies.
2025	RISC-V ratifies Debug Specification 1.0.[57]	A common external debug model spans an open instruction-set ecosystem.
2025	Alibaba Cloud reports Aegis fault diagnosis for a production AI model-training service.[90]	Job, GPU, host, and network evidence drive targeted diagnosis and operations.

PERIOD	DEVELOPMENT	DIAGNOSTIC SIGNIFICANCE
2025	UCIe 3.0 expands chiplet manageability and field capabilities.[91]	Package-level observability becomes a standardized lifecycle concern.
Emerging	Governed diagnostic agents join telemetry, topology, manuals, and fleet cases.	Automation shifts from alarm ranking toward evidence-backed hypothesis and action.

Table 3. Selected chronology of hardware diagnostics, observability, and debugging.

Appendix B. Glossary

Terms are defined for historical comparison. Vendor implementations and sector-specific standards may use narrower meanings.

Active diagnosis. Fault isolation by changing stimuli, configuration, environment, routing, or component state and observing the result.

Advanced Error Reporting (AER). A PCI Express capability for recording and reporting correctable, nonfatal, and fatal protocol or link errors.

Alarm. A notification that a monitored condition crossed a rule; it is not necessarily a diagnosis.

Anomaly. Observed behavior that differs from a baseline, model, or peer distribution.

ATE. Automatic test equipment that applies stimuli, measures responses, enforces limits, and records manufacturing or service results.

ATPG. Automatic test-pattern generation: computation of stimuli intended to expose modeled hardware faults at observable points.

Baseline. A reference description of normal behavior for a specified unit, configuration, workload, environment, and time.

BIST. Built-in self-test: stimulus generation and/or response evaluation implemented within the device or system.

BMC. Baseboard management controller: an out-of-band computer that monitors and controls a host platform.

Boundary scan. Scan cells at device boundaries used to control and observe pins and board interconnects, commonly through IEEE 1149.1.

Burn-in. Operation under selected stress intended to precipitate latent manufacturing defects before field deployment.

Calibration. The process and record relating an instrument’s response to known references, including uncertainty.

CAN. Controller Area Network, a message-oriented vehicle and industrial bus widely used among electronic control units.

Causal isolation. Distinguishing mechanisms that could produce the same symptom by evidence or controlled intervention.

Checker. Logic that verifies a code, invariant, comparison, protocol, or other expected relation.

Condition-based maintenance. Maintenance initiated from evidence of asset condition rather than a fixed interval alone.

Containment. Architectural prevention of fault or error propagation beyond a defined boundary.

Correctable error. A detected error for which the system reconstructs valid data or service under its fault model.

Counter. An accumulated count of selected events; interpretation requires scope, width, reset, and sampling semantics.

Coverage. The proportion of a stated fault model, behavior set, or requirement exercised or detected by a test or mechanism.

Debug interface. A path for controlling and inspecting hardware or processor state, often including halt, step, register, and memory access.

Degraded mode. A state that preserves selected essential service while reducing function, performance, or redundancy.

Detection. Establishing that an anomalous or specified fault-related condition is present.

DFT. Design for testability: structures added to make internal state controllable and observable for test.

Diagnostic trouble code (DTC). A standardized or vendor-defined code stored by a controller after a qualifying diagnostic condition.

Diagnostics. Inference from evidence to identify failure mechanisms, contributing conditions, and corrective action.

Digital twin. A model-based digital representation connected to a particular physical asset, process, or class through lifecycle data.

ECC. Error-correcting code that uses redundancy to detect and correct specified error patterns.

Emission microscopy. Localization of electrically active semiconductor defects by imaging photons emitted during biased operation.

Error. An incorrect internal state that may propagate to an externally visible failure.

Evidence provenance. Information establishing how, when, where, and by which calibrated or versioned path an artifact was produced.

Eye diagram. An overlay of many symbol intervals used to visualize timing and voltage opening, jitter, noise, and intersymbol interference in a link.

Failure mechanism. The physical, chemical, electrical, thermal, mechanical, or radiation process that produces damage or loss of required function.

Fault. A hypothesized or established cause of an error or failure, physical or design-related, transient or permanent.

Fault injection. Controlled introduction of a fault or error to evaluate detection, isolation, containment, or recovery.

Fault tree. A deductive model that relates an undesired top event to combinations of contributing events.

FDIR. Fault detection, isolation, and recovery: a staged fault-management function common in aerospace systems.

FIB. Focused ion beam: a site-specific material-removal and deposition tool used for cross-sectioning, sample preparation, and circuit edit.

First-failure data capture. Preservation of the earliest trustworthy state before retry, reset, failover, or repair changes the evidence.

Flight recorder. A bounded buffer or persistent store intended to retain pre-failure and failure-time events.

FMEA. Failure modes and effects analysis: systematic examination of possible failure modes and their local and system effects.

Freeze frame. Selected operating parameters preserved when a diagnostic condition is recorded, especially in vehicles.

FRU. Field-replaceable unit: a service boundary chosen for efficient isolation and replacement.

Graceful degradation. Continuation of prioritized service with reduced capability under fault.

Health score. A composite value that summarizes multiple indicators; its interpretation depends on training, weighting, and policy.

IJTAG. Internal JTAG, IEEE 1687: standardized access to and operation of embedded instruments within a semiconductor device.

Isolation. Narrowing a detected problem to a component, function, region, path, or candidate set.

JTAG. Common name for IEEE 1149.1 test access and boundary-scan infrastructure.

Lane margining. Controlled movement of a receiver's sampling point to estimate available timing and voltage margin on an operating serial link.

Logic analyzer. An instrument that samples many digital channels and displays timing, state, or decoded transactions.

Machine check. A processor or platform event indicating detected hardware malfunction, often accompanied by structured status.

March test. A family of ordered memory operations designed to detect specified cell, address, transition, and coupling faults.

Margin test. A controlled change of voltage, timing, temperature, or other condition to expose reduced tolerance.

MCTP. Management Component Transport Protocol, a DMTF transport used among platform-management components.[78]

Mercurial core. A processor core that intermittently produces incorrect computation without a conventional detected-hardware-error report.

Missingness. The explicit condition that expected telemetry is absent, stale, dropped, or invalid rather than equal to zero.

Model-based diagnosis. Inference of candidate faulty components or assumptions from inconsistency between a system model and observations.[85]

No fault found. A service outcome in which a reported symptom cannot be reproduced or localized under the available test conditions.

Nondestructive evaluation. Measurement or inspection that obtains information about an object without impairing its intended use.

Observability. The designed capability to infer internal physical or logical condition from outputs, telemetry, and controlled access.

Observer effect. A change in target behavior caused by the measurement or diagnostic method itself.

Out-of-band management. Observation and control through a path designed to remain available independently of the host operating system.

Parity. A redundant bit or relation used to detect selected odd-numbered bit errors.

Physical failure analysis. Laboratory investigation that correlates functional evidence with a localized physical defect, mechanism, and root cause.

Poison. A marker propagated with suspect data so downstream components can contain or report rather than silently consume it.

POST. Power-on self-test run during startup before normal software services are available.

Predictive maintenance. Maintenance planning informed by predicted risk or remaining useful life.

Prognostics. Estimation of how condition will evolve and when a defined functional limit may be reached.

RAS. Reliability, availability, and serviceability: system qualities covering sustained correct service, uptime, and efficient diagnosis and repair.

Redfish. A DMTF standard for secure, schema-driven management of servers and related infrastructure.[76]

Redundancy. Additional components, information, or paths used for detection, masking, recovery, or service continuity.

Remaining useful life (RUL). A conditional estimate of time or usage before a defined limit, given future loads and uncertainty.

Residual. Difference between measured behavior and a model's predicted behavior.

Retry. Repetition of an operation after detected difficulty; successful retry can restore service while consuming or revealing margin.

Scan chain. Storage elements connected for serial loading and unloading of internal test state.

Scrubbing. Periodic reading, checking, and correction or rewriting of stored data to prevent error accumulation.

Self-monitoring, analysis and reporting technology (SMART). A storage-device framework for health attributes, status, and self-tests.

SEM. Scanning electron microscopy: electron-beam imaging used for high-resolution surface and cross-section inspection, often with compositional analysis.

Sensor drift. Slow change in instrument response unrelated to the target quantity.

Service processor. A processor dedicated to monitoring, maintenance, diagnostics, and control outside normal workload execution.

Silent data corruption (SDC). Incorrect data or computation that propagates without being detected by the relevant hardware or system error-reporting path.

Soft error. A transient state corruption not caused by permanent damage, commonly associated with radiation or electrical disturbance.

Structural health monitoring. Repeated or continuous sensing and modeling of a structure's condition over time.

Syndrome. The pattern produced by parity checks or other relations that supports error detection and localization.

Test point. A deliberately accessible node or signal boundary with expected measurement behavior.

Thermography. Imaging of emitted infrared energy to infer surface temperature patterns.

Trace. An ordered record of selected execution, bus, control, or physical events.

Traceability. The linkage of a measurement or result to asset identity, configuration, procedure, reference, and calibration.

UDS. Unified Diagnostic Services, a protocol family used for diagnostic communication with vehicle controllers.

Uncorrectable error. A detected error outside the correction capability or confidence of the relevant code or mechanism.

Vibration spectrum. Distribution of measured vibration amplitude or power over frequency, conditioned by sampling and processing.

Watchdog. An independent or partially independent timer or monitor that initiates action when expected progress is absent.

X-ray computed tomography (CT). Nondestructive reconstruction of three-dimensional internal structure from X-ray projections.

Appendix C. Evidence taxonomy

Hardware investigations become stronger when each artifact is matched to the question it can answer and to the identity needed to trust it.

Evidence	Strongest questions	Characteristic blind spots	Identity / trust requirements
Direct inspection	Visible condition, geometry, damage, contamination, assembly	Hidden, internal, intermittent, and load-dependent mechanisms	Asset and location identity; lighting; scale; inspector; image provenance
Electrical measurement	Voltage, current, resistance, timing, continuity, integrity	Probe loading, grounding, bandwidth, aliasing, inaccessible nodes	Instrument and probe calibration; range; connection; circuit state
Waveform / trace	Sequence, phase, transient behavior, protocol and control flow	Finite trigger window; compression; missing events; observer effects	Clock source; trigger; sample rate; threshold; loss markers; decoder version
Self-test result	Conformance to a specific pattern set and test condition	Untested fault models; tester failure; destructive side effects	Test version; coverage claim; expected signature; environment; stage
Syndrome / machine check	Detected error class, candidate location, severity, validity	Shared-cause faults; mapping loss; secondary errors after propagation	Raw registers; decoding tables; topology; firmware and microcode
Counter / sensor trend	Rate, recurrence, degradation, comparison with peers or baseline	Reset, wrap, drift, thresholding, censoring, missingness	Units; sample interval; reset history; sensor location and calibration
Firmware / BMC event	Boot stage, platform sensor, power, thermal, FRU and recovery history	Observer compromise; shared rails and buses; aggregation; stale clocks	Controller identity; firmware; schema; signature; time quality; audit
Vehicle / control record	Controller-detected condition and selected operating context	Controller beliefs may differ from physical state; limited freeze frame	VIN/asset; ECU; DTC definition; mode; calibration; overwrite policy
NDE indication	Material discontinuity, thickness, crack, void, corrosion, delamination	Geometry artifacts; probability of detection; acceptance ambiguity	Procedure; examiner qualification; reference block; calibration; image chain
Physical failure-analysis artifact	Defect site, material anomaly, composition, damage morphology, plausible mechanism	Preparation artifacts; incidental anomalies; destructive loss of context; weak electrical correlation	Chain of custody; pre-analysis state; coordinates; instrument settings; sample preparation; analyst
End-to-end invariant / recomputation	Silent corruption that native hardware reporting did not detect	Common implementation or input faults; incomplete invariant; added cost and latency	Work identity; inputs; algorithm and version; comparison scope; core and host mapping
Maintenance and lifecycle record	Age, lot, duty, prior symptoms, interventions, replacement history	Incomplete free text; changed identity; unconfirmed part swaps	Serial/lot; service actor; timestamps; configuration; verification result
Fleet population evidence	Cohort risk, recurrence, environmental and revision correlations	Selection bias; policy effects; confounding; weak unit-level causality	Stable asset identity; exposure time; cohort definition; censoring and repair policy

Table 4. Evidence taxonomy. No evidence class is self-interpreting. A timestamp, counter, syndrome, or thermal image becomes diagnostic only when its provenance, configuration, units, sampling behavior, and relationship to the observed failure are known.

References

Primary standards, manuals, reports, archival material, and peer-reviewed studies are favored. URLs point to the cited publication, a verified publication copy, or an authoritative host. Citation metadata and linked-source identity were rechecked on 31 July 2026; some standards require registration, purchase, or subscription.

[1] Science Museum Group. “Watt’s steam engine indicator.” Collection record. https://collection.sciencemuseumgroup.org.uk/objects/co51439/watts-steam-engine-indicator

[2] Smithsonian National Museum of American History. “Watt Steam Engine Indicator Replica, 1927.” https://www.si.edu/object/watt-steam-engine-indicator-replica-1927%3Anmah_846148

[3] Smithsonian National Museum of American History. “Richards Steam Engine Indicator, ca. 1863.” https://www.si.edu/object/richards-steam-engine-indicator-ca-1863%3Anmah_846146

[4] R. W. Deltenre, J. J. Schwarz, and H. J. Wagnon. Underground Cable Fault Location: A Handbook to TD-153. EPRI EL-363, Research Project 481-1, Final Report. The BDM Corporation for the Electric Power Research Institute, January 1977. https://doi.org/10.2172/7233049

[5] Hawkins and Staff. Hawkins Electrical Guide, Number Three: A Practical Treatise. Theo. Audel & Co., 1914. https://www.gutenberg.org/files/49769/49769-h/49769-h.htm

[6] VintageTEK Museum. “Tektronix First Products – 101 & 511.” https://vintagetek.org/tektronix-first-products/

[7] Tektronix. “Oscilloscope Systems and Controls: Vertical, Horizontal & Triggering Explained.” https://www.tek.com/en/documents/primer/oscilloscope-systems-and-controls

[8] Keysight Technologies. “Our History.” https://www.keysight.com/us/en/about/keysight-technologies-history.html/1000

[9] Smithsonian National Museum of American History. “Log Book With Computer Bug.” https://americanhistory.si.edu/collections/object/nmah_334663

[10] Computer History Museum. “The Whirlwind Computer at CHM.” https://computerhistory.org/blog/the-whirlwind-computer-at-chm/

[11] Computer History Museum. “Jingle Bits: Auditory Maintenance, Whirlwind Holiday Songs, and the Dawn of Computer Music.” https://computerhistory.org/blog/jingle-bits-auditory-maintenance-whirlwind-holiday-songs-the-dawn-of-computer-music/

[12] R. W. Hamming. “Error Detecting and Error Correcting Codes.” Bell System Technical Journal 29, no. 2 (1950): 147–160. https://zoo.cs.yale.edu/classes/cs323/doc/Hamming.pdf

[13] John von Neumann. “Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components.” 1956. https://static.ias.edu/pitp/archive/2012files/Probabilistic_Logics.pdf

[14] R. E. Lyons and W. Vanderkulk. “The Use of Triple-Modular Redundancy to Improve Computer Reliability.” IBM Journal of Research and Development 6, no. 2 (1962): 200–209. https://doi.org/10.1147/rd.62.0200

[15] MIT Lincoln Laboratory. “SAGE: Semi-Automatic Ground Environment Air Defense System.” https://www.ll.mit.edu/about/history/sage-semi-automatic-ground-environment-air-defense-system

[16] MIT Lincoln Laboratory. MIT Lincoln Laboratory: Technology in Support of National Security. https://www.ll.mit.edu/sites/default/files/other/doc/2018-04/MIT_Lincoln_Laboratory_history_book.pdf

[17] Eldon C. Hall. MIT’s Role in Project Apollo: Final Report on Contracts NAS 9-153 and NAS 9-4065, Volume III: Computer Subsystem. R-700. Massachusetts Institute of Technology, Charles Stark Draper Laboratory, August 1972. https://www.ibiblio.org/apollo/Documents/R-700.pdf

[18] James E. Tomayko. Computers in Spaceflight: The NASA Experience. NASA, 1988. https://ntrs.nasa.gov/api/citations/19880069935/downloads/19880069935_Optimized.pdf

[19] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. “Basic Concepts and Taxonomy of Dependable and Secure Computing.” IEEE Transactions on Dependable and Secure Computing 1, no. 1 (2004): 11–33. https://www.landwehr.org/2004-aviz-laprie-randell.pdf

[20] NASA. NASA-STD-8729.1A, NASA Reliability and Maintainability Standard for Spaceflight and Support Systems. https://s3vi.ndc.nasa.gov/ssri-kb/static/resources/nasa-std-8729.1a.pdf

[21] NASA. NASA Systems Engineering Handbook, SP-2016-6105 Rev. 2, 2016. https://www.nasa.gov/wp-content/uploads/2018/09/nasa_systems_engineering_handbook_0.pdf

[22] U.S. Department of Defense. MIL-STD-1629A, Procedures for Performing a Failure Mode, Effects, and Criticality Analysis. 24 November 1980; cancelled 4 August 1998. https://everyspec.com/MIL-STD/MIL-STD-1600-1699/MIL_STD_1629A_1556/

[23] NASA. Fault Tree Handbook with Aerospace Applications. https://www.mwftr.com/CS2/Fault%20Tree%20Handbook_NASA.pdf

[24] NIST/SEMATECH. “Bathtub Curve” in e-Handbook of Statistical Methods. https://www.itl.nist.gov/div898/handbook/apr/section1/apr124.htm

[25] Daniel P. Siewiorek. “Architecture of Fault-Tolerant Computers: An Historical Perspective.” Proceedings of the IEEE 79, no. 12 (1991). https://bpb-us-w2.wpmucdn.com/sites.umassd.edu/dist/f/1260/files/2022/09/siew91.pdf

[26] IBM. System/360 Principles of Operation, GA22-6821-8, 1970. https://vtda.org/docs/computing/IBM/Mainframe/Hardware/System/GA22-6821-8_System360PrinciplesOperation_Nov70.pdf

[27] M. Y. Hsiao, W. C. Carter, J. W. Thomas, and W. R. Stringfellow. “Reliability, Availability, and Serviceability of IBM Computer Systems: A Quarter Century of Progress.” IBM Journal of Research and Development 25, no. 5 (1981): 453–468. https://doi.org/10.1147/rd.255.0453

[28] IBM. “Mainframe Strengths: Reliability, Availability, and Serviceability.” https://www.ibm.com/docs/en/zos-basic-skills?topic=it-mainframe-strengths-reliability-availability-serviceability

[29] Jim Gray. “Why Do Computers Stop and What Can Be Done About It?” Tandem Technical Report 85.7, 1985. https://pages.cs.wisc.edu/~remzi/Classes/739/Fall2018/Papers/gray85-easy.pdf

[30] Joel Bartlett, Jim Gray, and Bob Horst. Fault Tolerance in Tandem Computer Systems, Tandem TR 86.2, 1986. https://jimgray.azurewebsites.net/papers/TandemTR86.2_FaultToleranceInTandemComputerSystems.pdf

[31] E. S. Harrison and E. J. Schmitt. “The Structure of System/88, a Fault-Tolerant Computer.” IBM Systems Journal 26, no. 3 (1987): 293–318. https://pages.cs.wisc.edu/~david/papers/ibmsj1987_stratus.pdf

[32] Robin Adler, Mark Baker, and Howard D. Marshall. “The Logic Analyzer: A New Instrument for Observing Logic Signals.” Hewlett-Packard Journal 25, no. 2 (October 1973): 2–16. https://hparchive.com/Journals/HPJ-1973-10.pdf

[33] Hewlett-Packard. Hewlett-Packard Journal, February 1978: State Analysis. https://hparchive.com/Journals/HPJ-1978-02.pdf

[34] Intel. “The ICE-80.” Intel Technology Timeline, 1975. https://timeline.intel.com/1975/the-ice-80

[35] J. Paul Roth. “Diagnosis of Automata Failures: A Calculus and a Method.” IBM Journal of Research and Development 10, no. 4 (1966): 278–291. https://doi.org/10.1147/rd.104.0278

[36] Vishwani D. Agrawal, Charles R. Kime, and Kewal K. Saluja. “A Tutorial on Built-In Self-Test, Part 1: Principles.” IEEE Design & Test of Computers, 1993. https://www.ardent-tool.com/CPU/docs/AMD/anatomy/misc/articles/agrawal.pdf

[37] IEEE Standards Association. IEEE 1149.1: Test Access Port and Boundary-Scan Architecture. https://standards.ieee.org/ieee/1149.1/4484/

[38] IBM. Personal Computer Guide to Operations, 6025000, August 1981. https://ps2.infania.net/IBM_5150_Guide_to_Operations_6025000_AUG81.pdf

[39] IBM. Personal Computer Technical Reference, 6025005, August 1981. https://www.minuszerodegrees.net/manuals/IBM_5150_Technical_Reference_6025005_AUG81.pdf

[40] UEFI Forum. Platform Initialization Specification 1.9, Volume 3: Status Codes. https://uefi.org/specs/PI/1.9/V3_Status_Codes.html

[41] UEFI Forum. Unified Extensible Firmware Interface Specification 2.11. https://uefi.org/sites/default/files/resources/UEFI_Spec_Final_2.11.pdf

[42] UEFI Forum. ACPI Specification 6.5A, Platform Error Interfaces. https://uefi.org/specs/ACPI/6.5_A/18_Platform_Error_Interfaces.html

[43] Microsoft. “Introduction to the Windows Hardware Error Architecture.” https://learn.microsoft.com/en-us/windows-hardware/drivers/whea/introduction-to-the-windows-hardware-error-architecture

[44] Intel. Intel 64 and IA-32 Architectures Software Developer’s Manual, Volume 3B. https://cdrdv2-public.intel.com/671427/253669-sdm-vol-3b.pdf

[45] Intel. MCA Enhancements in Intel Xeon Processors. Reference Number 329176-002, Revision 1.1, January 2018. https://www.intel.com/content/www/us/en/content-details/671064/mca-enhancements-in-intel-xeon-processors.html

[46] J. F. Ziegler and W. A. Lanford. “Effect of Cosmic Rays on Computer Memories.” Science 206, no. 4420 (1979): 776–788. https://www.science.org/doi/10.1126/science.206.4420.776

[47] J. F. Ziegler and W. A. Lanford. “The Effect of Sea Level Cosmic Rays on Electronic Devices.” Journal of Applied Physics 52, no. 6 (1981): 4305–4312. https://doi.org/10.1063/1.329243

[48] JEDEC. JESD89B: Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices, 2021. JEDEC registration is required to access the hosted standard. https://www.jedec.org/system/files/docs/JESD89B.pdf

[49] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. “DRAM Errors in the Wild: A Large-Scale Field Study.” SIGMETRICS, 2009. https://research.google.com/pubs/archive/35162.pdf

[50] Yoongu Kim et al. “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors.” ISCA, 2014. https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf

[51] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. “Failure Trends in a Large Disk Drive Population.” FAST, 2007. https://research.google.com/archive/disk_failures.pdf

[52] Bianca Schroeder and Garth A. Gibson. “Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?” FAST, 2007. https://www.cs.toronto.edu/~bianca/papers/fast07.pdf

[53] DMTF. DSP1113, Disk Drive Diagnostics Profile 1.1.0a. https://www.dmtf.org/sites/default/files/standards/documents/DSP1113_1.1.0a.pdf

[54] Arm. CoreSight Architecture Specification. https://developer.arm.com/documentation/ihi0029/g/

[55] Arm. Embedded Trace Macrocell Architecture Specification. https://developer.arm.com/documentation/ihi0035/latest/

[56] IEEE-ISTO. The Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface. https://www.arbitrarytechnology.com/files/Download/ieee-5001%20nexus%20protocol.pdf

[57] RISC-V International. RISC-V Debug Specification, Version 1.0, ratified 21 February 2025. https://docs.riscv.org/reference/debug/v1.0/index.html

[58] U.S. Environmental Protection Agency. On-Board Diagnostic (OBD) Regulations and Requirements: Questions and Answers. EPA420-F-03-042, December 2003. https://nepis.epa.gov/Exe/ZyPURL.cgi?Dockey=P100LW9G.TXT

[59] California Air Resources Board. “On-Board Diagnostic II (OBD II) Systems Fact Sheet.” https://ww2.arb.ca.gov/resources/fact-sheets/board-diagnostic-ii-obd-ii-systems-fact-sheet

[60] SAE International. J1979: E/E Diagnostic Test Modes. https://www.sae.org/standards/j1979_201202-e-e-diagnostic-test-modes

[61] SAE International. J1979-2: E/E Diagnostic Test Modes—OBDonUDS, 2026. https://www.sae.org/standards/j1979-2_202604-e-e-diagnostic-test-modes-obdonuds

[62] NASA. “Fault-Detection, Fault-Isolation and Recovery (FDIR) Techniques.” Lessons Learned Information System, Lesson 839; based on Maintainability Technique DFE-7 in NASA TM-4628. https://llis.nasa.gov/lesson/839 Original DFE-7 preferred-practice PDF: https://extapps.ksc.nasa.gov/Reliability/Documents/Preferred_Practices/dfe7.pdf

[63] NASA. Fault Management Handbook, NASA-HDBK-1002 draft, 2012. https://s3vi.ndc.nasa.gov/ssri-kb/static/resources/636372main_NASA-HDBK-1002_Draft.pdf

[64] Deidre E. Paris, Luis C. Trevino, and Michael D. Watson. “IVHM Framework for Intelligent Integration for Vehicle Health Management.” NASA, 2005. https://ntrs.nasa.gov/citations/20050092389

[65] Mary S. Reveley, Jeffrey L. Briggs, Joni K. Evans, Sharon Monica Jones, Tolga Kurtoglu, Karen M. Leone, Carl E. Sandifer, and Megan A. Thomas. Commercial Aircraft Integrated Vehicle Health Management Study. NASA/TM-2010-215808, February 2010. https://ntrs.nasa.gov/citations/20100011007

[66] ISO. ISO 13374-1:2003, Condition Monitoring and Diagnostics of Machines—Data Processing, Communication and Presentation—Part 1: General Guidelines. https://www.iso.org/standard/21832.html

[67] Mehdi Dadfarnia, Michael Sharp, and Jeffrey Herrmann. “Comprehensive Evaluations of Condition Monitoring-Based Technologies in Industrial Maintenance: A Systematic Review.” Journal of Manufacturing Systems 82 (2025). https://doi.org/10.1016/j.jmsy.2025.06.015

[68] Brüel & Kjær. “Machine-Condition Monitoring Using Vibration Analysis: Permanent Monitoring of an Austrian Paper Mill.” Application Note BO 0247. https://www.bksv.com/doc/BO0247.pdf

[69] James R. Clifton and Nicholas J. Carino. “Nondestructive Evaluation Methods for Quality Acceptance of Installed Building Materials.” Journal of Research of the National Bureau of Standards 87, no. 5 (1982): 407–415. https://doi.org/10.6028/jres.087.024

[70] NIST. Nondestructive Testing (NDT) and Sensor Technology for Service Life Modeling of New and Existing Concrete Structures, NIST IR 7974, 2013. https://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7974.pdf

[71] Alkan Donmez, Jason Fox, Felix Kim, Brandon Lane, Maxwell Praniewicz, Vipin Tondare, Jordan Weaver, and Paul Witherell. In-Process Monitoring and Nondestructive Evaluation for Metal Additive Manufacturing Processes. NIST IR 8538, September 2024. https://doi.org/10.6028/NIST.IR.8538

[72] J. Case, M. Fedor, M. Schoffstall, and J. Davin. RFC 1157: A Simple Network Management Protocol, 1990. https://www.rfc-editor.org/rfc/rfc1157.html

[73] S. Waldbusser. RFC 2819: Remote Network Monitoring Management Information Base, 2000. https://www.rfc-editor.org/rfc/rfc2819.html

[74] OpenConfig. gRPC Network Management Interface (gNMI) Specification. https://openconfig.net/docs/gnmi/gnmi-specification/

[75] DMTF. Intelligent Platform Management Interface Specification Archive. https://www.dmtf.org/standards/ipmi

[76] DMTF. Redfish Standard. https://www.dmtf.org/standards/redfish

[77] OpenBMC Project. Telemetry Design. https://github.com/openbmc/docs/blob/master/designs/telemetry.md

[78] DMTF. Platform Management Communications Infrastructure standards. https://www.dmtf.org/standards/pmci

[79] Intel. Intel 64 and IA-32 Architectures Performance Monitoring Events. Document 335279-001, Revision 1.0, December 2017. https://cdrdv2-public.intel.com/671378/335279-performance-monitoring-events-guide.pdf

[80] NIST. “Digital Twins for Advanced Manufacturing.” https://www.nist.gov/programs-projects/digital-twins-advanced-manufacturing

[81] NASA Ames. Prognostics Center of Excellence Data Set Repository. https://www.nasa.gov/intelligent-systems-division/discovery-and-systems-health/pcoe/pcoe-data-set-repository/

[82] Abhinav Saxena and Kai Goebel. Turbofan Engine Degradation Simulation Data Set. NASA Ames Prognostics Data Repository, 2008. https://c3.ndc.nasa.gov/dashlink/resources/139/

[83] Allan L. White. “Designing Fault-Injection Experiments for the Reliability of Embedded Systems.” NASA, 2012. https://ntrs.nasa.gov/api/citations/20120016069/downloads/20120016069.pdf

[84] Wilfredo Torres-Pomales, Amy M. Yates, and Mahyar R. Malekpour. Fault Injection and Monitoring Capability for a Fault-Tolerant Distributed Computation System. NASA/TM-2010-216834, August 2010. https://ntrs.nasa.gov/citations/20100028297

[85] Raymond Reiter. “A Theory of Diagnosis from First Principles.” Artificial Intelligence 32, no. 1 (1987): 57–95. https://doi.org/10.1016/0004-3702(87)90062-2

[86] Ronglong Wu, Shuyue Zhou, Jiahao Lu, Zhirong Shen, Zikang Xu, Jiwu Shu, Kunlin Yang, Feilong Lin, and Yiming Zhang. “Removing Obstacles before Breaking Through the Memory Wall: A Close Look at HBM Errors in the Field.” 2024 USENIX Annual Technical Conference (USENIX ATC 24), 2024, 851–867. https://www.usenix.org/conference/atc24/presentation/wu-ronglong

[87] NVIDIA. Data Center GPU Manager User Guide. https://docs.nvidia.com/datacenter/dcgm/latest/user-guide/index.html

[88] NVIDIA. Xid Errors. https://docs.nvidia.com/deploy/xid-errors/index.html

[89] Meta Engineering. “How Meta Keeps Its AI Hardware Reliable.” 2025. https://engineering.fb.com/2025/07/22/data-infrastructure/how-meta-keeps-its-ai-hardware-reliable/

[90] Jianbo Dong et al. “Evolution of Aegis: Fault Diagnosis for AI Model Training Service in Production.” 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2025: 865–881. https://www.usenix.org/conference/nsdi25/presentation/dong

[91] UCIe Consortium. UCIe Specifications and Manageability / DFx Architecture. https://www.uciexpress.org/specifications

[92] NIST. Artificial Intelligence Risk Management Framework (AI RMF 1.0), 2023. https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf

[93] NASA. Report of Apollo 13 Review Board, 1970. https://ntrs.nasa.gov/api/citations/20170000913/downloads/20170000913.pdf

[94] Presidential Commission on the Space Shuttle Challenger Accident. Report, Volume I, 1986. https://www.nasa.gov/history/rogersrep/v1ch5.htm

[95] National Transportation Safety Board. Aloha Airlines Flight 243, Aircraft Accident Report AAR-89/03. https://www.ntsb.gov/investigations/AccidentReports/Reports/AAR8903.pdf

[96] Columbia Accident Investigation Board. Report, Volume I, 2003. https://ehss.energy.gov/deprep/archive/documents/0308_caib_report_volume1.pdf

[97] U.S. General Accounting Office. Three Mile Island: The Most Studied Nuclear Accident in History, EMD-80-109, 1980. https://www.gao.gov/assets/emd-80-109.pdf

[98] International Atomic Energy Agency. The Fukushima Daiichi Accident: Report by the Director General, 2015. https://www-pub.iaea.org/mtcd/publications/pdf/pub1710-reportbythedg-web.pdf

[99] U.S. Coast Guard and Bureau of Ocean Energy Management, Regulation and Enforcement. Deepwater Horizon Joint Investigation Team Final Report, 2011. https://www.bsee.gov/site-page/deepwater-horizon-joint-investigation-team-releases-final-report

[100] Tim Coe, Terje Mathisen, Cleve Moler, and Vaughan Pratt. “Computational Aspects of the Pentium Affair.” IEEE Computational Science & Engineering 2, no. 1 (1995): 18–30. https://doi.org/10.1109/99.372929

[101] Peter H. Hochschild, Paul Jack Turner, Jeffrey C. Mogul, Rama Krishna Govindaraju, Parthasarathy Ranganathan, David E. Culler, and Amin Vahdat. “Cores That Don’t Count.” 18th Workshop on Hot Topics in Operating Systems (HotOS), 2021. https://research.google/pubs/cores-that-dont-count/

[102] Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, and Sriram Sankar. “Silent Data Corruptions at Scale.” 2021. https://arxiv.org/abs/2102.11245

[103] IEEE Standards Association. IEEE 1500-2022, IEEE Standard Testability Method for Embedded Core-based Integrated Circuits. https://standards.ieee.org/ieee/1500/7704/

[104] IEEE Standards Association. IEEE 1687-2014, IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device. https://standards.ieee.org/ieee/1687/3931/

[105] IEEE Standards Association. IEEE 1838-2019, IEEE Standard for Test Access Architecture for Three-Dimensional Stacked Integrated Circuits. https://standards.ieee.org/ieee/1838/5073/

[106] Debendra Das Sharma. “Pushing to the Limits: Understanding Lane Margining for PCIe.” PCI-SIG, 2019. https://pcisig.com/blog/pushing-limits-understanding-lane-margining-pcie%C2%AE

[107] NASA Safety and Mission Assurance. “Reliability Assessment Using Physics-of-Failure Principles, Modeling and Simulation.” 2017. https://sma.nasa.gov/news/articles/newsitem/2017/05/22/reliability-assessment-using-physics-of-failure-principles-modeling-and-simulation

[108] Daniel P. Dennies. “How to Organize and Run a Failure Investigation.” In Failure Analysis and Prevention, ASM Handbook, vol. 11, 2021 ed., edited by Brett A. Miller, Roch J. Shipley, Ronald J. Parrington, and Daniel P. Dennies, 36–51. ASM International, 2021. https://doi.org/10.31399/asm.hb.v11.a0006755

[109] Tejinder Gandhi, ed. Microelectronics Failure Analysis: Desk Reference. 7th ed. ASM International, 2019. https://doi.org/10.31399/asm.mfadr7.9781627082471

[110] Wenbing Yun, Michael Feser, Jeff Gelb, and Luke Hunter. “Multi-scale Resolution 3D X-ray Imaging for 3D IC Process Development and Failure Analysis.” Abstract TH-19, 2011 International Conference on Frontiers of Characterization and Metrology for Nanoelectronics (IC-FCMN), MINATEC Campus, Grenoble, France, May 23–26, 2011. https://www.nist.gov/system/files/documents/pml/div683/conference/BookFCMN2011.pdf Note: A later FCMN 2013 presentation of nearly the same title is a distinct later record: NIST lists Michael Fesser of Xradia as presenter; the hosted deck bears Xradia, Inc. © 2012 and cites an October 2011 IMAPS paper. NIST 2013 presentations: https://www.nist.gov/pml/nanoscale-device-characterization-division/fcmn-publications-and-talks/2013-fcmn-presentations

[111] Frank Rogin and Rolf Drechsler. Debugging at the Electronic System Level. Springer Dordrecht, 2010. https://doi.org/10.1007/978-90-481-9255-7