A TECHNOLOGICAL AND INTELLECTUAL HISTORY

History of Hardware Diagnostics, Observability, and Debugging

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

Diagnostic artifacts and the mechanisms behind them

Concept and direction by Dmitry Vostokov, DumpAnalysis.org

Developed with AI assistance (GPT-6 Astra Max and Fable 5.1 Extra).

September 2026 · Version 14

Historical synthesis with 144 references, chronology, glossary, and evidence taxonomy

Abstract

Hardware diagnostics begins with observable signs of an internal problem. For example, a stopped shaft, an open cable, a drifting vacuum tube, a marginal memory cell, and a cracked solder joint may require different instruments and analysis techniques. However, in every case, we need to connect the observed symptom to the condition that produced it. This history follows the development of such connections through instruments, checking circuits, diagnostic programs, service records, and models.

We consider hardware diagnostics, observability, and debugging from early mechanical and electrical measurements to on-chip instrumentation and fleet analysis. The discussion includes reliability engineering, semiconductor test, physical failure analysis, firmware records, vehicle fault management, condition monitoring, and remote management. We also consider formal verification, post-silicon validation, and the use of digital twins and AI. These activities have different objects of analysis, although they frequently use the same diagnostic artifacts.

The main argument is that new diagnostic capabilities extend the available evidence and the ways we can analyze it. An embedded sensor does not remove the need for calibration, and an error code does not identify its own cause. A fleet correlation may suggest a component or operating condition for further investigation. To arrive at a supported diagnosis, we still need to distinguish the original fault, its propagation, the resulting symptoms, and the behavior of the instruments that recorded them.

How to read this history

This history is organized around diagnostic concepts and practices. Dates refer to documented instruments, publications, standards, systems, and investigations. Similar practices often developed in different laboratories, manufacturing organizations, and service communities. Therefore, a publication date should not automatically be interpreted as the date when a technique was first used. Our main interest is how the available evidence and the corresponding analysis changed.

We use the term diagnostic artifact for a recorded observation that can be examined during an investigation. Examples include an oscilloscope trace, an ECC syndrome, a maintenance log, and a microscopy image. Each artifact depends on how it was obtained. An oscilloscope trace depends on the probe and trigger settings; a syndrome depends on the code and address mapping; a management alert depends on sensors, firmware, thresholds, and time. We need this context to interpret the artifact correctly.

References in square brackets identify the sources. Standards and manuals describe particular interfaces and procedures; research papers and investigation reports provide findings within their stated scope. The case studies illustrate how physical condition, configuration, maintenance history, and interpretation contribute to a diagnosis. They should not be reduced to a single cause when the investigation identifies several contributing conditions.

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.

Recurring changes in diagnostic practice

- 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

We first distinguish several related activities. Testing compares a device with expected behavior under specified stimuli. Debugging includes interaction with the device or its model, for example, changing a signal, stopping execution, reading a register, or replacing a module. Monitoring records selected variables during operation. Diagnostics uses the resulting evidence to identify a problem and its possible mechanisms. Prognostics adds a prediction about future condition. These activities may be parts of the same investigation, but they answer different questions.

We can group the historical developments into three overlapping areas. Measurement converts physical quantities into signals and records, for example, pressure into an indicator diagram or vibration into a spectrum. Checking introduces an expected relation that can be tested, such as parity, a redundant result, or a self-test signature. Service practice connects symptoms and diagnostic actions to configuration, repair, and previous cases. Modern hardware observability incorporates all three areas.

The distinction between fault, error, and failure is useful here. A fault is a hypothesized or established cause of an error. Examples include a crack, a radiation event, a missing termination, and a design defect. An error is an incorrect internal state. A failure occurs when the delivered service deviates from the required service. Avizienis and colleagues organize these concepts within a dependability taxonomy that also includes design, physical, and malicious faults.[19] The same investigation may therefore involve a material defect, an incorrect bit pattern, and an externally visible failure.

Observability and distinguishable states

Observability also has a formal meaning in control theory. Kalman's 1960 work considers whether internal state can be determined from input and output behavior under a specified model.[112] This differs from counting sensors or collecting many logs. Here we also use observability in the broader engineering sense of access to interpretable internal evidence. Neither meaning implies that every physical cause is uniquely identifiable.

For example, two faults may produce the same output under the current test. An additional stimulus or measurement may separate them. This connects observability with diagnosability, the ability to distinguish the fault alternatives of interest. A large volume of repeated measurements can leave the same ambiguity unresolved. The diagnostic question determines which distinctions the observation system must support.

Historical sources also need context. An instrument preserved in a museum may be well documented, whereas the service records describing its everyday use may have been lost. A standard specifies an interface but does not establish how widely it was deployed. An accident report reconstructs events from the evidence available to its investigators. We distinguish these records from our interpretation of the longer development of diagnostics.

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

Before electronic instruments, maintainers used sound, heat, smell, touch, motion, leakage, wear, and fracture surfaces to identify machine problems. Such observations depended on experience with normal behavior. For example, an unusual bearing temperature becomes meaningful when we know the normal temperature for the same load and operating conditions. These observations provided useful diagnostic signs, although comparison between observers and sites was difficult.

The steam-engine indicator provided a recorded view of an internal process. Pressure in the cylinder acted on the instrument, and the recording mechanism related pressure to piston travel. Boulton and Watt introduced the Watt indicator in the late eighteenth century for engine adjustment and assessment of work. Spring indicators, including the Richards design, improved response on high-speed engines.[1, 2, 3] An engineer could now compare recorded cycles from different cylinders, loads, or valve settings.

We can identify several familiar diagnostic requirements in indicator practice. The measurement needs reference axes and calibration. The pressure trace must be related to the phase of the engine cycle. The instrument itself has a response, including friction, spring movement, and the effects of connecting tubing. Finally, the diagram needs a model of normal operation. For example, a particular curve may suggest late valve timing, but the curve alone does not establish that explanation.

Maintenance records added observations over longer periods. Running hours, lubrication, inspections, replacements, and recurring symptoms could be related to the same machine. Although such records were often local and incomplete, they made comparison with previous condition possible. We use the same principle in fleet analysis when we combine a current measurement with component identity, age, load, environment, and repair history.

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

Telegraph and cable maintenance required the localization of faults that could be far from the test station. An open conductor, short, insulation leak, crossed pair, or intermittent contact first had to be classified. Continuity and insulation measurements provided initial distinctions. Murray- and Varley-loop methods

then used resistance ratios and known cable properties to estimate the location of a fault.[4, 5] The measurement at an accessible terminal became evidence about an inaccessible part of the cable.

Consider the reasoning involved in a loop test. We connect the faulty conductor to a suitable sound return conductor, apply a source, balance the bridge, and infer position from the resistance relation. This inference depends on assumptions about conductor resistance, cable length, joints, and fault resistance. Temperature changes, parallel paths, or multiple faults can invalidate those assumptions. A disagreement between the estimate and the physical fault location may therefore indicate a problem in the measurement model.

Technicians also isolated cable sections, exchanged pairs, compared measurements from opposite ends, and applied signals at different points. These operations provided additional evidence before excavation or repair. Time-domain reflectometry uses a different stimulus: a propagating pulse produces reflections at impedance changes. The return time helps estimate distance, while the reflected waveform provides information about the discontinuity. In both cases, the test is designed to make location observable.

A successful cable test has a limited meaning. The line behaved acceptably at the voltage, frequency, temperature, and load used during that test. An intermittent fault may still occur under moisture, vibration, or service traffic. We therefore need to preserve test conditions and compare them with the conditions under which the reported failure occurred. This requirement also applies to modern buses and high-speed links.

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

Electrical meters made voltage, current, resistance, and continuity available for routine measurement. Oscillographs and cathode-ray oscilloscopes added a view of changes over time that a meter could average out. Tektronix was founded in 1946 and sold its first 511 oscilloscope in 1947.[6, 141] Hewlett-Packard developed a related range of oscillators and measurement instruments.[8] Its 1972 interface-bus article describes the connection of programmable instruments into automated measurement systems.[126]

An oscilloscope trace is produced by both the device and the observation path. Triggering selects the event around which the trace is captured. Bandwidth, sampling, coupling, and record length determine which changes can be represented. The probe adds capacitance and changes grounding and loading. For example, unsuitable settings may hide a short transient or make a signal appear abnormal. Instrument guidance consequently treats acquisition settings and probe choice as parts of measurement.[7, 125]

Test points connect measurements to design information. A named node with an expected voltage, waveform, and tolerance can be compared with its specification. Schematics and service procedures provide the relationships between such nodes. Checking power and clock before local logic is useful because a fault in either can affect many downstream observations. An incorrect output may be a consequence of an earlier problem in the same path.

We can also compare observations made with different probes, ranges, time bases, loads, or reference channels. If the apparent anomaly changes with the instrument configuration, we need to examine the measurement path. This is similar to changing a software logging or sampling configuration when investigating an execution artifact. The observation mechanism is itself a possible object of diagnosis.

Measurement uncertainty and traceability

Calibration does not make a measured value exact. The Guide to the Expression of Uncertainty in Measurement, developed from Recommendation INC-1 (1980) and issued as JCGM 100:2008, provides a framework for evaluating uncertainty from a measurement model.[113] For example, a small temperature difference between two devices may be less informative than it appears if sensor and acquisition uncertainties are comparable to that difference. Near a pass/fail limit, the decision rule and uncertainty should be recorded together.

Metrological traceability has a specific meaning. The International Vocabulary of Metrology relates a measurement result to a reference through a documented calibration chain, with uncertainty contributed at each step.[123] This differs from tracing a service record to an asset serial number. Both are useful in diagnostics, but they establish different relationships: one concerns the measurement reference, and the other concerns the identity and history of the object measured.

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

Early electronic computers exposed much of their state through panels, lamps, switches, and accessible circuitry. Test programs exercised arithmetic and memory, while oscilloscopes showed pulses and timing. Maintenance included inspection of relays, tubes, connectors, and power supplies. The 1947 Harvard Mark II logbook preserves a moth found at a relay.[9] The record is also an example of associating a physical finding with an operational incident; it should not be treated as the origin of the word debugging.

An incorrect calculation in an early computer could originate in a program, wiring, timing, a marginal tube, or a memory location. Investigators used repeated operations, known data patterns, manual calculations, signal tracing, and module substitution to distinguish these possibilities. Accounts of Whirlwind describe maintenance as part of everyday operation and show that characteristic sounds could also provide information about machine activity.[10, 11] Logical and physical observations were analyzed together.

Marginal checking deliberately varied operating conditions to expose reduced tolerance. For example, a component might pass at nominal voltage and fail when the voltage was changed within a controlled test range. The result showed sensitivity to that condition before the same problem necessarily appeared in normal operation. Such a test needs stated limits because excessive stress can create damage and change the object being diagnosed.

PART II

Reliability becomes an engineering discipline

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

Periodic checkout could not detect every error during computation. Checking circuits therefore operated together with the functional circuits. Parity uses redundant information to detect specified changes in a word. Checksums, residue codes, duplicated computation, and comparison circuits apply related ideas. In each case, an expected relation becomes a test that the machine can perform during operation.

Hamming's 1950 paper describes codes that use a pattern of parity checks to locate and correct errors within the code's capability.[12] The syndrome provides more information than a general failure indication. In an ECC memory system, the controller can use it to classify or correct an error and may record the associated address and status. We still need the code, mapping, and platform rules to interpret this information. A syndrome locates an error relative to an encoding; it does not directly identify a physical failure mechanism.

The checker can also fail. Redundant encoding, complementary signals, duplicated computation, and self-tests provide different ways of reducing undetected errors. Their effectiveness depends on the assumed faults and on independence from shared power, clock, environmental, and design problems. Von Neumann's work on reliable systems constructed from unreliable components and later work on modular redundancy provide part of the theoretical background.[13, 14] We need to include the checking structure in the system being analyzed.

Correction allows computation to continue and may also provide evidence for a later investigation. Repeated corrections at related addresses can suggest a device or channel problem when address mapping and operating context are available. An isolated correction may have a different explanation. We therefore distinguish successful correction from diagnosis: restoring a value does not establish why the original value became incorrect.

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

Redundancy provides additional results or service paths that can be compared or substituted. Two disagreeing results establish a difference but may not identify the correct result. Triple modular redundancy uses a voter to mask one discrepant module under the assumed fault conditions.[14] Standby units, replicated channels, spare rows, redundant supplies, and mirrored storage extend the same general idea to other system boundaries.

The useful independence of redundant components must be examined. Two units can share a design defect, input, power source, environment, or maintenance error. The voter can fail, and several sensors

can be exposed to the same physical hazard. Fault containment addresses how a fault or its resulting error can propagate. The architecture needs boundaries and procedures for detection, isolation, and recovery appropriate to those propagation paths.

SAGE combined marginal and parity checking with duplex computers to support continued operation.[15, 16] Their availability depended on maintenance and recovery as well as component failure rates. Apollo guidance-computer documentation also describes self-checking and alarm handling within the flight system.[17, 18] Restarting selected work while preserving essential state could be part of the intended response to an abnormal condition.

Graceful degradation means retaining selected services when full operation is no longer possible. We need to specify which functions have priority, which resources can be removed, and what state recovery must preserve. Redundancy is useful only together with these decisions. A spare component does not by itself define when it should take over or whether the remaining evidence will support diagnosis.

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

Reliability engineering made possible failures objects of analysis before operation. Failure modes and effects analysis examines a component or function, its failure modes, and their local and system effects. Fault-tree analysis starts with an undesired event and examines combinations of contributing events. NASA reliability guidance and military procedures document these methods.[20, 22, 23] The NASA Systems Engineering Handbook places analysis alongside requirements, verification, validation, and technical assessment.[21]

These analyses also help design diagnostics. They can identify where a detector is needed, which candidate faults must be distinguished, and which information a recovery procedure requires. However, the resulting model is limited by the faults and interactions considered. Manufacturing variation, maintenance, aging, and combinations of environmental conditions may introduce cases outside that model. A complete-looking diagram should not be confused with demonstrated completeness.

Burn-in and environmental stress screening expose units to selected conditions intended to reveal latent defects before deployment. Temperature cycling, vibration, humidity, voltage, and load may expose weak connections, contamination, or inadequate timing margin. The bathtub curve describes one possible population pattern of early failures, useful life, and wear-out. NIST's reliability guidance treats such hazard shapes as models.[24] We should not assume that every technology or population follows the same curve.

What physically fails

A failure mechanism describes how a problem develops. In interconnects and packages, examples include corrosion, solder fatigue, cracked vias, weak bonds, delamination, tin whiskers, and connector

fretting.[109, 129] Within semiconductor devices, mechanisms include electromigration, stress migration, dielectric breakdown, hot-carrier degradation, and bias-temperature instability.[129, 130] Radiation can cause soft errors.[48] Memories also have retention, disturbance, and wear mechanisms.[50, 109, 142] Mechanical systems add fatigue, creep, and wear.[129] Electrical malfunction can arise from power and ground noise, crosstalk, impedance changes, jitter, skew, or insufficient timing margin.[140, 143]

Several mechanisms can produce the same observed mode, such as an open circuit, short circuit, delay, incorrect bit, or reset. One mechanism can also produce different modes as conditions change. FMEA organizes modes and their effects. Physics-of-failure analysis relates material, geometry, stress, environment, and time to a mechanism.[107] Laboratory analysis then examines the corresponding physical evidence. Identifying a failed FRU or reading an error code completes only part of this analysis.

The results of an investigation can be used to revise design rules, screening, supplier requirements, firmware, maintenance, and training. Historical accounts of fault-tolerant computing show the related development of architecture, validation, and service practice.[25] Replacing the affected component restores operation. Feeding a confirmed mechanism back into these activities can reduce recurrence and improve the detection of similar problems.

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

Service manuals made design information available as procedures. They included block diagrams, expected signals, adjustment instructions, error codes, test points, and parts lists. A field-replaceable unit defines how far service diagnosis needs to proceed before a component can be exchanged. For example, field service may identify a board, whereas a repair depot later identifies the defective part on that board. The required depth of diagnosis depends on the service objective.

Substitution with a known-good unit is a controlled comparison only to the extent that other conditions remain the same. A replacement may have different firmware or a different revision. Removing it may disturb a connector, and reseating may temporarily remove the symptom. A power supply may also behave differently under the replacement load. We therefore record the original symptom, the substitution, the verification conditions, and any later confirmation of the mechanism.

Preventive maintenance uses scheduled cleaning, lubrication, calibration, adjustment, or replacement to reduce the chance of failure. Condition-based maintenance uses measured condition to help determine when intervention is needed. Both require information about the asset and its operating history. For example, a temperature threshold has a different meaning for different duty cycles, ambient conditions, and sensor locations. The maintenance rule needs the same context as the measurement.

Service communities also accumulated knowledge of recurring symptoms, affected serial-number ranges, and characteristic component problems. Bulletins and databases made some of this knowledge reusable.

Structured codes support comparison across many cases, while free-text notes preserve observations that the codes do not express. Both are useful when a subsequent investigator needs to distinguish a successful repair from a confirmed explanation.

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

Mainframe RAS made reliability, availability, and serviceability properties of the whole system. IBM System/360 documented machine-check interruption and status that software could examine after a detected malfunction.[26] Later systems extended checking, retry, sparing, partitioning, and error logging. These mechanisms supported continued service and repair, while also generating artifacts for diagnosis.

IBM describes extensive self-checking and self-recovery as aspects of RAS.[28] A machine-check record connects hardware detection with software policy. Checking logic identifies a condition, and registers or logout areas preserve available status, address, unit, and validity information. Firmware or the operating system then decides whether to retry, remove a resource, terminate work, or stop the machine. Validity is particularly important because a fault can leave only part of the record trustworthy. IBM's RAS literature describes these relationships between checking, recovery, and service information.[27]

First-failure data capture preserves evidence before recovery changes it. A retry, reset, cleared latch, or reinitialized channel may remove the state needed to explain the original event. Service processors provided a separate path for collecting information and controlling maintenance when the main system was unavailable. Their independence was relative to particular failures, but the approach anticipated later baseboard management controllers.

Error logs also supported recurring-error analysis, thresholds, online diagnostics, and resource deconfiguration. Tandem and Stratus systems used different combinations of replication, failover, and online service.[30, 31] Remote service added another requirement: the observation and control interface needed trustworthy identity, version information, and authorization. A path used to inspect the machine could also be used to alter its operation.

11. Diagnostic programs and automatic test equipment

A diagnostic program applies a workload or data pattern and compares the observed result with an expected result. It may exercise arithmetic, memory, channels, buses, or a complete configuration. Its coverage depends on which conditions are reached and which errors become observable. A passing test establishes agreement for those conditions. It does not establish that untested behavior is correct.

Automatic test equipment applies stimuli, measures responses, compares them with limits, and records results. In manufacturing, it can contact a board or device, apply vectors, measure analog characteristics,

and classify units. In a service depot, the same organization makes testing more repeatable. Instrument programming links configuration, stimulation, acquisition, comparison, and reporting into a reproducible sequence.

The test arrangement also needs diagnosis. A reported failure can arise from the device, contact resistance, a socket, fixture calibration, tester timing, or incorrect limits. Useful traceability connects the result to the unit, lot, test-program version, instrument state, and environment. A false failure can reject a usable part, while a false pass can release an affected part. Coverage and measurement integrity therefore need separate examination.

Manufacturing results and field returns provide complementary evidence. A yield change can direct attention to a process or material change. A returned unit may reveal a fault missed during production testing. Repeated no-fault-found results may indicate intermittent behavior or differences between the test and service conditions. Connecting these records requires persistent component identity and compatible descriptions of symptoms and tests.

PART III

Diagnosing digital hardware

12. Logic analyzers, emulation, and pre-silicon and post-silicon debugging

A digital system may require simultaneous observation of many signals. An oscilloscope shows electrical waveforms, while a logic analyzer samples channels and interprets their voltages as logical states. Hewlett-Packard's two-channel 5000A logic analyzer recorded clocked logic sequences in 1973.[32] Logic state analyzers also appeared that year; the later 32-bit 1610A illustrates the development of selective tracing for complex state flow.[33] Such instruments made it possible to relate an incorrect result to activity on the bus or control lines.

Trigger conditions select the sequence to preserve. For example, an address, data pattern, or combination of control signals can trigger capture of the preceding and subsequent activity. Timing analysis samples with an acquisition clock independent of the target; state analysis uses a target-related clock or strobe. Thresholds, loading, skew, sampling, and buffer depth affect both. A trace obtained with the wrong clock or channel assignment can be internally consistent and still describe the wrong events.

In-circuit emulation gave a development system control of the processor within its target environment. The emulator could replace or intercept the processor and expose registers, memory, and bus activity. Intel's ICE-80 for the 8080 is a documented example from 1975.[34] Operations familiar from software debugging, such as stopping, stepping, and inspecting state, were thus connected to physical signals and timing.

Increasing integration made many internal buses inaccessible at package pins. Debug functions moved into the device, and selected trace data was carried out through buffers and dedicated interfaces. This access has an implementation cost in area, pins, bandwidth, storage, power, and security. We can therefore consider observability part of the design: the required diagnostic operations influence which internal state must remain accessible.

Hardware debugging also moved to executable models before fabrication. Rogin and Drechsler's 2010 book describes electronic-system-level debugging using SystemC, including early error detection, exploration, design understanding, and automatic failure isolation.[111] Here the object being examined is a design model. Its execution, properties, program slices, and structural views provide diagnostic artifacts before a physical device is available.

Counterexamples and post-silicon validation

Formal verification adds a diagnostic artifact. Biere, Cimatti, Clarke, and Zhu's 1999 work on bounded model checking searches finite executions for property violations using satisfiability solving.[121] A counterexample gives a sequence to analyze in the model. Failure to find one within a selected bound does not by itself prove unrestricted correctness. The model, property, initial-state assumptions, and bound are part of the result.

After fabrication, post-silicon validation examines whether the implemented design behaves correctly under actual execution. Mitra, Seshia, and Nicolici distinguish this activity and its debug difficulties in their 2010 account.[120] Production testing primarily screens manufactured units for targeted defects; post-silicon validation also seeks design errors and interactions missed before fabrication. Limited internal visibility, nondeterminism, and differences between models and silicon can make reproduction and localization difficult. The two activities overlap, but a production-test pass does not replace design validation.

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

Integrated circuits reduced physical access while increasing the amount of internal state. Production testing uses fault models to make the problem manageable. Examples include stuck-at, bridging, delay, and memory-coupling faults. Such a model describes behavior useful for test generation. It need not be a literal description of every physical defect that the resulting test can detect.

J. Paul Roth's 1966 D-algorithm provides a systematic method for activating a modeled fault and propagating its effect to an observable output.[35] Automatic test-pattern generation develops this approach for larger circuits and additional models. Sequential logic adds a difficulty: the required internal state may be hard to reach through normal inputs, and the resulting error may remain hidden from external outputs.

Scan design connects storage elements into shift paths for test. We can load an internal state, apply functional clocks, and shift out the resulting state. Early scan-related work, including Williams and Angell's 1973 paper, made controllability and observability explicit design concerns.[114] This reduces dependence on long functional sequences, but introduces costs in area, timing, routing, power, test time, and access control.

Coverage always refers to a stated model and test implementation. High stuck-at coverage does not establish equivalent coverage for delay, analog, power-integrity, or design faults. Timing, voltage, compression, and masking can change whether an error is observed. Failing patterns also need to be related to the layout and candidate physical structures before they become evidence of a particular defect.

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

Built-in self-test places stimulus generation, response evaluation, or both inside the device. Logic BIST can use pseudorandom patterns and compacted signatures.[36] Memory BIST applies sequences of reads and writes selected for memory fault models, including addressing and coupling faults.[127] Analog and mixed-signal devices require corresponding test methods. Built-in loopback testing of RF transceivers is one example.[138] These techniques respond to circuit complexity and reduced external access, while the test must remain appropriate to the device and the faults considered.

Power-on self-test made automatic hardware checking familiar to personal-computer users. The IBM PC Guide to Operations describes startup tests, audible indications, and displayed error codes, while the Technical Reference provides implementation details.[38, 39] The tests determine whether sufficient processor, memory, and peripheral functionality is available to proceed with startup. POST can be implemented largely in firmware; it is not synonymous with dedicated on-chip BIST circuitry.

Self-test depends on hardware that may itself be affected. A failure in power, clock, reset, ROM, or the reporting path may prevent any useful result. Staged testing limits the amount of state trusted at the beginning. Beep codes, lamps, and diagnostic displays provide alternative reporting paths as additional resources become usable. A missing report therefore needs a different interpretation from a reported test failure.

A self-test result should identify the test version, configuration, conditions, subtest, and expected and actual response. It should also state whether the test changes or destroys stored data. A compacted signature can hide an erroneous response when it matches the expected signature, a condition called aliasing.[36] Consequently, a matching signature provides evidence within the test’s fault model and compaction assumptions, rather than a general proof of correct operation.

15. Boundary scan and standardized access: JTAG and successors

Surface-mount devices and multilayer boards made direct probing of interconnects increasingly difficult. IEEE 1149.1 placed boundary-scan cells near digital pins and standardized a test-access path. Board interconnects could then be controlled and observed without contacting every node. The first edition appeared in 1990.[37] Access to internal test structures became part of the interface between device and board diagnosis.

Boundary scan can load instructions and data, sample pins, apply test values, and bypass devices in a chain. It can expose interconnect opens, shorts, and assembly problems after packaging. Test-access ports have also been used to reach programming, core-debug, and internal-instrument functions. These additional capabilities depend on the device and should not all be assumed from the presence of a JTAG connector.

Successors and extensions

Later standards extend access to other structures. IEEE 1500 specifies testability methods for embedded cores. IEEE 1687, commonly called IJTAG, specifies access to embedded instruments and descriptions of instrument networks and procedures. IEEE 1838 specifies test-access architecture for three-dimensional stacked integrated circuits.[103, 104, 105] These standards address different diagnostic boundaries within a device or assembly.

The standards complement IEEE 1149.1. For example, a board-level test-access port may provide access to an IJTAG network inside a device. Embedded-core wrappers provide another level of control and observation, while stacked-die access crosses physical die boundaries. We can see the test point moving from an externally accessible node into a structured network of internal access paths.

A standard interface still requires correct configuration and descriptions. Device order, reset behavior, voltage domains, and power state affect a scan chain. A failure near the beginning of the chain can prevent access to later devices. Incorrect boundary descriptions can also produce misleading results. We therefore need to test the access path and distinguish its failure from a failure in the circuitry being examined.

Diagnostic access can also change or expose system state. Halting a core, reading memory, modifying registers, and programming storage are useful during development and repair. In a deployed system, these operations require appropriate lifecycle states, authentication, access restrictions, and service procedures. The diagnostic interface is part of both the maintenance design and the security design.

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

Memory errors have several possible mechanisms, including fabrication defects, timing, power noise, temperature, radiation, retention loss, wear, and interaction between cells. Walking patterns, address tests, March algorithms, and stress tests exercise different modeled behaviors. Error locations may help identify a device or channel, but only when we know the controller’s address mapping, interleaving, and sparing. A logical address is not automatically a physical cell coordinate.

Parity detects specified error patterns, while ECC adds correction within the code’s capability. Scrubbing reads memory and corrects or rewrites affected data to reduce the accumulation of errors in a codeword. Reports of corrected errors remain useful diagnostic artifacts. Their interpretation requires address, channel, component, workload, temperature, and time information. No reported corrections can mean no detected errors, disabled reporting, or a fault outside the checking mechanism.

Soft-error research connected incorrect memory state to physical processes that did not require permanent device damage. May and Woods identified alpha particles from radioactive contaminants in packaging materials as a source of dynamic-memory errors.[115] Ziegler and Lanford examined cosmic-ray

effects and their contribution at sea level.[46, 47] These are different radiation sources. Later field studies examined error behavior in deployed DRAM populations,[49] while JEDEC specified measurement and reporting methods for soft errors.[48]

RowHammer demonstrated another mechanism: repeated activation of DRAM rows can disturb stored values in other rows. The 2014 study showed this behavior in contemporary memory modules.[50] The problem connects an access sequence with device physics and can have security consequences. Its diagnosis and mitigation require information across the memory cells, controller, firmware, operating system, and workload.

17. Storage diagnostics: media errors, SMART, NVMe, and RAID

Storage devices translate between physical media and logical I/O. Remapping, retries, ECC, wear leveling, and firmware recovery can return successful I/O despite an underlying problem. Host errors and internal device records consequently provide different observations. To relate them, we need the device model, firmware, and the meaning of the reported counters and events.

SMART provides health attributes, status, and self-test information, and management profiles describe ways to expose diagnostic functions.[53] Some attributes and thresholds remain vendor-specific. Field studies found useful correlations between selected SMART indicators and failures, but also found failed drives without strong preceding indications.[51] We can use such measurements to prioritize investigation, while preserving the distinction between a population association and a prediction for one drive.

RAID can reconstruct data after some failures, but reconstruction depends on the remaining data being readable. Latent errors on surviving drives may become visible only when redundancy has already been reduced. Patrol reads, scrubbing, checksums, and end-to-end checks can expose these errors earlier. Successful recovery should be recorded with its context because it may otherwise hide the original difficulty from the host.

Flash storage introduces program/erase wear, retention, read disturbance, controller behavior, and power-loss handling. The flash translation layer also changes the relationship between a logical block and its physical location. Media errors, spare capacity, temperature, workload, unsafe shutdowns, and firmware identity provide different views of condition. A single health percentage cannot preserve all these distinctions.

NVMe diagnostic records

NVMe extends the storage evidence available through standardized logs and commands. The 2.0c specification includes SMART/Health Information, Error Information, device self-test, host- and controller-initiated telemetry, and a Persistent Event Log.[116] These records answer different questions. A health field summarizes condition, while an event log preserves selected changes. Telemetry payloads can require

vendor-specific decoding. Supported features, controller identity, firmware, log generation, and capture time must accompany the bytes; an unsupported log is different from an empty supported log.

18. Processor and interconnect diagnostics: machine checks, AER, CXL, and containment

Processors use parity, ECC, protocol checking, watchdogs, timeout logic, and other mechanisms to detect selected internal and external errors. Intel's machine-check architecture, for example, provides banks of status, address, and additional information with defined validity and recoverability flags.[44] Enhanced MCA reporting also supplies error-severity information.[45] Firmware, the operating system, a hypervisor, or a management controller may subsequently process the record. The point where an event is reported may differ from the point where it originated.

Interconnect error handling adds further transformations. A link may retry transmission, retrain, report a protocol error, or become unavailable. PCI Express Advanced Error Reporting distinguishes correctable errors and uncorrectable errors classified as nonfatal or fatal. These classifications help determine a response. They do not identify a physical cause: a receiver's report may result from a transmitter, channel, connector, power, clock, or protocol problem.

Silent data corruption and mercurial cores

Silent data corruption is a limit of this checking approach. A processor may produce an incorrect result without an applicable machine check, ECC report, or useful counter. Google described rare, intermittently incorrect cores as mercurial cores. Facebook, now Meta, also reported CPU defects that escaped conventional hardware error reporting in a large fleet.[101, 102] Instruction completion and successful retirement are therefore different observations from verified computational correctness.

Additional checks can expose such behavior. The Google and Facebook reports describe testing and validation beyond ordinary hardware alarms.[101, 102] Repeated tests, comparison of results, application checks, and analysis by core or machine can help localize the affected resource. The test conditions remain important because a fault may depend on an instruction sequence or operating condition absent from an earlier test. Other information, such as aging and temperature history, may then help investigate the mechanism.

Signal-integrity margining

Signal-integrity tests examine the available operating margin. Eye diagrams, jitter analysis, bit-error-rate tests, receiver stress, and voltage or timing sweeps provide different measurements. For example, a link can have no observed protocol errors during a test while its usable sampling region is small. Margin measurements help distinguish these situations, although they are still conditional on the link settings and environment.

PCI Express 4.0 introduced Lane Margining at the Receiver. Software can move the receiver sampling point and obtain information about available timing and voltage margin on an operating link.[106] The result needs lane identity, speed, equalization state, topology, temperature, workload, and firmware context. A satisfactory result at one receiver or operating point does not exclude a shared power or clock problem elsewhere.

Containment can stop an error from affecting other work. Possible responses include marking data as poisoned, aborting a transaction, resetting a link, retiring a page, or removing a core from service. These responses also change the diagnostic state. We need to capture the original status, mapping, time, and configuration before recovery removes them whenever the system permits such capture.

Consider a failure reported at several levels. An application terminates, the kernel records a machine check, firmware preserves an error record, and the BMC reports a voltage excursion. A fleet comparison may show that similar events occur on one board revision. These artifacts support different steps in the analysis. Their relationship must be established through time, identity, topology, and correct decoding; several reports of the same propagated error are not necessarily independent confirmations.

CXL memory and error propagation

Compute Express Link introduces further relationships between hosts, memory devices, and management software. The consortium's 2024 RAS white paper covers event records, component identifiers, poison handling, scrubbing, sparing, and test operations across CXL 2.0 to 3.1.[117] A device may record a media event before the host consumes poisoned data. Correlation can require translation between host and device physical addresses. We therefore distinguish detection, recording, consumption, and recovery times. Optional features must be discovered on the actual device before their absence is interpreted as a diagnostic result.

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

Operational diagnosis often localizes a problem to a component, path, or condition. Physical failure analysis continues toward the affected material or structure and the mechanism involved. The ASM Handbook describes failure investigation as a systematic connection between evidence, mechanism, root cause, and corrective action.[108] The laboratory can therefore connect a fleet symptom with manufacturing, design, handling, or operating conditions that should be changed.

Preserving evidence before preparation

An investigation begins with the received condition, photographs, identity, configuration, event history, and electrical verification. Candidate explanations guide the order of examination. Noninvasive inspection generally precedes decapsulation, sectioning, polishing, or delayering because preparation can remove

evidence or introduce new artifacts. Locations found by electrical tests, scan diagnosis, imaging, or thermal measurements need to be related to the same schematic and layout coordinates.[108, 109]

Examining buried structures

Radiography and X-ray computed tomography can reveal buried features such as voids, bridges, bond geometry, inclusions, and misalignment when resolution and material contrast permit. Acoustic microscopy provides complementary information about interfaces and delamination. The 2011 multi-scale X-ray work illustrates nondestructive inspection of three-dimensional integrated structures for process development and failure analysis.[110] The resulting image is still limited by attenuation, orientation, reconstruction, and resolution. Absence of a visible feature does not prove absence of the suspected defect.

Localizing electrical behavior

Emission microscopy uses photons associated with electrically active sites to help localize abnormal behavior. Thermal and stimulation methods provide other spatial observations. For example, lock-in thermography examines a modulated thermal response, while optical or thermal stimulation can relate a change in measured electrical behavior to a location. The ASM desk reference describes these methods within a coordinated isolation process.[109] A bright or hot site is a diagnostic indicator whose mechanism needs further examination.

Examining the localized site

Scanning electron microscopy provides surface and cross-sectional images. Backscattered-electron contrast and energy-dispersive X-ray spectroscopy add information about material composition. A focused ion beam can expose a selected structure, prepare a sample for transmission electron microscopy, or modify a circuit to test a hypothesis. These operations can reveal cracks, voids, residues, corrosion, or damaged interconnects. They can also introduce beam damage, redeposition, charging, and preparation artifacts that must be recorded.[109]

A supported mechanism connects the failing electrical condition, the localized site, a relevant physical finding, and the conditions capable of producing it. Reproduction or a verified corrective action can strengthen that connection. An image of damage alone may show an incidental feature or a consequence of the failure. Conversely, a fleet correlation can identify an affected population without identifying the physical process. Laboratory analysis and operational evidence answer complementary questions.

PART IV

Embedded, vehicle, aerospace, and physical-asset health

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

Firmware provides diagnostic functions before the operating system starts. It initializes components, discovers topology, runs early tests, and configures reporting. It may also handle platform errors during operation. Firmware therefore transforms device-specific status into information available to later software. We need to examine this transformation when a hardware event is missing, summarized, or interpreted differently at another level.

The UEFI Platform Initialization specification defines status-code reporting during firmware execution.[40] UEFI provides the boot and runtime environment, while ACPI Platform Error Interfaces describe error sources and reporting mechanisms, including the Hardware Error Source Table and error-record serialization.[41, 42] Windows Hardware Error Architecture provides operating-system handling and structured records for consumers of hardware-error information.[43] These interfaces make records reusable across tools without removing their platform-specific meaning.

The Common Platform Error Record (CPER), specified by UEFI, separates an error record into a header, section descriptors, and typed sections with validity information.[41] ACPI Generic Hardware Error Source mechanisms provide paths through which platform error status can reach an operating system.[42] A standard container makes records exchangeable; it does not guarantee that every section is present or identifies the initiating component. Raw sections and validity flags should remain available beside the decoded summary.

Persistent error storage can preserve a record across reset for retrieval during a later boot. Persistence alone is insufficient. We also need to know when the record was created, whether it has already been processed, when it is cleared, and how repeated events are represented. Early boot time may be uncertain, and a component may have been replaced before the record is read. These conditions affect the reconstructed event sequence.

Firmware, microcode, and board revisions can change what is reported. Events may be masked, remapped, aggregated, or rate-limited before reaching the operating system. We should therefore preserve these versions together with raw status and its decoded interpretation. Keeping both forms allows a later decoder to revisit an earlier artifact without treating the original interpretation as the only possible one.

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

Embedded systems often need to be observed while control continues. Debug modules provide operations such as halt, step, breakpoint, watchpoint, and state access. Trace mechanisms record selected control flow, accesses, exceptions, or system events. These records can be less intrusive than printing messages, but they still have bandwidth and storage limits. Compression and filtering determine how much of the execution can be preserved.

The Nexus 5001 standard describes a scalable embedded-debug interface. Arm CoreSight organizes trace sources, interconnects, funnels, buffers, and sinks.[54, 56] Program-flow trace can encode changes in control flow so that a decoder reconstructs execution using the corresponding program image.[55] The ratified RISC-V Debug Specification 1.0 defines external-debug mechanisms across implementations.[57] Debug access and continuous execution trace are related capabilities but are not interchangeable specifications.

A trace is a selected execution history. Trigger windows, filters, buffer size, timestamps, synchronization, and data loss determine what can be reconstructed. A buffer frozen at a trigger may preserve the preceding events. A streaming path may preserve a longer interval but lose data when its transport is overloaded. We need the capture configuration, matching executable image where required, and explicit indications of loss to distinguish an absent event from an unrecorded event.

Production debug access can expose keys, private data, and control state. Its use depends on the device lifecycle and the identity and authority of the investigator. The design should specify what can be enabled, read, changed, and audited, including recovery when the main processor is unavailable. Diagnostic access remains part of the device’s trust boundary even when it is used only for service.

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

Automotive diagnostics combined local mechanical and electrical examination with controller-generated records. Engine-control units detect selected sensor, actuator, circuit, and emissions conditions, store trouble codes, and provide information to a scan tool. U.S. and California requirements made OBD-II broadly applicable to 1996 and later light-duty vehicles.[58, 59] The resulting interface standardized a significant part of service evidence, especially for emissions-related diagnosis.

SAE J1979 specifies diagnostic test modes and access to emissions-related information, including codes and freeze-frame data.[60] A freeze frame stores selected operating parameters associated with a recorded condition. Examples include speed, load, and temperature. These values describe the controller’s available observations. A defective sensor can affect both the control decision and its recorded context, and limited storage may determine which event is retained.

Vehicle networks extended access to additional controllers and functions. CAN provides communication between controllers, while Unified Diagnostic Services include sessions, data identifiers, routines, security access, and reprogramming. SAE J1979-2 specifies OBD on UDS.[61] Diagnosis may consequently involve the sensor, wiring, controller, network, calibration, and software that interpret the same physical process.

A diagnostic trouble code identifies a detected condition. It does not necessarily identify the component that should be replaced. For example, an oxygen-sensor-related indication may involve wiring, an exhaust leak, fuel delivery, or combustion. The code should be combined with its definition, live data, electrical measurements, controlled tests, and service history. A repair is then verified under conditions relevant to the reported symptom.

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

Spacecraft and aircraft have limited opportunities for physical inspection and repair during operation. Built-in tests and fault detection, isolation, and recovery must work within constraints on power, mass, computation, communication, and intervention. NASA’s FDIR guidance and draft Fault Management Handbook relate fault management to requirements, architecture, testing, operations, and anomaly response.[62, 63] Diagnostic capability must therefore be planned before the particular failure occurs.

FDIR separates several functions. Detection establishes that a specified abnormal condition is present. Isolation narrows the responsible component or region. Recovery may retry, reconfigure, enter a safe mode, or remove a function. Further diagnosis can continue on the ground with additional models and records. Preserving snapshots and event history is useful because recovery can remove the state that first distinguished the fault.

Integrated vehicle health management combines observations and reasoning across subsystems. A NASA framework relates vehicle-health functions to information integration.[64] A commercial-aircraft IVHM study examines technologies for monitoring, diagnosing, and predicting system health.[65] Integration requires common identity, time, units, mode, and configuration. A temperature, a model residual, and a maintenance action cannot be related reliably if they describe different operating states or different component instances.

An autonomous response must be appropriate to the evidence and the time available. A false detection may cause an unnecessary or hazardous reconfiguration, while a missed fault may allow propagation. Persistence rules, voting, inhibition conditions, thresholds, and safe-state design affect these outcomes. We therefore analyze the detector and the response together, including what happens when the observations remain ambiguous.

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

Condition monitoring examines machinery during use. Vibration amplitude, phase, waveform, and spectrum can provide indications of imbalance, misalignment, looseness, bearing problems, or gear behavior. Acoustic emission, motor current, lubricant debris, temperature, pressure, and infrared images provide additional evidence. Their usefulness depends on the mechanism and operating conditions being investigated.

Consider a vibration spectrum. Its peaks depend on speed, load, sensor mounting, orientation, bandwidth, sampling, and processing. The same peak may therefore need a different interpretation in another operating condition. Vibration-monitoring guidance emphasizes comparison with baselines and trends.[68] ISO 13374 organizes processing from data acquisition through data manipulation, state detection, health assessment, prognostic assessment, and advisory generation.[66] These are separate transformations from a measurement to a maintenance decision.

Condition-based maintenance uses measured condition to help determine when work is needed. This can reduce unnecessary intervention, but the decision depends on sensors and models. A drifting sensor may suggest deterioration, while a stuck sensor may conceal change. Sensor plausibility, calibration, missing data, and comparison with independent observations are therefore part of the diagnosis of the machine.

NIST's systematic review examines condition-monitoring technologies in industrial maintenance.[67] For a particular investigation, we can compare several kinds of evidence. A bearing problem may be supported by vibration, temperature, lubricant findings, and inspection under known load. If these observations disagree, the disagreement may identify a sensor problem, a different operating regime, or a mechanism not yet included in the explanation.

25. Nondestructive evaluation and structural health monitoring

Nondestructive evaluation obtains information about a material or structure while preserving its intended use. Visual, ultrasonic, radiographic, magnetic, eddy-current, penetrant, thermal, acoustic, and impact methods are sensitive to different properties and discontinuities. NIST publications describe such methods for installed materials and structures.[69, 70] The selected method must be appropriate to the geometry, material, and suspected condition.

An instrument indication requires interpretation before it becomes a defect finding. Geometry, surface condition, calibration, detection probability, and acceptance criteria affect that interpretation. A false indication may lead to unnecessary removal; a missed indication may leave a relevant flaw. The procedure, examiner qualification, calibration, and recorded configuration therefore accompany the image or measurement as diagnostic evidence.

Structural health monitoring uses repeated or continuous measurements of a structure. Strain, acceleration, acoustic activity, corrosion, displacement, and environmental observations can reveal change. However, different damage states can produce similar measurements, especially with sparse sensors and variable loading. A changing sensor or baseline can also imitate structural change. Additional observations may be needed to distinguish these cases.

Additive manufacturing brings monitoring into the production process itself. NIST distinguishes a measured process signature from evaluation of a flaw in the resulting material.[71] For example, an abnormal process signal can identify a region for further examination without proving that the finished part contains an unacceptable defect. The connection requires a validated model, appropriate measurements, and stated acceptance criteria.

PART V

Remote and fleet-scale hardware observability

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

Remote network management requires names for the state a device exposes. Interfaces, fans, supplies, temperatures, link status, packet errors, and configuration become managed information that another system can retrieve. RFC 1157, published in 1990, specifies SNMP operations and traps for this purpose.[72] The remote record adds distance between the physical condition, its local detector, and the investigator.

Counters preserve accumulated observations but omit much of their context. Reset, wraparound, sampling interval, and changing interface identity can create misleading trends. A receiver-error count alone does not distinguish the cable, optic, connector, transmitter, or receiver. To compute a rate, we need compatible samples, elapsed time, and knowledge of counter discontinuities. To diagnose the cause, we usually need additional evidence.

RMON, first specified in RFC 1271 in 1991, extended remote management with traffic statistics, history, alarms, and other monitoring functions.[73, 139] Model-driven streaming telemetry and gNMI subsequently added structured paths, timestamps, and subscription mechanisms.[74] SNMP also has typed values; the change should not be described as the invention of typed management data. Streaming changes collection and delivery, while schema, identity, units, and loss behavior remain essential to interpretation.

The collection system can affect what it observes. Polling uses device resources, and detailed streams consume bandwidth, memory, and storage. An overloaded agent may report late or stop reporting. Freshness, sequence gaps, and backpressure therefore need to be visible. Missing telemetry should not be converted into a zero value or interpreted as evidence that no hardware event occurred.

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

A baseboard management controller provides a management processor alongside the host. It can monitor platform sensors, record events, control fans and power, and provide console or reset functions while the host operating system is unavailable. Standby power makes some of these operations possible when normal host power is off. IPMI standardized many platform-management functions and records. Intel hosts the specification developed by Intel, Hewlett-Packard, NEC, and Dell.[137] In 2020, the promoters announced that no further updates were planned and pointed to Redfish.[75]

The BMC's independence is relative to particular fault domains. It may preserve information during a host crash but share power, buses, sensors, or update infrastructure with the host. A board-level failure may affect both. We need to identify these dependencies before assuming that an out-of-band record will remain available during every failure being investigated.

Redfish provides a resource-oriented management interface and schemas for systems, chassis, managers, telemetry, events, and updates.[76] OpenBMC provides an open-source implementation environment with documented telemetry design.[77] These interfaces make observations easier to collect and compare across systems. OEM extensions and implementation differences still require the model and schema context to remain attached to the data.

A compromised or defective management controller can misreport state or alter the host. Diagnostic trust therefore depends on controller identity, firmware integrity, authenticated access, authorization, and audit. Network isolation and update procedures also affect that trust. We need to diagnose the management path as a system with its own software, hardware, and failure modes.

Management transport and evidence identity

Inside the platform, MCTP provides message transport between management endpoints.[78] PLDM adds data models and commands, including platform sensors, effecters, events, and descriptors that relate them to entities.[118] Redfish exposes management resources to higher-level clients. These functions occupy different parts of a possible reporting path. A failure to retrieve a reading may arise in transport, discovery, the descriptor mapping, or the sensor. The visible API response alone does not distinguish them.

SPDM provides mechanisms for device authentication and measurement exchange.[119] These can support a decision about the identity and reported firmware state of a component supplying evidence. An authenticated component can still have a drifting sensor or an incorrect measurement model. Authentication therefore strengthens provenance within its scope; it does not replace calibration, plausibility checks, or diagnosis of runtime behavior.

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

At data-center scale, we analyze devices together with their physical and operational relationships. Servers belong to racks, power domains, cooling arrangements, networks, firmware groups, and workload placements. An aggregate should remain traceable to the individual unit and its components. Otherwise, a fleet trend cannot be converted into a useful inspection or test of a particular machine.

Power and temperature observations often need to be analyzed together. A hot accelerator may be associated with workload, airflow, fan control, inlet temperature, thermal-interface condition, voltage, or

firmware policy. We can compare chassis sensors, facility records, fan commands, power limits, and neighboring devices to distinguish these possibilities. A threshold detects one condition; comparison with peers may reveal another.

Fleet records allow comparison of corrected errors, retries, throttling, fan events, and replacements across groups. Possible groupings include model, lot, revision, firmware, rack, environment, and workload. A higher rate in one group can direct the investigation. It does not establish that the grouping variable is the mechanism in every affected unit. The selected population and replacement policy also influence the result.

Component identity must survive movement and replacement. Serial numbers, slots, FRU identifiers, topology addresses, firmware versions, service events, and host names need explicit relationships. For example, historical errors from a removed board must not become errors of the replacement board simply because the slot name is unchanged. Identity mistakes can create apparently convincing but incorrect diagnostic histories.

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

Hardware performance counters record selected microarchitectural events. Examples include cycles, retired instructions, cache activity, branch outcomes, stalls, and memory or interconnect transactions. Intel’s event guides specify model-dependent events and programming constraints.[79] Such counters provide observations from within the processor without requiring direct access to internal wires.

A counter summarizes events and usually loses their order. Its meaning also depends on scope, filters, and the event definition. Multiplexing estimates values from intervals when an event was enabled. Some events include speculative work, and sampling skid can separate a reported sample from the instruction associated with it. Virtualization adds further scope and scheduling conditions. Similar event names across processors do not establish identical semantics.

Trace records can restore selected sequence information. Program-flow trace, system trace, and fabric observations can relate execution to interrupts, transactions, and external events.[54, 55] Available bandwidth and buffer space determine how much history can be retained. Trigger selection, pre-trigger capture, and alignment of clocks should therefore follow the diagnostic question rather than be left as unexplained defaults.

Performance analysis uses cooperation between software and hardware. Software configures counters and supplies execution context and symbols, while hardware defines the events and capture behavior. We need the processor model, microcode, firmware, operating system, virtualization state, and sampling settings to interpret the artifact. An accurate count can still support an incorrect explanation when this context is missing.

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

Large disk and DRAM studies provide observations from deployed populations under operational workloads.[49, 51, 52] They show differences between devices, recurring errors, and limits to simple independence assumptions. The populations also have particular screening, operating, and replacement practices. We should preserve these conditions when comparing their results with laboratory reliability estimates or another fleet.

HBM studies extend population analysis to memory stacks used in accelerator systems. The 2024 field study examines errors in deployed HBM and their operational context.[86] Such work can identify associations for further investigation. A statistical association with temperature, workload, or configuration is not by itself a confirmed physical mechanism. Returned-unit analysis and controlled tests remain useful for that next step.

A health score combines several observations into one value. This is useful for triage, but the combination hides weights, assumptions, and missing data. A model trained on removed devices may learn the removal policy as well as properties of the hardware. We should also distinguish the observed failures from units whose later outcome is unknown because they were removed or the study ended. Calibration and uncertainty are necessary when the score is used to predict an individual outcome.

Population findings can guide bench testing, supplier investigation, microscopy, firmware experiments, and environmental checks. The selected tests should distinguish candidate explanations for the population pattern. This connects a fleet-level observation with a mechanism that can be examined on particular devices. Ranking affected units is useful, but it is a different result from explaining why they failed.

PART VI

From evidence to prediction and automation

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

Model-based diagnosis compares observations with expected behavior and searches for failed assumptions that explain a contradiction. Reiter's 1987 theory provides a general logical formulation.[85] Engineering models can use physical equations, qualitative relations, graphs, probabilities, or combinations of these representations. Each model determines which explanations can be expressed and tested.

A residual compares a measurement with a model prediction. Different patterns of residuals may distinguish candidate faults. This relationship also guides sensor placement and test selection. If two candidates produce the same available observations, those observations cannot identify one uniquely. We need another operating condition, measurement, or controlled intervention. Increasing the complexity of a classifier does not supply evidence that was never captured.

Fault injection introduces controlled faults or errors to evaluate the diagnostic and recovery mechanisms. It may be performed in hardware, simulation, emulation, or software interfaces. NASA studies describe injection and monitoring in embedded and fault-tolerant systems.[83, 84] The results can measure detection latency, coverage, propagation, and recovery. We must still distinguish the injected model from a physical mechanism: forcing one register bit does not necessarily reproduce an analog transient, correlated damage, or progressive degradation.

Active diagnosis distinguishes explanations by changing conditions, for example, swapping channels, varying load, changing routing, or substituting a known-good unit. Before the intervention, we specify the expected result under each hypothesis and preserve the current state. The test also needs operating limits and a way to restore configuration. Conditions that could not be held constant remain part of the comparison.

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

Prognostics estimates how condition may evolve. Remaining useful life depends on a defined functional limit, future load, environment, maintenance, and uncertainty. For example, the same observed wear can imply different remaining use under different operating plans. We should therefore present a conditional estimate and its assumptions, rather than treat remaining life as a directly measured property of the component.

Physics-based models represent processes such as crack growth, wear, or thermal cycling. Data-driven models use patterns learned from observations, and hybrid models combine both approaches. NASA's prognostics repository and simulated C-MAPSS engine-degradation datasets provide resources for evaluating methods.[81, 82] A result on a benchmark establishes performance for that dataset and evaluation procedure. Transfer to another fleet requires examination of sensors, loads, maintenance policy, and failure modes.

A digital twin relates a physical asset or process to a digital representation through data and models. NIST discusses their use in advanced manufacturing.[80] Diagnostic applications need current configuration, measured state, model parameters, uncertainty, and provenance. Geometry alone is insufficient. We need to specify which decisions the twin has been validated to support.

AI can retrieve records, summarize event sequences, identify anomalies, and suggest explanations or tests. We can relate these results to manuals, topology, telemetry, and previous cases. The diagnosis should identify supporting artifacts and unresolved alternatives. Recommended actions still require appropriate authorization, operating limits, and verification.

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

Accelerator systems combine high power density, memory bandwidth, fast links, and synchronized distributed work. A problem in an accelerator, HBM stack, optical path, PCIe connection, or cooling region may first appear as a failed or delayed training job. We need to relate the job's phase and communication topology to device records, workload placement, software versions, and environmental conditions.

NVIDIA DCGM provides GPU health, diagnostic, telemetry, profiling, and policy functions. Xid messages identify classes of GPU conditions reported through the driver.[87, 88] These are useful entry points for investigation, but the code must be interpreted in context. A reported GPU problem can involve software or another part of the system, and one code may have more than one possible cause.

Fleet diagnosis combines continuous observation with targeted tests. Meta discusses reliability practices across manufacturing, operation, and repair.[89] Alibaba Cloud's Aegis combines evidence from training jobs, GPUs, hosts, and networks.[90] We can use a reported symptom to select tests, then combine their results with the original records before repair or removal.

Automation changes the population from which it learns. Removing devices, rerouting work, and changing thresholds affect later observations. Evaluation should account for these interventions. Recording the recommendation, evidence, action, and outcome helps distinguish improved reliability from reproduction of earlier operational decisions.

PART VII

Case studies, synthesis, and future directions

34. Diagnostic lessons from consequential hardware failures

Large investigations combine evidence from different stages of a component's life. The Apollo 13 review examined the oxygen tank's manufacture, handling, testing, electrical design, and in-flight behavior.[93] Telemetry acquired during flight could not explain the event without this earlier history. The case illustrates why diagnostic identity should connect an installed component to its previous configuration and interventions.

The Challenger investigation examined O-ring sealing in cold conditions and the interpretation of earlier test and flight evidence.[94] The Aloha Airlines Flight 243 investigation considered fatigue, corrosion, inspection, and damage in an aging structure.[95] The Columbia investigation combined imagery, telemetry, recovered material, and tests to reconstruct the consequences of foam impact.[96] These cases required relationships between observations made at different physical scales and by different organizations.

Three Mile Island illustrates how incomplete or misleading indications can affect the interpretation of plant state.[135] GAO subsequently surveyed the investigations.[97] At Fukushima Daiichi, loss of power and instrumentation restricted reliable observation and control.[98] Deepwater Horizon investigations examined pressure tests, alarms, control systems, components, maintenance, and operational decisions.[99, 144] An observation system should therefore be evaluated under the failures it is expected to help diagnose, including loss or invalidity of its own signals.

The Pentium division defect provides an example of a logical design fault. Numerical tests made the incorrect result reproducible outside the manufacturer's laboratory.[100] Reproduction established the behavior, while further design analysis explained the omitted lookup-table entries involved. The case also limits a purely material account of hardware failure: a correctly manufactured device can implement an incorrect design. The diagnosis must identify the relevant design or physical mechanism and the conditions that expose it.

35. Recurring principles and future directions

Establish a baseline before interpreting deviation

We compare a measurement with a baseline appropriate to the component, configuration, workload, environment, and age. For example, a corrected-error rate or vibration level has little meaning without its operating context. A general threshold and a change from the unit's own history answer different questions.

Preserve identity, topology, configuration, and time

A syndrome needs its address mapping, a temperature needs its sensor location, and a counter needs its reset history. Serial numbers, slots, lots, firmware, calibration, replacements, and clock quality allow these observations to be related. They are part of the diagnostic evidence.

Separate detection, isolation, diagnosis, recovery, and prognosis

Detection, isolation, diagnosis, recovery, and prognosis produce different claims. A detector may identify the condition correctly while the inferred cause is wrong. Recovery may restore service without resolving that cause. A prognosis introduces assumptions about future operation. Each claim should retain its evidence and uncertainty.

Treat corrected and retried events as evidence

Correction, retry, remapping, sparing, and restart can conceal a problem from the user while leaving useful diagnostic records. Some responses consume spares or reflect declining margin; an isolated correction or retry does not necessarily imply permanent degradation. Rates, locations, recurrence, and operating conditions help distinguish these cases.

Combine passive telemetry with controlled tests

Operational records show behavior under actual use, while controlled tests can distinguish candidates that those records leave ambiguous. We need both the test's fault model and its possible effects on the device. A test result should include the conditions under which it was obtained.

Preserve first-failure data before changing state

Reset, retry, failover, reseating, and replacement change the state being investigated. Flight recorders, freeze frames, persistent records, and trace buffers can preserve information from before and during recovery. The earliest available report should be examined as evidence of detection time; it may still follow the original fault.

Make telemetry loss visible

A constant value may describe a stable quantity, a stuck sensor, or a stopped collector. Freshness, sequence information, quality flags, and explicit missing data help distinguish these possibilities. Absence of a report should remain distinguishable from a reported zero.

Diagnose the observation path

Probes load circuits, sensors drift, firmware decodes status, and management services transform records. Independent measurements, calibration, self-tests, raw artifacts, and version information can expose

errors in these transformations. We should include the observation path among the candidate sources of an anomaly.

Design for containment and degraded operation

A containment boundary limits propagation and may preserve enough of the system for further diagnosis. Page retirement, link rerouting, function removal, and safe modes are examples. Recovery should retain the original records and identify the configuration changes that follow.

Close the loop from field evidence to design and manufacturing

A confirmed mechanism can lead to changes in design, test patterns, screening, suppliers, firmware, thresholds, and service procedures. A replaced part alone does not establish that mechanism. Repeated no-fault-found returns should also be analyzed because they may reveal a limitation in the current diagnostic process.

Govern powerful diagnostic interfaces

Test access, debug modules, service processors, and management APIs can read or change sensitive state. Their use requires defined lifecycle conditions, authentication, authority, and audit. Autonomous repair adds the need to preserve the proposed action, supporting evidence, and verified result.

Instrument chiplets, packages, and materials

Heterogeneous packages introduce diagnostic boundaries between dies and shared package resources. UCIe 2.0 describes optional manageability and DFx structures for test, telemetry, and debug across the system-in-package lifecycle. UCIe 3.0 adds further management and event-signaling capabilities.[91] The specification provides mechanisms; implementation and configuration determine which observations a particular package actually exposes.

Use digital twins and AI as auditable hypotheses

A model or agent should preserve the measurements, configuration, assumptions, and competing explanations behind its output. NIST's AI Risk Management Framework provides a broader vocabulary for evaluating validity, transparency, and accountable use.[92] We can apply these requirements to the diagnostic process itself, including the effect of its recommendations on later evidence.

Connect the explanation to a mechanism

The final explanation should connect the observed behavior to a mechanism and its relevant conditions. That mechanism may involve material, energy, timing, environment, manufacture, use, or an incorrect logical design. Its value is demonstrated by evidence and, where possible, by an intervention that prevents or changes the predicted behavior.

Organize reusable diagnostic analysis patterns

We can also organize the recurring analysis techniques as patterns. Theoretical Software Diagnostics distinguishes analysis patterns, the structural and behavioral problems they identify, and the mechanisms considered during root cause analysis.[122] For hardware, comparing a device with a known-good unit is an analysis technique; recurring corrected errors form part of the observed problem; a damaged interconnect is a candidate mechanism. Keeping these levels separate helps prevent an error signature from becoming an unsupported cause label.

The catalog proposed here brings the historical practices together and connects them with the unified hardware–software perspective discussed below. It does not imply that their original developers used the same pattern vocabulary. An entry in a hardware analysis catalog could state the context, required artifacts, comparison or intervention, expected distinctions, and limitations. The resulting analysis narrative would preserve how the investigator moved from the original observations to candidate mechanisms and a verified action.

Unified computer diagnostics and hardware narratology

We can extend pattern-oriented diagnostics across the hardware–software boundary. Unified Computer Diagnostics proposes relating signals to messages in both directions, allowing Software Narratology and trace analysis patterns to be applied to hardware diagnostics.[124] This approach is also described in Theoretical Software Diagnostics as part of Hardware-Software Narratology, also called Computer Narratology.[122]

The same approach generalizes the organization of a trace beyond time. An index, a pointer relation, or a location in memory can provide another way to connect the artifacts.[124] We can therefore examine recorded events together with structures preserved in memory. Such a relationship must be established from the available evidence; address order alone does not establish execution order.

Consider a platform on which a memory controller reports an ECC error, firmware records it in a CPER section, and a crash dump preserves the operating system’s response. We can relate the controller status, firmware record, and memory artifact through source identifiers, addresses, time, and configuration. The narrative describes how the condition became observable at successive layers. Several records may concern one propagated error. Establishing the physical cause still requires evidence that distinguishes the candidate mechanisms.

Conclusion

Hardware diagnostics has accumulated methods that remain useful at different levels. Sensory inspection, electrical measurement, on-chip checking, field substitution, model-based reasoning, and laboratory failure analysis can all contribute to one investigation. New instruments extend the available artifacts and introduce additional assumptions about how those artifacts were produced.

We can follow this development through the distinctions that became observable. An indicator relates pressure to a cycle; a bridge measurement helps localize a cable fault; a logic analyzer records related digital states; an ECC syndrome distinguishes error patterns. Fleet records add comparisons across devices and operating histories. In each case, the observation needs a model, identity, and acquisition context before it supports a diagnosis.

The quality of hardware observability therefore depends on the use that can be made of its records. The records should preserve relevant state, identify invalid or missing data, remain available under the intended fault conditions, and connect logical locations with physical components. Keeping original artifacts and transformation details also allows their interpretation to change as better models and decoders become available.

AI-assisted analysis can help retrieve related cases, compare artifacts, and propose tests. The same diagnostic requirements still apply: identify the problem, consider alternative mechanisms, preserve the reasoning, and verify the result. A useful explanation allows another investigator to examine the evidence and repeat the relevant analysis or test. This makes diagnostic knowledge reusable across instruments, systems, and generations of hardware.

Appendix A. Selected chronology

Dates identify documented instruments, publications, standards, and investigations. They do not imply that every technique originated in one place. Rows are ordered by the beginning of the stated period, with an exact year before a broader period beginning in the same year. Early and late parts of a century are placed according to their stated approximate interval.

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, and sectional tests.	Faults in inaccessible lines are classified and localized from terminal measurements.
1860s	The Richards steam-engine indicator improves response on high-speed engines.[3]	Lower moving inertia allows rapidly changing cylinder pressure to be recorded more faithfully.
Late 1800s	Cable bridge-testing practices develop; later handbooks document Murray and Varley methods.[4, 5]	A physical fault location is inferred from resistance ratios and cable models.
1897	Cathode-ray tube demonstrated as an instrument for electrical phenomena.	Electrical variation gains a display based on cathode-ray deflection.
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.
1946–1947	Tektronix is founded and sells its first 511 oscilloscope.[6, 141]	Oscilloscopes become more practical tools for electrical measurement.
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.

PERIOD	DEVELOPMENT	DIAGNOSTIC SIGNIFICANCE
1950s	Marginal checking and diagnostic programs are used on operational computers.	Controlled stress exposes declining tolerance before nominal failure.
1950s–1960s	SAGE combines marginal and parity checking, duplex computers, and continuous operation.[15, 16]	Availability and maintainability become system-level properties.
1956	Von Neumann's analysis of reliable systems built from unreliable components is published, following his 1952 lectures.[13]	Redundancy and fault models become formal design concerns.
1960	Kalman formalizes state-based control concepts, including observability.[112]	State reconstruction is related to a specified model and its input/output behavior.
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 introduces System/360; its machine-check interface is documented in the cited 1970 manual.[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 the two-channel 5000A logic analyzer.[32]	Clocked digital sequences can be stored and compared at selected nodes.
1973	Williams and Angell publish testability methods using additional logic and test access.[114]	Internal control and observation become explicit design choices.
1975	Intel introduces the ICE-80 in-circuit emulator.[34]	Processor execution can be controlled and inspected within the target system.
1977	Frohwerk describes signature analysis using self-stimulating test modes and compacted responses.[128]	Digital field diagnosis becomes a design concern, with expected signatures at test points.
1979	Terrestrial cosmic-ray soft-error mechanisms are reported.[46]	Transient memory faults are connected to environment and device physics.
1979	May and Woods describe alpha-particle-induced memory errors from packaging contaminants.[115]	A soft-error symptom is connected to a packaging-related radiation mechanism.

PERIOD	DEVELOPMENT	DIAGNOSTIC SIGNIFICANCE
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.
1993–1995	The Guide to the Expression of Uncertainty in Measurement is published and corrected; JCGM issues the cited version in 2008.[113]	Uncertainty is treated as part of a measurement result and its model.
1994	Pentium FDIV defect becomes publicly reproducible.[100]	A compact external test turns a rare design fault into accountable evidence.
1995	The SFF Committee approves SMART for publication as SFF-8035i.[136]	Storage devices expose internal indicators beyond host-visible I/O failure.
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	Colvin reviews emission microscopy, SEM, FIB, and related failure-analysis methods.[131]	Electrical symptoms gain a return path to a physical site and mechanism.
1998	IPMI version 1.0 is released on 16 September.[137]	A controller outside the host operating system observes and controls the server.
1999	Biere and colleagues present SAT-based bounded model checking.[121]	Finite counterexample executions become artifacts for examining design-property violations.
1999–2003	Nexus and CoreSight-era embedded debug architectures emerge.[54, 56]	Trace and debug move on chip as external buses disappear.
2007	Large disk-fleet studies quantify replacement trends and population behavior.[51, 52]	Population evidence challenges simple vendor health scores.
Late 2000s	ACPI error interfaces and OS hardware-error architectures mature.[42, 43]	Firmware, operating systems, and applications exchange structured error records.
2009	Large-scale production DRAM errors are reported.[49]	Correctable and uncorrectable errors show clustering and operational context.
2009	PLDM platform monitoring and control specification version 1.0 is published; the cited 1.3.0 edition follows in 2024.[118]	Management records relate sensors, effecters, events, and platform entities.

PERIOD	DEVELOPMENT	DIAGNOSTIC SIGNIFICANCE
2010	Rogin and Drechsler systematize electronic-system-level debugging around SystemC and automatic failure isolation.[111]	Hardware debugging moves into executable design models before fabrication.
2010	Mitra, Seshia, and Nicolici review post-silicon validation and debug.[120]	Validation of implemented designs is distinguished from manufacturing defect screening.
2010s	Streaming telemetry and gNMI supplement polling.[74]	Typed, subscription-based device state becomes a continuous evidence stream.
2012	Glaessgen and Stargel describe a digital-twin paradigm combining simulation, vehicle-health data, and maintenance history.[132]	Diagnosis extends toward future condition and decision support.
2014	RowHammer disturbance errors are demonstrated in commodity DRAM.[50]	Access patterns interact with device physics across cell boundaries.
2015	DMTF releases Redfish 1.0 on 4 August.[134]	Hardware management moves toward schema-driven web APIs.
2018	OpenBMC joins the Linux Foundation as an open platform-management project.[133]	BMC sensors, events, and telemetry become inspectable and extensible.
2020s	Accelerator telemetry and health tooling become fleet infrastructure.[87, 88]	GPU, HBM, fabric, workload, and thermal evidence must be correlated.
2021	Google and Facebook (now 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.
2022	NVMe Base Specification 2.0c documents several diagnostic log classes.[116]	Health, error, telemetry, and persistent-event records provide different storage observations.
2023	SPDM 1.3.0 specifies authentication and measurement exchange.[119]	Component identity and reported state become evidence with explicit trust requirements.
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.
2024	CXL consortium guidance describes RAS from CXL 2.0 through 3.1.[117]	Memory-event records, address translation, and poison handling support cross-device diagnosis.

PERIOD	DEVELOPMENT	DIAGNOSTIC SIGNIFICANCE
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.
2025	UCIe 3.0 expands chiplet manageability and runtime event signaling.[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

The definitions support comparison across the chapters. A particular standard or implementation may use a narrower meaning.

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.

Aliasing in signature analysis. An erroneous test response produces the expected compacted signature.

Analysis pattern. A reusable diagnostic analysis technique defined by its context, artifacts, operations, and limitations.

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.

Attestation. Evidence supporting claims about a component's identity or measured state, evaluated against a trust policy.

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.

Bounded model checking. Search for a property violation in model executions up to a specified bound.

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

Calibration. Establishing a relation between reference values and instrument indications, with associated uncertainties; it is distinct from adjustment.[123]

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.

CPER. Common Platform Error Record, a UEFI format with an error-record header and typed sections.

CXL. Compute Express Link, an interconnect supporting coherent device and memory access with associated RAS mechanisms.

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.

Diagnosability. Ability to distinguish the fault alternatives of interest using the available observations and tests.

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.

Effecter. A PLDM abstraction through which management software can request a change in a represented entity.

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. A process or design behavior that explains how a fault produces an error 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.

Formal observability. Ability to determine internal state from input and output behavior under a specified system model.

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.

GHES. Generic Hardware Error Source, an ACPI mechanism describing access to platform-provided hardware-error status.

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

Hardware narratology. Application of narrative concepts and trace analysis patterns to hardware signals and their recorded representations, within unified hardware–software diagnostics.[124]

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]

Measurement uncertainty. Quantified dispersion associated with values attributed to a measured quantity using available information.

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

Metrological traceability. Relation of a measurement result to a reference through a documented calibration chain with stated uncertainties.

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.

NVMe. NVM Express, specifications for access to nonvolatile storage and associated management and diagnostic information.

Observability. Access to interpretable evidence about internal condition; formal state observability has a narrower model-dependent meaning.

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 any odd number of bit errors within the checked group.

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

PLDM. Platform Level Data Model, a family of management data models and commands including monitoring and control.

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.

Post-silicon validation. Investigation of an implemented design’s behavior on fabricated hardware under actual execution conditions.

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; its outcome and recurrence provide evidence without necessarily implying physical degradation.

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.

SPDM. Security Protocol and Data Model, supporting component authentication and measurement exchange.

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

We can classify a diagnostic artifact by the questions it can answer, the information it omits, and the context required for its interpretation.

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

Evidence	Strongest questions	Characteristic blind spots	Identity / trust requirements
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
Model counterexample	A property violation and the model execution that produces it	Model abstraction, assumptions, finite bounds, omitted physical behavior	Model and property versions; initial-state constraints; solver result; execution bound
NVMe diagnostic log	Controller health, errors, selected events, and captured internal telemetry	Optional support; vendor-specific payloads; finite event history	Controller and firmware identity; log type and generation; capture time; decoder
Authentication and measurement record	Component identity and reported measured state within a trust model	Compromised trust assumptions; runtime faults; sensor drift remain possible	Certificate and reference policy; freshness; component identity; validation result

Table 4. Evidence taxonomy. The recorded value or image is interpreted together with its origin, configuration, units, acquisition conditions, and relationship to the reported problem.

References

The references identify the publications and editions used in this history. They include standards, manuals, archival records, research papers, and investigation reports. Links lead to publisher records, institutional repositories, source organizations, or identified archival copies. Named specification versions identify the cited editions, not necessarily the latest revisions. 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] N. Hawkins and Staff. Hawkins Electrical Guide, Number Three: A Practical Treatise. Theo. Audel & Co., 1914. https://www.gutenberg.org/ebooks/49769

[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. “Timeline.” https://www.keysight.com/us/en/about/timeline.html

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

[10] Guy Fedorkow. “The Whirlwind Computer at CHM.” Computer History Museum, 30 November 2018. https://computerhistory.org/blog/the-whirlwind-computer-at-chm/

[11] Guy Fedorkow and David C. Brock. “Jingle Bits: Auditory Maintenance, Whirlwind Holiday Songs & the Dawn of Computer Music.” CHM Editorial, Computer History Museum, 16 December 2018. 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 (April 1950): 147–160. https://doi.org/10.1002/j.1538-7305.1950.tb00463.x Archival scan: 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.” In Automata Studies, edited by C. E. Shannon and J. McCarthy, Annals of Mathematics Studies 34, 43–98. Princeton University Press, 1956. Linked archive: January 1952 Caltech lecture notes and publication history. 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] Alan A. Grometstein, ed. MIT Lincoln Laboratory: Technology in Support of National Security. Lexington, MA: MIT Lincoln Laboratory, 2011. Chapter 2, "The SAGE Air Defense System," p. 33, Figure 2-11. 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 Contractor Report 182505, March 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 (R&M) Standard for Spaceflight and Support Systems. Approved 13 June 2017. 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 by Notice 3, 4 August 1998. https://everyspec.com/MIL-STD/MIL-STD-1600-1699/MIL_STD_1629A_1556/

[23] W. Vesely, M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick, and J. Railsback. Fault Tree Handbook with Aerospace Applications. Version 1.1. NASA Office of Safety and Mission Assurance, August 2002. Course archive: 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 (December 1991): 1710–1734. https://doi.org/10.1109/5.119549 Archival scan: 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, ninth edition, November 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, June 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] George A. Haag. "A Logic State Analyzer for Evaluating Complex State Flow." Hewlett-Packard Journal 29, no. 6 (February 1978): 2–10. 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 10, no. 1 (March 1993): 73–82. https://doi.org/10.1109/54.199807 Author manuscript: https://www.ardent-tool.com/CPU/docs/AMD/anatomy/misc/articles/agrawal.pdf

[37] IEEE Standards Association. IEEE Std 1149.1-2013, IEEE Standard for Test Access Port and Boundary-Scan Architecture. 2013 revision; the original standard was IEEE Std 1149.1-1990. https://standards.ieee.org/ieee/1149.1/4484/

[38] IBM. Personal Computer Guide to Operations, document part 6025003 (binder part 6025000), first edition, August 1981. https://ps2.infania.net/IBM_5150_Guide_to_Operations_6025000_AUG81.pdf

[39] IBM. Personal Computer Technical Reference, document part 6025008 (binder part 6025005), first edition, August 1981. https://www.minuszerodegrees.net/manuals/IBM_5150_Technical_Reference_6025005_AUG81.pdf

[40] UEFI Forum. Platform Initialization Specification 1.9, Volume 3: Shared Architectural Elements, Chapter 6: Status Codes. https://uefi.org/specs/PI/1.9/V3_Status_Codes.html

[41] UEFI Forum. Unified Extensible Firmware Interface Specification, Release 2.11, 2024. Appendix N: Common Platform Error Record. https://uefi.org/sites/default/files/resources/UEFI_Spec_Final_2.11.pdf

[42] UEFI Forum. ACPI Specification, Version 6.5 Errata A, December 2024. Chapter 18: 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 (WHEA).” Accessed 13 September 2026. 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: System Programming Guide, Part 2. Order 253669-078US, December 2022. Chapter 16: Machine-Check Architecture. 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 ’09, 2009, 193–204. https://doi.org/10.1145/1555349.1555372 Author copy: 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.” 41st International Symposium on Computer Architecture (ISCA), 2014, 361–372. https://doi.org/10.1109/ISCA.2014.6853210 Author copy: 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.” 5th USENIX Conference on File and Storage Technologies (FAST ’07), February 2007, 17–28. 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?” 5th USENIX Conference on File and Storage Technologies (FAST ’07), February 2007, 1–16. https://www.cs.toronto.edu/~bianca/papers/fast07.pdf

[53] DMTF. DSP1113, Disk Drive Diagnostics Profile, Version 1.0.0. DMTF Standard, 16 September 2011. https://www.dmtf.org/sites/default/files/standards/documents/DSP1113_1.0.0.pdf

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

[55] Arm. CoreSight Program Flow Trace Architecture Specification. ARM IHI 0035B, 31 March 2011. https://developer.arm.com/documentation/ihi0035/latest/

[56] IEEE-ISTO. The Nexus 5001 Forum Standard for a Global Embedded Processor Debug Interface. IEEE-ISTO 5001-2003, Version 2.0, 23 December 2003. https://www.arbitrarytechnology.com/files/Download/ieee-5001%20nexus%20protocol.pdf

[57] RISC-V International. RISC-V Debug Specification, Version 1.0, ratified February 2025; document dated 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_201202: E/E Diagnostic Test Modes. Revision dated 23 February 2012. https://www.sae.org/standards/j1979_201202-e-e-diagnostic-test-modes

[61] SAE International. J1979-2_202604: E/E Diagnostic Test Modes: OBDonUDS. Issued 21 April 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; Maintainability Technique DFE-7. 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, unapproved Draft 2, 2 April 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 E. Sharp, and Jeffrey W. Herrmann. “Comprehensive Evaluations of Condition Monitoring-Based Technologies in Industrial Maintenance: A Systematic Review.” Journal of Manufacturing Systems 82 (October 2025): 449–477. 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): 407–438. https://doi.org/10.6028/jres.087.024

[70] Kenneth A. Snyder, Li-Piin Sung, and Geraldine S. Cheok. Nondestructive Testing (NDT) and Sensor Technology for Service Life Modeling of New and Existing Concrete Structures, NIST IR 7974, December 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. Accessed 13 September 2026. https://openconfig.net/docs/gnmi/gnmi-specification/

[75] DMTF. “Intelligent Platform Management Interface: Joint Message from the IPMI Promoters.” June 2020. https://www.dmtf.org/standards/ipmi

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

[77] Piotr Matuszczak, with Pawel Rapkiewicz and Kamil Kowalski. “OpenBMC platform telemetry.” OpenBMC design document, created 7 August 2019. Accessed 13 September 2026. 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. Accessed 13 September 2026. 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. Accessed 13 September 2026. https://c3.ndc.nasa.gov/dashlink/resources/139/

[83] Allan L. White. “Designing Fault-Injection Experiments for the Reliability of Embedded Systems.” 31st IEEE/AIAA Digital Avionics Systems Conference (DASC), October 2012. NASA NTRS 20120016069. 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. NVIDIA DCGM Documentation: User Guide. Accessed 13 September 2026. 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] Harish Dattatraya Dixit and Sriram Sankar. “How Meta Keeps Its AI Hardware Reliable.” Engineering at Meta, 22 July 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 2.0 Specification” (6 August 2024) and “UCIe 3.0 Specification” (5 August 2025). Specification summaries: https://www.uciexpress.org/specifications Release announcements: https://www.uciexpress.org/press-releases

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

[93] NASA. Report of Apollo 13 Review Board. E. M. Cortright, chair. 15 June 1970. https://ntrs.nasa.gov/api/citations/20170000913/downloads/20170000913.pdf

[94] Presidential Commission on the Space Shuttle Challenger Accident. Report, Volume I, 6 June 1986. Chapter IV, "The Cause of the Accident": https://www.nasa.gov/history/rogersrep/v1ch4.htm Chapter V, "The Contributing Cause of the Accident": https://www.nasa.gov/history/rogersrep/v1ch5.htm

[95] National Transportation Safety Board. Aloha Airlines, Flight 243, Boeing 737-200, N73711, near Maui, Hawaii, April 28, 1988. Aircraft Accident Report NTSB/AAR-89/03, adopted 14 June 1989. 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, 9 September 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] Bureau of Ocean Energy Management, Regulation and Enforcement. Report Regarding the Causes of the April 20, 2010 Macondo Well Blowout. Deepwater Horizon Joint Investigation Team Final Report, Volume II, 14 September 2011. https://www.bsee.gov/sites/bsee.gov/files/2024-12/DWH%20DOI%20Vol%20II%20Final.pdf Release page: 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." Facebook, Inc., February 2021. arXiv:2102.11245. 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, 16 April 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

[112] R. E. Kalman. “On the General Theory of Control Systems.” IFAC Proceedings Volumes 1, no. 1 (1960): 491–502. https://doi.org/10.1016/S1474-6670(17)70094-8

[113] Joint Committee for Guides in Metrology. Evaluation of Measurement Data—Guide to the Expression of Uncertainty in Measurement. JCGM 100:2008; GUM 1995 with minor corrections. https://doi.org/10.59161/JCGM100-2008E

[114] M. J. Y. Williams and J. B. Angell. “Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic.” IEEE Transactions on Computers C-22, no. 1 (1973): 46–60. https://doi.org/10.1109/T-C.1973.223600

[115] T. C. May and M. H. Woods. “Alpha-Particle-Induced Soft Errors in Dynamic Memories.” IEEE Transactions on Electron Devices 26, no. 1 (1979): 2–9. https://doi.org/10.1109/T-ED.1979.19370

[116] NVM Express. NVM Express Base Specification, Revision 2.0c. 4 October 2022. Sections 5.16.1.2, 5.16.1.3, 5.16.1.7–5.16.1.9, and 5.16.1.14. https://nvmexpress.org/wp-content/uploads/NVM-Express-Base-Specification-2.0c-2022.10.04-Ratified-1.pdf

[117] Antonio Hasbun, Jordan Chin, Daniele Balluchi, Thomas Won Ha Choi, Tony Benavides, and Sanjay Goyal. An Overview of RAS for Compute Express Link: Covering from CXL 2.0 to CXL 3.1. CXL Consortium white paper, 27 March 2024. Informative guidance; the specification is normative. https://computeexpresslink.org/wp-content/uploads/2024/08/An-Overview-of-RAS-for-Compute-Express-Link-3.1-Whitepaper.pdf

[118] DMTF. Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification. DSP0248, version 1.3.0, document dated 20 June 2024. https://www.dmtf.org/sites/default/files/standards/documents/DSP0248_1.3.0.pdf

[119] DMTF. Security Protocol and Data Model (SPDM) Specification. DSP0274, version 1.3.0, 28 June 2023. https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.3.0.pdf

[120] Subhasish Mitra, Sanjit A. Seshia, and Nicola Nicolici. “Post-Silicon Validation Opportunities, Challenges and Recent Advances.” Proceedings of the 47th Design Automation Conference (DAC), 2010, 12–17. https://doi.org/10.1145/1837274.1837280

[121] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. “Symbolic Model Checking without BDDs.” In Tools and Algorithms for the Construction and Analysis of Systems, TACAS 1999, LNCS 1579, edited by W. Rance Cleaveland, 193–207. Springer, 1999. https://doi.org/10.1007/3-540-49059-0_14

[122] Dmitry Vostokov. Theoretical Software Diagnostics: Collected Articles. Fourth edition. OpenTask, February 2024. “Principles of Pattern-Oriented Software Data Analysis,” “Defect Mechanism Patterns (DMP),” and “Unified Computer Diagnostics: Incorporating Hardware Narratology.” ISBN 978-1-912636-91-4. https://www.patterndiagnostics.com/theoretical-software-diagnostics-book

[123] Joint Committee for Guides in Metrology. International Vocabulary of Metrology—Basic and General Concepts and Associated Terms (VIM). JCGM 200:2012, third edition, 2008 version with minor corrections. Sections 2.39 and 2.41. https://doi.org/10.59161/JCGM200-2012

[124] Dmitry Vostokov. “Unified Computer Diagnostics: Incorporating Hardware Narratology.” Software Diagnostics and Observability Institute, n.d. Accessed 13 September 2026. https://dumpanalysis.org/unified-computer-diagnostics-hardware-narratology

[125] Tektronix. ABCs of Probes Primer. Sections on probe loading, capacitance, and grounding. Accessed 13 September 2026. https://www.tek.com/en/documents/whitepaper/abcs-probes-primer

[126] Gerald E. Nelson and David W. Ricci. “A Practical Interface System for Electronic Instruments.” Hewlett-Packard Journal 24, no. 2 (October 1972): 2–7. https://hparchive.com/Journals/HPJ-1972-10.pdf

[127] Vishwani D. Agrawal, Charles R. Kime, and Kewal K. Saluja. “A Tutorial on Built-In Self-Test, Part 2: Applications.” IEEE Design & Test of Computers 10, no. 2 (June 1993): 69–77. https://doi.org/10.1109/54.211530

[128] Robert A. Frohwerk. “Signature Analysis: A New Digital Field Service Method.” Hewlett-Packard Journal 28, no. 9 (May 1977): 2–8. https://hparchive.com/Journals/HPJ-1977-05.pdf

[129] J. W. McPherson. Reliability Physics and Engineering: Time-To-Failure Modeling. Springer, 2010. Chapters 11–12, failure mechanisms in integrated circuits and mechanical engineering. https://doi.org/10.1007/978-1-4419-6348-2

[130] Mark White. Scaled CMOS Technology Reliability Users Guide. JPL Publication 09-33, January 2010. NASA Electronic Parts and Packaging Program. https://nepp.nasa.gov/files/20273/09_102_5_JPL_White_Scaled%20CMOS%20Technology%20Reliability%20Users%20Guide%201_10.pdf

[131] Jim Colvin. “ESD Failure Analysis Methodology.” Microelectronics Reliability 38, no. 11 (November 1998): 1705–1714. https://doi.org/10.1016/S0026-2714(98)00173-5

[132] Edward H. Glaessgen and David S. Stargel. “The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles.” 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, AIAA 2012-1818, 2012. https://doi.org/10.2514/6.2012-1818 NASA copy: https://ntrs.nasa.gov/citations/20120008178

[133] Jim Zemlin. “OpenBMC Project Community Comes Together at The Linux Foundation to Define Open Source Implementation of BMC Firmware Stack.” Linux Foundation, 19 March 2018. https://www.linuxfoundation.org/blog/blog/openbmc-project-community-comes-together-at-the-linux-foundation-to-define-open-source-implementation-of-bmc-firmware-stack

[134] DMTF. “DMTF Helps Enable Multi-Vendor Data Center Management with New Redfish 1.0 Standard.” 4 August 2015. https://www.dmtf.org/news/pr/2015/8/dmtf-helps-enable-multi-vendor-data-center-management-new-redfish-10-standard

[135] U.S. Nuclear Regulatory Commission. “Backgrounder on the Three Mile Island Accident.” Summary of Events. Accessed 13 September 2026. https://www.nrc.gov/reading-rm/doc-collections/fact-sheets/3mile-isle

[136] Dal Allan. “Liaison Report from SFF.” X3T10/95-292r0, 1995. Records approval of SMART for publication as SFF-8035i. https://www.t10.org/ftp/t10/document.95/95-292r0.pdf

[137] Intel, Hewlett-Packard, NEC, and Dell. Intelligent Platform Management Interface Specification, Second Generation v2.0. Document Revision 1.1, 1 October 2013. Revision history records the initial v1.0 release on 16 September 1998. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ipmi-second-gen-interface-spec-v2-rev1-1.pdf

[138] Jerzy J. Dabrowski and Rashad M. Ramzan. “Built-in Loopback Test for IC RF Transceivers.” IEEE Transactions on Very Large Scale Integration (VLSI) Systems 18, no. 6 (June 2010): 933–946. https://doi.org/10.1109/TVLSI.2009.2019085

[139] S. Waldbusser. RFC 1271: Remote Network Monitoring Management Information Base. November 1991. https://www.rfc-editor.org/rfc/rfc1271.html

[140] Alexander Weiler, Alexander Pakosta, and Ankur Verma. High-Speed Layout Guidelines. Texas Instruments Application Report SCAA082A, November 2006, revised August 2017. https://www.ti.com/lit/an/scaa082a/scaa082a.pdf

[141] Tektronix. “Global technology company Tektronix marks 75 years of innovation.” 23 June 2021. Founding date and Tektronix Timeline entry for 1947. https://www.tek.com/en/news/2021-06-23-global-technology-company-tektronix-marks-75-years-of-innovation

[142] Macronix International. Program/Erase Cycling Endurance and Data Retention of Macronix SLC NAND Flash Memories. Technical Note AN0339, Revision 1, 15 October 2014, pp. 1–2.

https://www.mxic.com.tw/Lists/ApplicationNote/Attachments/1920/AN0339V1-Endurance%20and%20Retention%20of%20NAND%20Flash.pdf

[143] Tektronix. Understanding and Characterizing Timing Jitter. Primer 55W-16146-5, September 2012. Sections 1–2, pp. 3–5. https://download.tek.com/document/55W_16146_5_MR_Letter.pdf

[144] U.S. Coast Guard. Report of Investigation into the Circumstances Surrounding the Explosion, Fire, Sinking and Loss of Eleven Crew Members Aboard the Mobile Offshore Drilling Unit Deepwater Horizon in the Gulf of Mexico, April 20–22, 2010. Volume I, MISLE Activity Number 3721503, released 22 April 2011. Chapter 1, subsection “Fire and Gas Detection System Logic,” p. 20. Official report index: https://www.dco.uscg.mil/OCSNCOE/Accidents-Investigations/DWH-Macondo/