DOI: 10.21791/IJEMS.2026.13

Research Paper

Hardware-Native Engineering Management: Reclaiming Manufacturing Discipline from Software-Derived Practice

BABU GEORGE[1], DIVYA CHOUDHARY[2]

[1]School of Business, Alcorn State University, Mississippi-39056, USA.
bgeorge@alcorn.edu (corresponding author)
[2]Bailey College of Engineering & Technology, Indiana State University, Terre Haute, Indiana - 47809, USA.
Divya.Choudhary@indstate.edu

Received: 19 April 2026 **Accepted:** 21 August 2026 **Online Publication Date:** 3 Sept 2026

How to cite

B. George and D. Choudhary, "Hardware-Native Engineering Management: Reclaiming Manufacturing Discipline from Software-Derived Practice", *International Journal of Engineering and Management Sciences*, pp. 1–27, Sept 2026. doi: 10.21791/IJEMS.2026.13

Abstract. Recent management doctrine is often associated with software-sector practices such as agile methods, sprint cycles, and minimum viable products. That account obscures an earlier manufacturing lineage. Many of the practices software popularized were first abstracted from hardware manufacturing. Scrum grew out of Takeuchi and Nonaka's 1986 study of Honda, Canon, Fuji-Xerox, Toyota, and other manufacturers; kanban and just-in-time came from Toyota's production system. Software borrowed these methods and adapted them to a medium in which rollback is cheap, feedback is quick, and most failures carry limited cost. During that adaptation, software practice reduced the emphasis on front-loaded discipline that the manufacturing originals retained for physical reasons. The lighter versions were later applied in hardware contexts as general management practice, often without sufficient attention to the constraints that physical work imposes. We argue that managing hardware with these lighter methods produces a predictable class of failure, and that two forces now make correction urgent: renewed investment in physical systems (energy, semiconductors, defence, robotics, and the physical plant of AI itself), while generative AI automates much of the software work whose practices were treated as universal. We identify the physical properties that separate the two domains, compare the resulting framework with established industrial standards, and propose five principles for hardware-native engineering management: simulation-first design, risk as a first-class metric, decision quality over speed, structured management of irreversibility, and dual-speed organizational architecture. We operationalize each principle with metrics and a maturity model, then test the framework's diagnostic value through a structured retrospective analysis of three documented failures and one contrasting success.

Keywords: High-Reliability Organizations, Simulation-First Management, Systems Engineering, Resilience Engineering, Set-Based Concurrent Engineering, Institutional Logics, AI and the Future of Engineering Work.

DOI: 10.21791/IJEMS.2026.13

Introduction

Contemporary management doctrine is often traced to software. Agile methods, objectives and key results, sprint cycles, and "move fast and break things" spread from Silicon Valley into engineering management across sectors over the past two decades [1], [2], [3]. Oil refineries adopted sprint reviews. Pharmaceutical firms hired agile coaches. Construction companies ran daily stand-ups. The lineage is less straightforward than that account allows, and the correction is relevant for the argument that follows. The methods that software made famous did not begin with software. Scrum traces to a 1986 study by Takeuchi and Nonaka of how Honda, Canon, Fuji-Xerox, Toyota, and other manufacturers developed physical products through overlapping phases and self-organizing teams [4]; Jeff Sutherland and Ken Schwaber adapted that manufacturing research into a software method in the early 1990s. Kanban and just-in-time came from Toyota's production line [5], [6]. Lean software development was an explicit translation of Toyota's manufacturing principles into code [7]. These practices therefore originated in manufacturing before being adapted to software development.

Software adapted those methods to a medium with different physical and economic properties, and the adaptation changed their meaning. Software retained practices suited to cheap, reversible, fast-feedback work: short iterations, deferred specification, and learning through deployment. It reduced the emphasis on front-loaded discipline that manufacturing had retained for physical reasons. These lighter versions were then applied in hardware industries as contemporary best practice, often without equal attention to the constraints that physical work imposes. One question received limited attention during this diffusion: whether the properties of the work matched the properties that made the lighter methods effective in software [8]–[11]. In many hardware contexts, they did not, and that mismatch now contributes to recurring organizational problems as investment returns to hardware-intensive systems at scale.

A second development compounds the problem. Generative AI systems are automating substantial portions of the software engineering workflow itself [12]. Handa et al. [13], analyzing millions of conversations with AI assistants, found that software development was one of the largest single categories of economic tasks performed with AI, with coding, debugging, and documentation accounting for a disproportionate share of usage. Sergeyuk et al. [14] documented that AI-based coding assistants moved from novelty to standard practice within roughly three years, changing the skills required by software teams. Khojah et al. [15] observed that professional software engineers now use AI tools across the full range of engineering activities, including architecture decisions, code review, and test design. The profession whose working methods were universalized into a management philosophy is itself being remade by automation.

Two forces converge here: renewed investment in hardware and the automation of parts of software work. For two decades, software professionals occupied a privileged position in the engineering labour market, and their practices strongly influenced engineering management more broadly. That position is changing. Several competencies likely to remain difficult to automate in the coming decade lie in the physical domain: power systems, semiconductor fabs, robotics platforms, and energy infrastructure. The forces driving renewed hardware demand are structural rather than cyclical. The energy transition requires solar panels, wind turbines, grid-scale batteries, and power electronics at a pace without

DOI: 10.21791/IJEMS.2026.13

modern precedent. The 2020–2023 chip shortage prompted new fabrication capacity to address the vulnerabilities it exposed. Defence modernization depends on complex physical systems, from hypersonic vehicles to satellite constellations. The AI systems automating software work also depend on physical infrastructure: advanced chips, data centres, and new power generation at a scale that places AI's supply chain within the domain of physical systems [16].

Engineering management as a discipline has long addressed the relationship between technical complexity and organizational practice [10], [11]. Lannes [17] defined the field as the integration of engineering knowledge with business administration, but the body of knowledge tilted heavily toward software-derived practice over the past two decades [18]. Earlier scholars recognized that engineering management requires competencies tied to the physical realities of each domain [9], [19]. This paper extends that recognition into a framework for conditions in which AI is reducing the distinctiveness of some software tasks while physical systems impose constraints that software-derived management often treats too lightly.

We make four contributions. First, we correct the historical account: dominant methods were abstracted from hardware manufacturing, lightened for software's permissive physics, and then re-exported to hardware in a reduced form. Hardware management therefore benefits from revisiting the disciplined manufacturing practices that software adapted. Second, we ground the hardware-software distinction in physical properties rather than cultural preference, and we show why the AI-driven compression of software labour makes that correction urgent. Third, we set the resulting five-principle framework against established industrial standards, including Stage-Gate, systems engineering, lean and set-based product development, Six Sigma, functional-safety standards, and critical-chain methods, making explicit what the framework recombines and what it adds. Fourth, we operationalize the framework with metrics, a maturity model, and a diagnostic, and we examine its explanatory value against documented cases.

1. Historical Lineage and Institutional Diffusion of Software-Derived Management

The Agile Manifesto was written by seventeen software developers at Snowbird, Utah, in 2001 [3]. Its subtitle, "Uncovering Better Ways of Developing Software," signals an explicit awareness of its scope. The authors responded to recognized limitations in waterfall project management, including large specifications, long development cycles, and products that could become misaligned with user needs before release. Their iterative, team-based, customer-collaborative approach worked well in that setting. The relevant historical lineage, however, extends through manufacturing.

The genealogy supports this point. The term "scrum" entered software through Takeuchi and Nonaka's 1986 account of new product development at hardware manufacturers, where overlapping phases and self-organizing teams outperformed the old sequential handoff [4]. Kanban was a signalling system Taiichi Ohno built for the Toyota production line to pull inventory just in time [5], and Womack, Jones, and Roos named the whole approach "lean" after studying automobile plants [6]. When Poppendieck and Poppendieck wrote the bridge text for software, they presented it openly as a translation of Toyota

DOI: 10.21791/IJEMS.2026.13

manufacturing into code [7]. The historical sequence therefore runs from manufacturing practice to software adaptation. Figure 1 lays out the path.

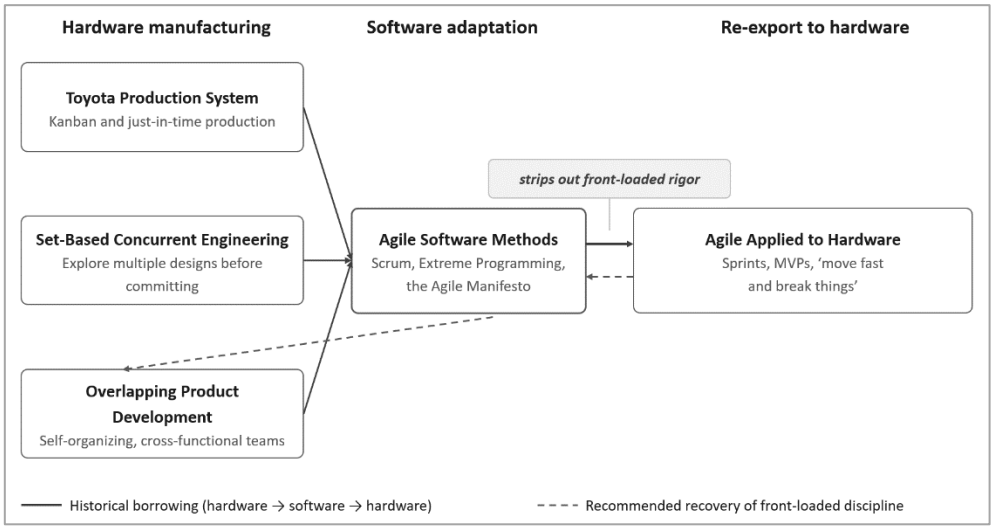

Figure 1. Genealogical development of dominant engineering management methods from manufacturing origins through software adaptation and subsequent reapplication in hardware contexts. Source: prepared by the authors.

The adaptation shown in the middle band of Figure 1 was not a faithful copy. Software kept what fit a cheap, reversible, fast-feedback medium and left aside what did not. The clearest example is set-based concurrent engineering, the practice Ward, Liker, Cristiano, and Sobek documented at Toyota [20]. Toyota carries several design alternatives in parallel and delays commitment until uncertainty has been reduced, precisely because tooling, sheet metal, and supplier contracts cannot be undone once chosen. This approach differs from releasing a minimum viable product and iterating in production. Software did not need set-based discipline, because a software team can commit early and reverse course cheaply. The software adaptation therefore discarded much of the front-loaded analysis, parallel option-carrying, interface control, and tolerance control that manufacturing had retained for physical reasons. This reduction was reasonable within many software contexts, but became problematic when the lighter method was reapplied to hardware.

Within a decade, the lightened method had migrated well beyond software. By 2015, agile coaches were operating in oil refineries, pharmaceutical labs, and construction sites, and consultancies were selling agile transformations across multiple sectors. However, equally strong evidence supporting the effectiveness of these practices outside of software was lacking. Shenhar and Dvir [21] observed that project management research had already moved toward a one-size-fits-all approach, treating all projects as variants of a single type, and the agile movement accelerated that convergence. The terminology also transmitted assumptions about reversibility, feedback speed, and acceptable failure costs. A manager who adopted "sprint" often accepted the premise that two weeks is a meaningful unit of engineering progress; a programme manager who used "MVP" could imply that releasing an incomplete product is a valid learning strategy. In many hardware contexts, those premises require qualification.

Institutional theory explains how a poorly fitting practice spreads and persists. Greenwood et al. [22] showed that organizations in complex institutional fields face pressure to adopt practices that carry legitimacy even when those practices conflict with operational demands. The diffusion of agile into

DOI: 10.21791/IJEMS.2026.13

hardware ran through all three isomorphic channels: coercive pressure from consultancies and investors who treated agile adoption as a marker of modernity, mimetic imitation of high-status technology firms whose valuations dwarfed traditional manufacturers, and normative transmission through MBA programmes and professional networks that treated software management as the default curriculum [23]. Chandler and Hwang [24] modelled how organizations shift from mindful evaluation of fit to unreflective copying once a practice has reached sufficient diffusion. By the early 2010s, organizations widely regarded agile adoption as evidence of managerial modernity.

The result was institutional decoupling. Bromley and Powell [25] documented how organizations adopt practices for their symbolic value while core work continues under different logics. Hardware firms that adopted agile ceremonies often retained separate decision processes rooted in older engineering disciplines, running sprint reviews for reporting while schedule commitments continued to depend on gate reviews. Seo and Creed [26] theorized that such contradictions generate friction that can eventually destabilize the adopted practice. Some dysfunctions visible in hardware-intensive industries today, including schedule overruns, design rework after premature commitment, and weak escalation of engineering concerns, can be read as examples of this friction. Gosain [27] showed that information systems themselves carry institutional logic; project management software built around sprint velocity and backlog grooming encodes software assumptions into the tools through which hardware teams plan and report, making the imported logic difficult to revise even when managers recognize the mismatch.

This account reframes the problem. The universalization of software management was not a collective lapse in judgment by individual managers. It was a field-level process driven by institutional pressures that organizational theory has studied at length [28], [29]. Purdy and Gray [30] showed that when conflicting logics operate in the same field, the logic backed by more powerful actors and more established channels tends to win regardless of technical fit. Software-sector management logic, supported by venture capital, elite universities, and highly capitalized technology firms, gained influence over hardware-sector management logic through these channels. Recognizing the mechanism changes the prescription. Correcting the mismatch requires more than replacing one set of ceremonies with another. This process involves challenging the established logic related to professional identity, the educational pipeline, and the reward structure [31], as well as restoring the original disciplined manufacturing practices that software has borrowed and simplified.

Three older engineering management traditions became less visible as the lightened method spread. Systems engineering, developed in aerospace and defence for products that cannot be patched after deployment, emphasized rigorous requirements, interface control, and verification before production [32]. Reliability engineering treated failure as a statistical certainty and designed systems to degrade gracefully [33]. Institutional knowledge management recognized that deep domain expertise takes decades to build and is difficult to replace once lost. Organizations that reduced their reliance on these traditions are now returning to them in response to hardware-intensive execution problems.

2. Domain Properties of Software and Hardware Engineering

Every management system rests on a theory of the work, even when that theory is unstated. Agile rests on a theory about information: requirements are uncertain, building produces learning, and changing

DOI: 10.21791/IJEMS.2026.13

direction should be relatively cheap. Those claims are often true in software and only partly true in hardware. They are not merely matters of preference. A team does not simply select its management regime from a neutral menu; the physical and economic properties of the work constrain the feasible choices. Where commitment is reversible and feedback is quick, iteration-first can be rational. Where commitment is difficult to reverse and feedback arrives late, iteration-first can increase risk. Five physical properties separate the two domains, and each carries a management consequence.

The first is reversibility. A software deployment rolls back in seconds; a factory that pours a defective foundation cannot undo it. A chip fabrication run that yields defective wafers costs raw materials, machine time, clean-room hours, and twelve to sixteen weeks of calendar time. A power transformer that fails during commissioning costs roughly three to ten million dollars in equipment plus an eighteen-month replacement lead time, and if it serves a critical substation it compromises regional grid reliability for the duration. In many software settings, the recoverability of errors supports an emphasis on speed. In hardware, higher error costs give accuracy and prevention greater priority.

The second is feedback latency. Mature software teams deploy hundreds of times a day and see performance, behaviour, and error rates immediately. A hardware team may complete one full build-test cycle per quarter. A team designing a new inverter topology might spend eight to twelve weeks on simulation, four on layout and fabrication, two on assembly, then run a test programme for months; a single iteration can consume six months. In fast-feedback settings, quick learning allows for decision-making based on incomplete information. In slow-feedback settings, the penalty for finding a mistake late is a wasted year, not a wasted afternoon.

The third is failure severity. A crashed web application inconveniences users; a malfunctioning power inverter can start a fire; a defective pharmaceutical formulation can harm thousands. Perrow [34] argued in *Normal Accidents* that complex, tightly coupled physical systems produce failure modes not predictable from individual components, a dynamic with no clean analogue in well-architected software where modularity limits cascades. Bakhshi et al. [35] extended their analysis, showing that project complexity in physical systems is multidimensional, involving technological interdependence, organizational coupling, and environmental uncertainty.

The fourth is supply-chain materiality. Software supply chains are digital, governed by dependency files and package registries. Hardware supply chains involve mining, refining, alloying, machining, testing, shipping, and warehousing across geopolitical boundaries. Lead times for specialty power transformers run twelve to twenty-four months; for advanced semiconductor equipment, two to three years. Bayraktar et al. [36] traced how operations management evolved alongside physical supply-chain complexity, and their analysis shows that supply-chain management in hardware is a strategic discipline requiring long-range, relationship-intensive planning that is less central in many software contexts. The Industry 4.0 literature [37] shows that digitizing manufacturing did not remove these constraints; it added a layer of software complexity on top of them.

The fifth, newly salient, is exposure to AI automation. Code is represented as text, and current AI systems are effective at generating, transforming, and evaluating text-based artifacts. That form less readily represents physical intuition, material knowledge, and hands-on fabrication. Physical tasks such as concrete placement, soldering, and transformer commissioning remain materially embodied and only partially reducible to text-based automation. The same resistance to abstraction that limits the transfer

DOI: 10.21791/IJEMS.2026.13

of software management practices to physical work also limits the automation of some hardware competencies.

Table 1 sets the five dimensions side by side with their management implications. Together they constitute the physical basis of management fit. A material's physical properties determine which tools and processes suit it; an engineering domain's physical properties similarly shape which management practices are appropriate. The recurring problem in hardware-intensive industries is the transfer of methods across domain boundaries without adequate verification of the assumptions on which those methods depend.

Dimension	Software engineering	Hardware engineering	Management implication
Reversibility	High. Rollback in seconds; code is rewritable	Low. Poured concrete, fabricated wafers, manufactured transformers cannot be undone	Staged commitment and pre-commitment analysis, not trial and error
Feedback latency	Minutes to hours; deploy many times per day	Months to years; one build-test cycle per quarter	Simulation-first; decisions cannot wait for empirical correction
Failure severity	Roughly linear; most failures recoverable	Convex; failures cascade nonlinearly and can destroy equipment or endanger lives	Optimize for error prevention; risk is a first-class metric
Supply-chain materiality	Digital; instant global distribution	Physical; 12–36 month lead times for critical components	Long-range, relationship-intensive supply planning
AI automation exposure	High; code generation, testing, debugging increasingly automated	Low; physical intuition and fabrication resist codification	Hardware competence becomes relatively more valuable

Table 1. Comparative physical properties of software and hardware engineering domains. Source: prepared by the authors.

Figure 2 plots reversibility against feedback latency. The two arrows mark converging forces: increasing AI assistance in high-reversibility, fast-feedback software work, and renewed demand for low-reversibility, slow-feedback physical systems where hardware-native management applies.

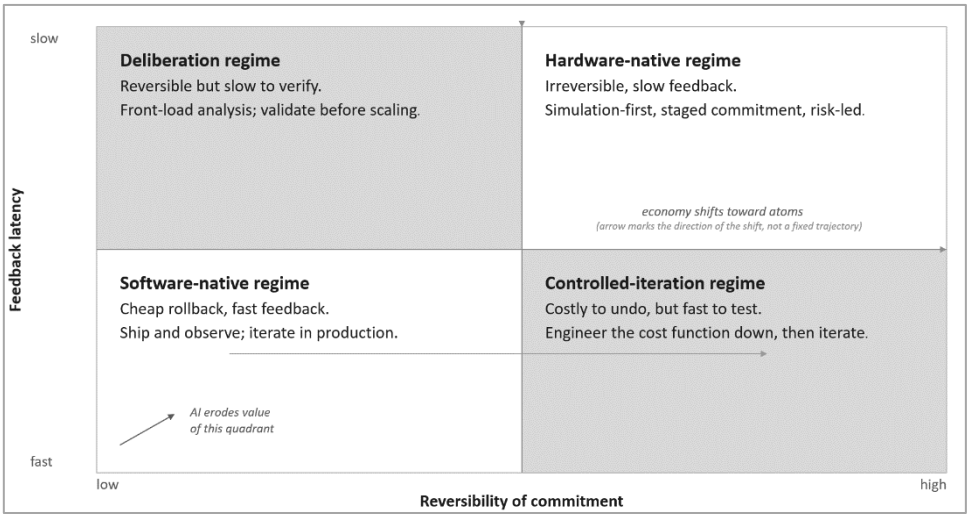

Figure 2. Management regime matrix defined by reversibility and feedback latency. Source: prepared by the authors.

3. Generative AI and the Changing Economics of Software Work

The case for hardware-native management does not depend on changes in software engineering, but those changes strengthen its relevance. Generative AI is automating several cognitive tasks that have been central to the profession, and the evidence is accumulating. Handa et al. [13] analyzed millions of

DOI: 10.21791/IJEMS.2026.13

real-world conversations with an AI assistant and found that software tasks constituted a large share of economically meaningful AI use; coding, debugging, refactoring, testing, and documentation together represented a substantial portion of professional tasks delegated to AI. This was observed behaviour at scale rather than surveyed intention. Pothukuchi et al. [38] traced the use of AI across the software lifecycle and found that it compresses timelines for requirements, design, coding, testing, and maintenance, affecting each phase that agile was designed to manage.

Sergeyuk et al. [14] found that adoption of AI coding assistants moved from experimentation to daily reliance within about three years, with engineers redirecting effort toward architectural and integration work. Khojah et al. [15] observed engineers using these tools beyond code generation, including architectural reasoning, debugging, documentation, and informal review. Vimpari et al. [39], studying game-industry artists facing image generation, documented practitioners' perception that core skills were becoming automatable within months. Software engineers face a related trajectory: tasks such as producing clean code, identifying edge cases, writing test suites, and producing readable documentation are among those for which large language models are increasingly used. Buttazzo [16] argued that work built on pattern-recognizable cognitive labour is most exposed, a category that includes much of software and less of hardware engineering.

The same properties that made the lighter methods effective in software, reversibility, fast feedback, and a low cost of change, also make many software tasks suitable for AI assistance. Hardware is less exposed on both dimensions. This creates a change in the engineering labour market: software skills and software practices have commanded wage and status premiums for two decades, and both may weaken as automation expands. Competencies tied to irreversible physical commitment, long feedback loops, and embodied domain knowledge are likely to remain harder to automate. Managing organizations that produce physical goods according to principles suited to physical work becomes a more central capability.

The implications for the workforce extend beyond individual careers. Organizations that built their management pipelines around software-era competencies now face a double adjustment: retrain managers for hardware-appropriate practice while rethinking which human competencies remain valuable as AI absorbs software-centric tasks. Barrett [40] identified systems thinking, risk cognizance, and long-horizon accountability as competencies current programmes underweight, and these are exactly what hardware management demands and AI cannot readily substitute. Cristofaro and Giardino [41] traced recurring waves of AI enthusiasm and retrenchment in management studies, each reaching further into the decision core; the current wave differs because it automates production tasks, not just analysis. Kohli and Melville [42] noted that the most disruptive technologies change the underlying logic of value creation. As AI takes on more software production tasks, management models that treated software work as the paradigmatic form of knowledge work require reconsideration.

4. Failure Cost Structures and Risk Governance in Hardware-Intensive Systems

Every management system encodes assumptions about the cost of failure. Software management developed in settings where this cost function is often comparatively flat, low, and recoverable. A small

DOI: 10.21791/IJEMS.2026.13

bug causes a small inconvenience; a larger bug usually causes a larger but still repairable problem. The curve is often roughly linear, which supports a management philosophy that treats many errors as acceptable costs of learning. In hardware, and especially in power systems, aerospace, and chemical processing, the curve can be convex: small mistakes may remain inexpensive, medium mistakes can impose large costs, and certain mistakes can cross a threshold into cascading failure.

When the cost function has this shape, the logic of management changes. Optimizing for throughput while accepting errors risks the steep part of the curve, where a single error can negate all value created. The organization must instead optimize for error prevention, even at the expense of throughput. In convex-cost domains, a managerial bias toward rapid action can increase exposure to high-consequence failure. Haldar [33] formalized this concept for structural engineering, arguing that safety under uncertainty requires probabilistic reasoning fundamentally different from the deterministic pass/fail logic common in software quality assurance.

A caveat prevents this argument from becoming a general preference for caution. Controlled failure can provide valuable information in hardware programmes when it occurs before irreversible commitment. The relevant distinction is between recoverable, bounded failure before commitment and irreversible, in-service failure after it. A prototype that fails on a test stand is a controlled experiment; a transformer that fails in a live substation is a high-consequence event. Mature hardware organizations deliberately seek controlled experiments to prevent in-service failures by testing components to destruction, conducting accelerated-life trials, and intentionally breaking things when it is cost-effective to do so. The principle is to place failure where it is recoverable and keep it out of unrecoverable settings. We return to this distinction in Section 10, where a programme that reduced its local cost function (SpaceX) illustrates the same logic from another direction.

The temporal dimension compounds the problem. A manager who cuts cost by substituting a cheaper component captures savings on the current balance sheet, while the failure the original part would have prevented may surface five or ten years later under different management. The incentive structure rewards the cost-cutter and is blind to the future consequence. This creates an asymmetric accountability gap: the costs and benefits of hardware decisions distribute across horizons far longer than any career or planning cycle.

The megaproject literature has documented the pattern with unusual precision. Flyvbjerg [43] analyzed cost and schedule data across hundreds of large infrastructure projects and summarized an “iron law”: over budget, over time, over and over again. The pattern is not random variance; it reflects a structural tendency to underestimate complexity, suppress bad news, and commit past the point where reversal remains feasible. Winch [44], studying the Channel Tunnel, traced how escalation runs through sunk-cost commitment, political lock-in, and the progressive narrowing of options as irreversible commitments accumulate. Söderlund et al. [45] argued that the field’s central lesson, that physical scale amplifies the consequences of organizational error in ways software-scale projects never experience, has been learned and forgotten repeatedly. These cases collectively indicate that hardware-intensive projects often exhibit recurring failure patterns that software-derived management assumptions inadequately address.

In many software organizations, risk management means a register reviewed at periodic meetings. When most risks are recoverable, that may be sufficient. In hardware, a realized risk can consume the

DOI: 10.21791/IJEMS.2026.13

budget, delay the schedule by years, or cause injury or death. The primary management function in hardware is therefore not simply maximizing output; it is maintaining the organization's capacity to prevent, absorb, and recover from failures.

5. High-Reliability Organizations and Resilience Engineering

The management science literature offers an established body of theory directly relevant to complex physical systems: research on high-reliability organizations. Weick and Sutcliffe [46] identified five traits of organizations that manage risk well in dangerous, tightly coupled settings such as nuclear plants, aircraft carriers, and air traffic control. The first is preoccupation with failure: they look for what might go wrong rather than celebrating what goes right. The second is reluctance to simplify: they resist dismissing anomalies, knowing that small deviations often precede large failures. The third is sensitivity to operations: they focus on the front line, not just what the hierarchy reports. The fourth is commitment to resilience: they invest in the capacity to respond and recover when prevention fails. The fifth is deference to expertise: they let the person with the most relevant knowledge decide, regardless of rank [46].

A complementary tradition, resilience engineering, extends and partly challenges this picture. Pariès et al. [47] found that while HRO theory concentrates on preventing known failure modes through vigilance, resilience engineering attends to how organizations hold acceptable performance under surprise and profound uncertainty. The distinction is relevant for hardware. Failure mode and effects analysis works well against anticipated failures, but complex physical systems also fail through novel interactions between components each operating within specification. Sujan et al. [48] highlighted Hollnagel's Safety-II: study what goes right under difficult conditions and strengthen those adaptive capacities, not only what goes wrong. Luther et al. [49] concluded that the most effective approaches combine both traditions, the HRO emphasis on vigilance and structured dissent with the resilience emphasis on adaptive capacity. Salmon et al. [50] provided a practical toolkit for analyzing how work is performed in practice rather than how it is prescribed, which maps directly onto the decoupling problem in Section 2.

These traits describe a culture that differs sharply from lighter software-derived practice. Software organizations often value speed, simplification, and flat hierarchy. HRO culture prioritizes caution in critical situations, detailed understanding when facing unusual cases, and authority based on expertise rather than job title. These differences can be understood as adaptations to different physical and operational conditions. Kotnour and Farr [10] anticipated the tension, noting that engineering management must integrate technical and organizational knowledge in ways specific to the physical domain.

HRO theory implies that better hardware management depends on culture as much as process. Processes can be added to an organization without changing how it responds to risk. Design reviews can be mandated without changing the culture's relationship to dissent; risk registers can be maintained without changing willingness to act on them; simulation campaigns can be funded without changing tolerance for their schedule implications. Unless the culture makes clear that raising a safety concern is rewarded, that delays are accepted to address genuine risk, and that the person with the most relevant

DOI: 10.21791/IJEMS.2026.13

knowledge has standing to stop a programme, the processes may operate mainly as compliance mechanisms rather than effective governance tools. Gorod et al. [32] reached a parallel conclusion: governance in complex physical systems fails when it optimizes for procedural compliance rather than for the quality of technical judgment moving through the organization.

NASA offers a longitudinal illustration. The agency that landed humans on the Moon in 1969 treated irreversibility as the paramount constraint: components tested to failure, interfaces documented, critical decisions reviewed by independent engineers with authority to block progress. That culture eroded over subsequent decades under budget and schedule pressure, and the erosion contributed to the Challenger loss in 1986 and the Columbia loss in 2003. In both, managers overrode engineering concerns because the culture had drifted from safety preoccupation toward schedule optimization. The Columbia Accident Investigation Board [51] concluded that the drift was systemic and required deliberate institutional effort to reverse. Its findings suggest that irreversibility management depends on reward structures, communication patterns, and decision norms, rather than on review gates alone.

6. A Hardware-Native Engineering Management Framework

Effective management for hardware-intensive systems needs a framework built from first principles suited to physical systems, rather than adapted from software. Drawing on the physical distinctions above, HRO theory, resilience engineering, and operational experience across hardware industries, we propose five organizing principles. Figure 3 maps each principle to the physical property that motivates it and the HRO trait it anchors to.

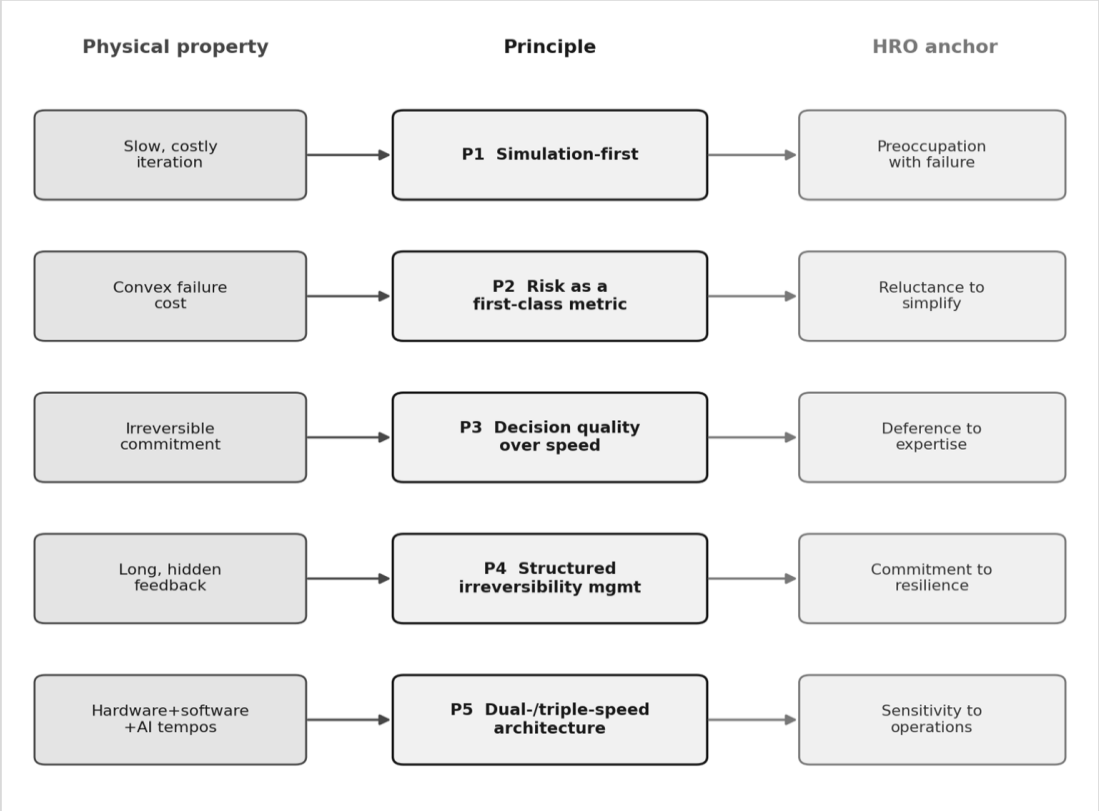

Figure 3. Mapping of hardware-native management principles to domain properties and high-reliability organization anchors. Source: prepared by the authors.

DOI: 10.21791/IJEMS.2026.13

6.1. Simulation-Based Design and Pre-Commitment Learning

If hardware's defining constraint is that physical iteration is slow and expensive, the first principle follows directly: move as much iteration as possible into simulation before committing physical resources. Implementing this principle requires organizational changes: hiring engineers skilled in modelling and simulation, maintaining simulation infrastructure, allocating calendar time for campaigns that produce no visible physical deliverable, and resisting pressure to start building before simulation work is complete. The insight behind digital twins is that mistakes made in simulation are cheap; the simulation environment becomes the place where hypotheses are tested and confidence is built before resources are committed. Teich [52] documented how simulation-based approaches transformed the semiconductor industry by letting designers explore architectural alternatives at a fraction of the cost of physical prototyping, and the same logic extends to any hardware domain.

AI may strengthen this principle rather than threaten it. While AI automates the writing of software, it can also accelerate the construction and execution of simulations, generating parametric variations, running optimization sweeps, and flagging anomalies faster than human analysts. AI investment in hardware organizations should therefore go toward simulation infrastructure, not toward automating the design judgments that require physical intuition. The managerial challenge is assessing the level of confidence that a given simulation can support. We propose a discipline of assumption auditing: before a model informs an irreversible decision, the team names the three most consequential assumptions and estimates the effect of each being wrong by a material amount. If a twenty per cent error in any one would change the decision, either the model fidelity must improve in that area or a physical prototype must validate the assumption before commitment.

6.2. Risk Governance as a Primary Management Function

Velocity, throughput, and customer satisfaction measure output rather than risk exposure. Hardware management must treat risk as a primary metric, not as an afterthought reviewed by a separate function. Failure mode and effects analysis, developed by the U.S. military in the 1940s and adopted across aerospace, automotive, medical-device, and nuclear industries, forces teams to ask how a system can fail at every component, interface, and process step, and to design mitigations rather than relying on testing to catch problems. FMEA without authority to change the design based on its findings has limited governance value. The analysis must bind the design: a high-severity failure mode without adequate mitigation should require a design change even when that delays the schedule. Luther et al. [49] found that the frameworks most effective at preventing catastrophic failure pair structured failure analysis with governance that allows risk assessors to halt production, which reinforces binding FMEA to design authority rather than treating it as documentation.

A complementary metric tracks uncertainty reduction per unit of investment: maintain an explicit list of open uncertainties about system performance, rank them by consequence, and measure weekly progress in retiring the highest-consequence items. This redirects attention from visible output toward less visible but consequential risk. Silvius and Schipper [53] found that the ability to reason about uncertainty and long-term consequence was among the least developed competencies in their sample,

DOI: 10.21791/IJEMS.2026.13

which helps explain why hardware organizations can struggle with risk-centric management. In this model, the manager is responsible for risk architecture as well as delivery coordination.

6.3. Decision Quality and Independent Technical Review

Software organizations optimize for decision speed because immediate feedback makes rapid correction possible. Hardware organizations must optimize for decision quality because many decisions are irreversible: once a chip is fabricated, a transformer manufactured, or a foundation poured, reversal ranges from prohibitive to impossible. This calls for a different posture at decision gates. Design reviews should be genuine gates where independent engineers evaluate the quality of analysis, challenge assumptions, and are required to put concerns on the record. Asking every attendee to state one concern before discussion proceeds turns the default from polite silence into structured dissent; the concern may prove minor, but the discipline of raising it surfaces assumptions that would otherwise stay buried until they produce a failure. The Columbia Accident Investigation Board [51] found that NASA's culture had suppressed dissent before the disaster, and concluded that the suppression of engineering judgment was as causally significant as the physical damage. Omurtag [54] argued that the field's standing challenge is building structures that protect technical judgment from schedule and budget pressure, a challenge that sharpens as physical systems grow more complex.

6.4. Irreversibility, Optionality, and Staged Commitment

Many organizations acknowledge irreversibility in the abstract while failing to manage it operationally. Hardware management needs explicit mechanisms to track where decisions fall on the reversibility spectrum and to ensure that high-consequence, low-reversibility decisions receive proportionally more analysis. Staged commitment offers one mechanism: a series of gates at which the organization decides to proceed, redirect, or stop, organized around irreversible decision points rather than calendar milestones. A project passes a gate when the organization understands and manages the remaining risks, rather than when it produces a set of documents. This approach is set-based concurrent engineering by another name: Toyota carries several design alternatives in parallel and narrows them as uncertainty falls, which is precisely the optionality that staged commitment preserves [20]. Preserving optionality has direct cost, but it can reduce the expected cost of late discovery after irreversible commitment. Žižlavský [55] found that organizations using stage-gate models with genuine authority at each gate significantly outperformed those using gates as ceremonial checkpoints.

Pre-mortems are a complementary technique. Before crossing a major threshold, the team envisions the decision's failure and explains the reasons behind it. The reframe from "what could go wrong" to "given that it went wrong, what caused it" allows engineers permission to voice concerns they might otherwise suppress. Red teams, experienced engineers tasked with finding flaws from a position of constructive adversarialism, serve the same function at larger scale and work best when staffed by people not invested in the design's success.

DOI: 10.21791/IJEMS.2026.13

6.5. Multi-Tempo Organizational Architecture

Most hardware companies are also software companies. An electric vehicle is a motor and battery pack run by millions of lines of code; a grid-scale storage system is a rack of cells managed by software; a fab is a set of process tools orchestrated by a manufacturing execution system. Teich [52] documented how this convergence has accelerated since the 1990s. AI adds a third speed: models retrained and redeployed on timescales of days. Organizations building AI-enabled hardware must manage three tempos, not two. The structural answer is to decouple the development paths as far as possible and define explicit interfaces between them. The hardware team fixes a stable platform, a set of physical specifications, electrical interfaces, and performance parameters, against which the software team builds and tests using a digital model until physical hardware arrives. The manager's role shifts from coordinating day-to-day work across incompatible timelines to maintaining the integrity of the interface.

Liang et al. [56] found an alignment paradox: tight intellectual alignment between business and technology functions can reduce agility by creating inertia, while social alignment, informal coordination and shared understanding, enables emergent coordination. The lesson translates directly: locking hardware and software teams into rigid shared plans produces the inertia that slows both, whereas strong informal channels across the boundary let each adapt without destabilizing the other. Formal interface contracts must therefore be supported by informal coordination mechanisms that help teams resolve exceptions and maintain alignment. Managing across tempos is a dynamic-capabilities problem. Teece [57] defined dynamic capabilities as the capacity to sense opportunities, seize them through resource reconfiguration, and transform internal structures when the environment shifts; a dual- or triple-speed organization exercises this capacity each time it renegotiates the interface between its fast software layer and its slow physical layer. Töytäri et al. [58] found that successful adopters aligned mindset and capabilities at once, using institutional logics to coordinate the cultural shift while building dynamic capabilities to execute the operational one.

The boundary between hardware and software is a common source of integration failure. The pattern is familiar: one team changes a parameter, the other team does not receive or act on the change, and the mismatch appears months later at integration, when correction is more expensive. Treating the interface as a formal, versioned contract that requires sign-off from both teams before any parameter changes, with a maintained change history, reduces this failure mode. Gorod et al. [32] argued that the quality of interface management predicts system integration success more reliably than any other single variable. Organizationally, the structure works best as a matrix: engineers belong to a functional discipline that maintains standards and peer review while being assigned to product teams that manage delivery. The critical requirement is that the functional discipline hold real authority, including the ability to veto a design decision that violates technical standards over the objection of a product-team lead. Without that authority, the matrix can become a product-team structure with a functional label. Kocaoglu [11] identified this balance between disciplinary depth and cross-functional integration as a standing structural challenge of engineering management.

DOI: 10.21791/IJEMS.2026.13

Principle	HRO anchor	Key practices	Replaces (software default)
P1 Simulation-first	Preoccupation with failure	Digital twins, assumption auditing, fidelity-graded campaigns	Ship and observe; learn from production failures
P2 Risk as first-class metric	Reluctance to simplify	FMEA bound to design authority, uncertainty-reduction tracking	Velocity and throughput as primary metrics
P3 Decision quality over speed	Sensitivity to operations; deference to expertise	Structured dissent at gates, independent technical review	Bias toward action; fast decisions, fast correction
P4 Irreversibility management	Commitment to resilience	Staged commitment, set-based options, pre-mortems, red teams	Calendar milestones; MVP and iterate
P5 Multi-speed architecture	All five (organizational design)	Versioned interface contracts, platform decoupling, functional veto	Single-speed product teams; uniform sprints

Table 2. Hardware-native management principles, high-reliability organization anchors, associated practices, and corresponding software-derived defaults. Source: prepared by the authors.

7. Relationship to Established Industrial Standards

A fair objection to any new framework is that industry already has mature standards for managing physical work. The five principles therefore need to be measured against those standards rather than asserted in isolation. We compare the framework with seven established bodies of practice and state what it recombines and what it adds. Figure 4 summarizes the comparison as a coverage map and Table 3 provides the details.

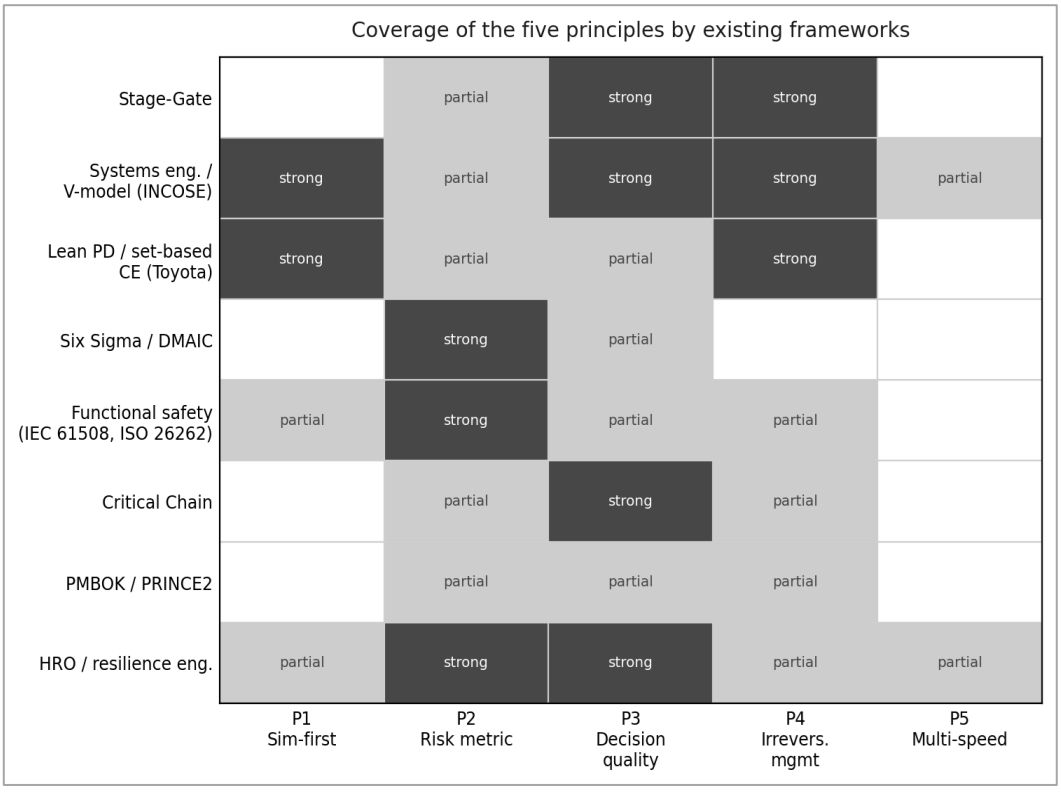

Figure 4. Coverage of hardware-native management principles across established industrial and engineering management frameworks. Source: prepared by the authors.

Stage-Gate, introduced by Cooper, structures product development as phases separated by go/kill gates [59]. It supports staged commitment (P4) and disciplined decision review (P3), but its gates are often tied to calendar progress and treated as approval rituals, which is the failure mode P4 is meant to

DOI: 10.21791/IJEMS.2026.13

prevent. Systems engineering, codified in the INCOSE handbook and the ISO/IEC/IEEE 15288 life-cycle standard, is the closest existing relative [60]. The V-model front-loads requirements, interface control, and verification, covering P1, P3, and P4 while addressing P5 only partly through interface management. What it underweights is the cultural and institutional account supplied by HRO theory, and it predates the AI-driven tempo problem that P5 must now address.

Lean product development and Toyota's set-based concurrent engineering are, by the argument of Section 2, the disciplined originals the framework is partly recovering [6], [20]. Set-based engineering is the strongest existing expression of P4: carry multiple design sets, delay commitment, narrow as uncertainty falls. Its limit is that it was articulated for automotive development and does not address the convex-failure and risk-metric concerns (P2) that dominate in aerospace, power, and process industries. Six Sigma and the DMAIC cycle excel at P2, reducing variation and defects in defined processes, but say little about P1, P4, or P5 and assume a process stable enough to measure, which early-stage hardware development typically lacks. Functional-safety standards, IEC 61508 and its automotive specialization ISO 26262, are the most rigorous existing treatment of risk as a binding constraint (P2) and contribute to irreversibility management (P4) through safety lifecycles [61], [62]; they are domain-bounded to safety-critical systems and do not speak to organizational tempo or simulation strategy. Critical-chain project management, from Goldratt, sharpens decision quality around resource constraints and buffer management (P3, partial P4) but is silent on the physical-properties rationale that motivates the whole framework [63]. Generic project bodies of knowledge such as PMBOK and PRINCE2 are method-agnostic by design and therefore cover each principle only weakly. HRO theory and resilience engineering, treated in Section 6, supply the cultural backbone for P2 and P3 but were not written as a development framework and offer little on P1 or P5.

Framework	Origin and core mechanism	Strongest overlap	What hardware-native management adds
Stage-Gate [59]	Marketing/NPD; phase-gate go/kill decisions	P4, P3	Gates tied to uncertainty retirement, not calendar; physics rationale
Systems engineering / V-model [60]	Aerospace, defence; requirements and verification	P1, P3, P4	HRO culture, AI tempo (P5), institutional reframe
Lean PD / set-based CE [6], [20]	Toyota manufacturing; delayed commitment, option sets	P1, P4	Convex-failure risk metric (P2), multi-speed coordination (P5)
Six Sigma / DMAIC	Manufacturing quality; variation reduction	P2	Front-end simulation (P1), irreversibility (P4), tempo (P5)
Functional safety [61], [62]	Process and automotive safety; safety lifecycle	P2, P4	Whole-development scope beyond safety; simulation and tempo
Critical chain [63]	Operations; constraint and buffer management	P3	Physical-properties rationale; risk and simulation emphasis
HRO / resilience eng. [46]–[50]	Sociology of safety; vigilance and adaptive capacity	P2, P3	Translation into a development framework with metrics

Table 3. Comparative positioning of hardware-native management relative to established industrial frameworks. Source: prepared by the authors.

The comparison clarifies the contribution and bounds the novelty of the claim. Several individual practices in the framework are not new; they are recovered from manufacturing and systems engineering, as Section 2 argues they should be. Four contributions lie in the combination. First, the framework offers a physics-grounded rationale that explains why these scattered practices belong together and when each applies, rather than presenting them as a checklist. Second, it accounts for the possibility that AI assistance may make some software competencies less distinctive while increasing the relative value of hardware-related competencies. Third, it identifies the multi-speed architecture

DOI: 10.21791/IJEMS.2026.13

(P5) that existing standards largely miss, because simultaneous management of hardware, software, and AI tempos is a recent problem. Fourth, it adds an institutional explanation of why correction is difficult and where pressure must be applied. The framework should therefore be read as a synthesis that links systems engineering, lean development, physical-domain constraints, and current shifts in engineering work.

8. Operational Measures, Maturity Levels, and Diagnostic Application

This section provides measurable indicators for each principle, a five-level maturity model, and a short readiness diagnostic. The metrics are deliberately leading rather than lagging, because hardware-native management is meant to identify and reduce exposure before failures become expensive or irreversible.

Table 4 translates the five hardware-native management principles into observable leading indicators and the organizational mechanisms needed to act on them. The rationale and implementation of each row are developed in Sections 6.1–6.5: simulation-to-physical iteration, assumption auditing, and model fidelity are discussed under simulation-based design (Section 6.1); uncertainty retirement and binding FMEA are explained under risk governance (Section 6.2); concern logging and independent technical review are addressed under decision quality (Section 6.3); staged gates, option sets, pre-mortems, and red teams are developed under irreversibility management (Section 6.4); and interface lead time, integration defects, and versioned interface contracts are discussed under multi-tempo organizational architecture (Section 6.5).

Principle	Example leading metrics	Organizational mechanism
P1 Simulation-first	Simulation-to-physical iteration ratio; assumption-audit coverage (share of irreversible decisions with a documented top-three sensitivity analysis); tracked model-to-reality error	Dedicated modelling and simulation function; fidelity-graded gate criteria
P2 Risk as first-class metric	Mean time to retire the top-consequence uncertainty; share of high-severity FMEA items with a binding mitigation; weekly risk burn-down	Named risk-architect role; FMEA bound to design-change authority
P3 Decision quality over speed	Concerns logged per design review; share of decisions with independent technical review; decision-reversibility classification coverage	Mandatory concern-raising protocol; independent technical authority that can stop work
P4 Irreversibility management	Share of irreversible decisions passing a true staged gate versus a calendar milestone; number of live design sets carried into each gate; pre-mortem and red-team coverage	Irreversibility register; gate criteria tied to uncertainty retirement
P5 Multi-speed architecture	Interface-change lead time; interface-defect escape rate at integration; cross-tempo synchronization cadence	Versioned interface contract; functional discipline with veto authority

Table 4. Operational indicators and organizational mechanisms for hardware-native management principles. Source: prepared by the authors.

Metrics need an accompanying trajectory, otherwise they can become another reporting artifact. Table 5 supplies that trajectory through five maturity levels, moving from software-default routines to a physics-native regime in which the principles are measured, enforced, and treated as ordinary management discipline. The model is a self-assessment scaffold, not a certification scheme, and an organization may occupy different levels for different principles.

DOI: 10.21791/IJEMS.2026.13

Level	Name	What it looks like across the five principles
1	Software-default	Calendar sprints applied to physical work; risk is a quarterly register; gates are approval rituals; no simulation strategy; single-tempo product teams
2	Aware	Leaders can name the mismatch; some design reviews invite dissent; FMEA exists but does not bind the design; simulation used ad hoc
3	Instrumented	The metrics in Table 4 are collected; an irreversibility register exists; a modelling and simulation function is staffed; interfaces are documented
4	Bound	Metrics carry consequences: high-severity risks block release, gates turn on uncertainty retirement, functional authority can stop work, interface changes require dual sign-off
5	Physics-native	The posture is cultural, not procedural; the regime is matched to the local physics of each decision; AI is directed at simulation; disciplined manufacturing and systems-engineering practices are recombined as default management routines

Table 5. Maturity levels for hardware-native engineering management. Source: prepared by the authors.

The diagnostic is an initial assessment a leadership team can complete in an afternoon. Five questions, one per principle, distinguish formal adoption from operational use. For P1: before the last irreversible commitment, did the team document its three most consequential assumptions and what a material error in each would cost? For P2: does any high-severity failure mode currently lack a mitigation, and if so, who has the authority to stop release because of it? For P3: at the last major design review, how many concerns were logged, and what happened to them? For P4: of the irreversible decisions made last quarter, how many passed a gate keyed to retired uncertainty rather than a date? For P5: what is the lead time for a change to the hardware-software interface, and how many integration defects last cycle traced to an interface mismatch without documented sign-off? An organization that cannot answer these questions is likely to be at Level 1 or 2, regardless of the number of agile ceremonies it conducts.

9. Retrospective Case Analysis

This section presents a structured retrospective analysis of four documented cases: three failures and one contrasting success, each examined against the five principles. The method is theory-illustrating multiple-case analysis in the sense of Eisenhardt [64]: the cases are used to test explanatory fit and probe boundary conditions, not to establish causation. We state the limits of this design at the end of the section and convert them into the prospective study proposed in Section 12. Cases were chosen for the depth of their public investigative record, which biases the sample toward well-documented disasters and away from quiet successes, a point we return to under limitations.

The Boeing 737 MAX illustrates failures of P2 and P3. A flight-control function, MCAS, was allowed to act on a single angle-of-attack sensor with authority to repeatedly push the nose down, a high-severity failure mode that reached service without a binding mitigation, which corresponds to the P2 failure of an unbound risk analysis. Engineering concerns about the design lacked sufficient authority to halt the project, illustrating the P3 failure of decision quality under schedule and commercial pressure. The outcome was 346 deaths, a roughly twenty-month grounding of the fleet, and direct costs exceeding twenty billion dollars. The framework identifies a specific decision point at which a bound FMEA and an independent authority to halt could have changed the design path.

The Space Shuttle Columbia loss in 2003 is a failure of P3 and of the HRO culture behind it. Engineers who raised concerns about foam strikes were overruled by managers under schedule pressure, and the Columbia Accident Investigation Board concluded that the suppression of technical dissent was as causal as the physical damage to the thermal protection system [51]. In terms of the framework, the case

DOI: 10.21791/IJEMS.2026.13

illustrates a shift away from deference to expertise toward positional authority under schedule pressure, the failure mode that P3 seeks to prevent.

The 2003 Northeast blackout shows the convex cost curve of P2. The proximate chain began when an overloaded transmission line sagged into trees in Ohio, and a defect in the alarm-system software in the FirstEnergy control room left operators unaware that they needed to redistribute the load. A local, recoverable fault cascaded into the loss of power for roughly fifty-five million people across eight states and Ontario, with damages near six billion dollars and an estimated toll approaching a hundred deaths. The software defect was small, but its consequence was amplified by a tightly coupled physical system. A management posture that treated the alarm path as a high-severity, consequence-ranked uncertainty (P2) rather than a routine software component would have assigned it a different level of review.

The contrasting case is SpaceX, because it appears at first to challenge the framework. SpaceX deliberately flies prototypes to failure, including early Starship test articles, in order to learn quickly. This approach does not contradict simulation-first or irreversibility management when the local cost function has been deliberately reduced. By lowering unit cost and build cadence through stainless-steel construction, high production rate, and reuse, SpaceX reduced the marginal cost and feedback latency of iteration for uncrewed test articles, moving that work toward the controlled-iteration quadrant of Figure 1. Where irreversibility remains high and lives are at stake, such as crewed flight, it remains more conservative, qualifying the human-rated vehicle through extensive analysis and test before commitment. The case is therefore consistent with the claim that the appropriate management regime depends on the local physics of the decision. SpaceX illustrates the application of P1 and P4, with explicit attention to distinguishing between reversible and irreversible decisions.

Table 6 consolidates the four case analyses presented in the preceding paragraphs and identifies the principle and decision point illuminated by each case. The Boeing 737 MAX row summarizes the discussion of unbound high-severity risk and suppressed technical authority; the Columbia row summarizes the discussion of deference to expertise under schedule pressure; the Northeast blackout row summarizes the discussion of convex failure cost and the hardware-software interface; and the SpaceX row summarizes the contrasting case in which lower iteration cost and feedback latency make controlled failure appropriate. The underlying concepts are developed in the sections on failure-cost structures and risk governance (Section 4), high-reliability organizations and resilience engineering (Section 5), and the five framework principles (Sections 6.1–6.5).

Case	Principles most engaged	What the framework explains
Boeing 737 MAX	P2 (unbound risk), P3 (suppressed dissent)	A high-severity single-sensor failure mode reached service without a binding mitigation or an authority to halt
Columbia 2003	P3, HRO culture	Drift from deference-to-expertise to positional authority under schedule pressure overrode a known physical risk
Northeast blackout 2003	P2 (convex cost), P5 (interface)	A small software defect on an under-ranked path cascaded across a tightly coupled physical system
SpaceX (contrasting)	P1, P4 applied well	Iteration-first is appropriate where the local cost function has been reduced; more conservative controls remain appropriate where irreversibility remains high

Table 6. Cross-case assessment of selected hardware-intensive systems against the proposed principles. Source: prepared by the authors.

The case analysis supports explanatory fit and face validity. In each case, the framework identifies a decision or governance mechanism rather than a broad cultural diagnosis. The SpaceX contrast also

DOI: 10.21791/IJEMS.2026.13

shows that the argument is not caution as a default, but fit between the management regime and the local physics of the decision. The analysis does not establish predictive or causal validity, because the cases were selected retrospectively for the richness of their records. Testing whether the framework would flag such decisions in advance requires the prospective study proposed in Section 12.

Discussion

The framework carries several practical implications. The most immediate is that organizations moving into hardware-intensive domains should audit their management vocabulary before auditing their processes, because the words a team uses to discuss its work encode assumptions beneath it. Terms borrowed from software carry assumptions about reversibility, feedback, and failure cost that may not hold. Organizations can therefore examine whether terms such as sprint, MVP, and velocity are shaping schedule estimates, review practices, or expectations about recoverability in ways that conflict with physical constraints. The change is structural rather than cosmetic; vocabulary alters how teams reason about the work.

The institutional analysis in Section 2 explains why vocabulary audits, while necessary, are not sufficient. The universalization of software management reshaped reward structures, hiring criteria, curricula, and professional identities across the field, and reversing it requires working at the same level. Gümüsay et al. [65] found that organizations facing incompatible logics did best through elastic hybridity, stretching and contracting the boundary between logics by context rather than choosing one or partitioning rigidly. For firms that still employ software engineers alongside mechanical and electrical teams, elastic hybridity is a more realistic model than wholesale replacement: software logic is appropriate where software properties hold, hardware logic must govern where they do not, and the management task is to identify the boundary and staff the interface with people who understand both sides. Hitt et al. [66] argued that management research needs bridges across levels, from individual cognition to field-level forces, and the framework operates across all of them, which its implementation will require too: individual competency development, team-level process change, organizational restructuring, and field-level work to legitimize hardware-native management as a discipline.

Two possible objections clarify the scope of the framework. The first is that the management approach is not chosen freely, so blaming managers for adopting agile misreads the situation. We agree, and the framework depends on that point: the physics constrains the regime, and the institutional account in Section 2 shows that firms adopted the lighter method under field-level pressure rather than through individual error. The prescription follows the diagnosis, working at the institutional level rather than scolding managers. The second objection is that lean, kanban, and agile came from manufacturing in the first place, so telling hardware to abandon them is incoherent. We also agree with this point, as it forms the core of the argument. Hardware should recover their disciplined originals, including set-based commitment, front-loaded analysis, and interface control, rather than import the lighter software reinterpretation.

The AI dimension sharpens the implications. As AI absorbs routine software tasks, professionals displaced from software roles may move toward hardware-adjacent industries and bring software-derived work habits with them. Organizations hiring from this pool will need to decide deliberately

DOI: 10.21791/IJEMS.2026.13

which habits transfer and which require retraining. The iterative mindset is appropriate where decisions are reversible; it is problematic when applied to irreversible decisions [67]. Sergeyuk et al. [14] found that engineers who used AI assistants most effectively were those who understood the limits of automation and maintained critical judgment about its output; this same calibrated skepticism, knowing when to trust the tool and when to override it, is what hardware management requires from human decision-makers at irreversible gates.

The management development pipeline also requires revision. Programmes that teach management primarily through the lens of software and financial services may produce managers who misapply those lessons in physical domains. Barrett [40] found that current engineering management competency frameworks underweight risk cognizance, systems thinking, and long-horizon accountability, the competencies that hardware contexts demand. Hardware-literate leadership requires exposure to systems engineering, reliability engineering, and the institutional practice of industries with long experience managing irreversibility [18], [19]. The framework also connects to the innovation literature: Agarwal and Brem [68] argued that frugal innovation under severe constraint demands practices unlike those from resource-rich software environments, and hardware-native management shares that constraint-driven emphasis. Pollack et al. [69] found, across the classic megaproject works, that severe project failures often involved management approaches that underestimated physical constraints; the framework addresses that tendency by treating those constraints as boundary conditions for management design.

Limitations and Future Research

The framework has specific limits. The distinction between bits and atoms is a gradient, not a binary. Modern hardware is deeply software-dependent and modern software often touches physical infrastructure [37], [52], so the five principles are sharpest at the hardware-dominant pole and less prescriptive toward the software-dominant pole. A fuller theory would model that gradient and specify where regime boundaries fall for a given technology. The regime matrix in Figure 1 is a first step, but it treats the axes as continuous without locating those boundaries.

A second limit concerns novelty. As Section 8 shows, several principles recombine practices that already exist in systems engineering and lean development; the contribution is the physics-grounded synthesis, the AI inversion, the multi-speed architecture, and the institutional reframe, not the invention of each practice. A third limit concerns prescription. The SpaceX case shows that where an organization can lower its local cost function, iteration-first becomes rational, so the framework prescribes fit to physics rather than caution everywhere. A fourth limit is methodological. The case analysis in Section 10 is retrospective and selected for documentary richness, which invites hindsight and confirmation bias. The framework's generalization and predictive validity are not yet established, which is expected for a proposed model but should be tested directly.

These limits define the research agenda. The most important study is prospective: identify hardware programmes at their outset, score each on the Table 4 metrics and the maturity model, and track outcomes (cost and schedule variance, in-service failures, rework after commitment) to test whether the framework predicts rather than merely explains. A complementary line would run longitudinal case

DOI: 10.21791/IJEMS.2026.13

studies of organizations that suffered major hardware failures, mapping the specific software-derived practices that preceded them into an empirical typology. A third would test the boundary conditions of individual principles, comparing the effectiveness of simulation-first and staged commitment across domains with different cost functions. A fourth would study the migration of AI-displaced software professionals into hardware industries and the management challenges it creates, a question with immediate practical stakes. A fifth would extend the competency frameworks of Barrett [40] and Silvius and Schipper [53] into a validated assessment instrument for hardware management readiness. A sixth would examine the institutional dynamics of dislodging entrenched software-derived logic, including the role of professional associations, accreditation bodies, and educational institutions in legitimizing the disciplined alternatives. These studies would convert the framework's conceptual propositions into empirically testable claims.

Conclusion

Many management methods associated with the first two decades of this century were abstracted from hardware manufacturing, adapted to software's more reversible and faster-feedback medium, and later reapplied to hardware in lighter form. Software's physics made iterative, fast-moving, low-documentation management a good fit for code: rollback is easier, feedback is faster, most failures are cheaper, distribution is nearly instant, and the marginal cost of change is low. Those properties do not characterize steel, concrete, silicon wafers, power electronics, or biological manufacturing.

Physical systems are receiving renewed attention through energy infrastructure, semiconductor fabs, defence systems, robotics, and advanced materials. At the same time, the profession that generated the lightened methods is being changed by the technology it built, as AI assists with coding, testing, debugging, and documentation. The lightened methods are therefore being applied to hardware while the professional practice that helped legitimize them is itself changing. If management practices persist after the conditions that justified them have changed, they risk becoming detached from empirical feedback and resistant to revision.

Hardware-native engineering management matches management practice to the physical properties of the work. Its five principles are simulation before fabrication, risk before throughput, decision quality before speed, explicit management of irreversibility, and organizational structures that let hardware, software, and AI move at their natural speeds while preserving interface discipline. The mismatch between inherited management philosophy and hardware requirements is a mismatch in assumptions about cost, feedback, reversibility, and consequence. Organizations that do not address it may misdiagnose hardware execution failures as implementation problems rather than as conflicts between management assumptions and physical constraints.

Author Contributions

Babu George: Theoretical framing, methodology, visualization, writing – original draft, writing – final draft. Divya Choudhary: Conceptualization, literature review, project administration.

DOI: 10.21791/IJEMS.2026.13

AI Use Statement

Microsoft Copilot was used to refine the manuscript's grammar, enhance stylistic elements, and support consistency of tone. An AI-driven script developed by the first author was used to batch-convert the references into the journal's prescribed citation format. The intellectual framing, argument, selection and interpretation of sources, conceptual framework, case discussion, and revision decisions remain the authors' responsibility.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] T. J. Kloppenborg and W. A. Opfer, "The current state of project management research: trends, interpretations, and predictions," *Project Manag. J.*, vol. 33, no. 2, pp. 5–18, 2002, doi: 10.1177/875697280203300203.

[2] P. W. G. Morris, *Reconstructing Project Management*. Chichester, U.K.: Wiley, 2013.

[3] K. Beck et al., "*Manifesto for agile software development,"* 2001. [Online]. Available: https://agilemanifesto.org

[4] H. Takeuchi and I. Nonaka, "The new new product development game," *Harvard Bus. Rev.*, vol. 64, no. 1, pp. 137–146, Jan.–Feb. 1986.

[5] T. Ohno, *Toyota Production System: Beyond Large-Scale Production*. Cambridge, MA, USA: Productivity Press, 1988.

[6] J. P. Womack, D. T. Jones, and D. Roos, *The Machine That Changed the World.* New York, NY, USA: Rawson Associates, 1990.

[7] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*. Boston, MA, USA: Addison-Wesley, 2003.

[8] B. George and J. Paul, *Digital Transformation in Business and Society*. New York, NY, USA: Springer International, 2020.

[9] B. R. Sarchet, "Engineering management: key to the future," *Eng. Manag. J.*, vol. 1, no. 1, pp. 4–8, 1989, doi: 10.1080/10429247.1989.11414512.

[10] T. Kotnour and J. V. Farr, "Engineering management: past, present, and future," *Eng. Manag. J.*, vol. 17, no. 1, pp. 15–26, 2005, doi: 10.1080/10429247.2005.11415273.

[11] D. F. Kocaoglu, "Engineering management: where it was, where it is now, where it is going," *Eng. Manag. J.,* vol. 21, no. 3, pp. 39–42, 2009, doi: 10.1080/10429247.2009.11431815.

[12] B. George and O. S. Wooden, *AI Empowered: Pioneering African American Entrepreneurship in the Digital Age*. Bingley, U.K.: Emerald, 2025.

DOI: 10.21791/IJEMS.2026.13

[13] K. Handa, A. Tamkin, and M. McCain, *"Which economic tasks are performed with AI? Evidence from millions of Claude conversations*," arXiv:2503.04761, 2025, doi: 10.48550/arXiv.2503.04761.

[14] A. Sergeyuk, Y. Golubev, and T. Bryksin, "Using AI-based coding assistants in practice: state of affairs, perceptions, and ways forward," *Inf. Softw. Technol*., vol. 178, p. 107610, 2025.

[15] R. Khojah, M. Mohamad, and P. Leitner, "Beyond code generation: an observational study of ChatGPT usage in software engineering practice," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 1819–1840, 2024, doi: 10.1145/3660788.

[16] G. Buttazzo, "Rise of artificial general intelligence: risks and opportunities*," Front. Artif. Intell*., vol. 6, art. 1226990, 2023, doi: 10.3389/frai.2023.1226990.

[17] W. J. Lannes, "What is engineering management?," *IEEE Trans. Eng. Manag*., vol. 48, no. 1, pp. 107–115, 2002.

[18] H. Shah and W. Nowocin, "Yesterday, today and future of the engineering management body of knowledge," *Front. Eng. Manag.,* vol. 2, no. 2, pp. 127–132, 2015.

[19] R. C. Waters, "Engineering management tradition and education: past, present, and future," *Eng. Manag. J*., vol. 6, no. 3, pp. 15–20, 1994, doi: 10.1080/10429247.1994.11414791.

[20] A. Ward, J. K. Liker, J. J. Cristiano, and D. K. Sobek II, "The second Toyota paradox: how delaying decisions can make better cars faster," *Sloan Manag. Rev.,* vol. 36, no. 3, pp. 43–61, 1995.

[21] A. J. Shenhar and D. Dvir, "Project management evolution: past history and future research directions," in *Innovations: Project Management Research*, D. P. Slevin, D. I. Cleland, and J. K. Pinto, Eds. Newtown Square, PA, USA: PMI, 2004, pp. 57–64.

[22] R. Greenwood, M. Raynard, F. Kodeih, et al., "Institutional complexity and organizational responses," *Acad. Manag. Ann.,* vol. 5, no. 1, pp. 317–371, 2011, doi: 10.5465/19416520.2011.590299.

[23] K. Kauppi, "Extending the use of institutional theory in operations and supply chain management research," *Int. J. Oper. Prod. Manag.,* vol. 33, no. 10, pp. 1318–1345, 2013.

[24] D. Chandler and H. Hwang, "Learning from learning theory: a model of organizational adoption strategies at the microfoundations of institutional theory," *J. Manag*., vol. 41, no. 5, pp. 1446–1476, 2015, doi: 10.1177/0149206315572698.

[25] P. Bromley and W. W. Powell, "From smoke and mirrors to walking the talk: decoupling in the contemporary world," *Acad. Manag. Ann.,* vol. 6, no. 1, pp. 483–530, 2012, doi: 10.5465/19416520.2012.684462.

[26] M. G. Seo and W. E. D. Creed, "Institutional contradictions, praxis, and institutional change: a dialectical perspective," *Acad. Manag. Rev*., vol. 27, no. 2, pp. 222–247, 2002, doi: 10.2307/4134353.

[27] S. Gosain, "Enterprise information systems as objects and carriers of institutional forces: the new iron cage?," *J. Assoc. Inf. Syst*., vol. 5, no. 4, art. 6, 2004.

DOI: 10.21791/IJEMS.2026.13

[28] M. Smets, T. Morris, and R. Greenwood, "From practice to field: a multilevel model of practice-driven institutional change," *Acad. Manag. J.,* vol. 55, no. 4, pp. 877–904, 2012, doi: 10.5465/amj.2010.0013.

[29] B. Gray, J. M. Purdy, and S. Ansari, "From interactions to institutions: microprocesses of framing and mechanisms for the structuring of institutional fields," *Acad. Manag. Rev*., vol. 40, no. 1, pp. 115–143, 2015, doi: 10.5465/amr.2013.0299.

[30] J. M. Purdy and B. Gray, "Conflicting logics, mechanisms of diffusion, and multilevel dynamics in emerging institutional fields," *Acad. Manag. J.,* vol. 52, no. 2, pp. 355–380, 2009, doi: 10.5465/amj.2009.37308255.

[31] M. L. Besharov and W. K. Smith, "Multiple institutional logics in organizations: explaining their varied nature and implications," *Acad. Manag. Rev*., vol. 39, no. 3, pp. 364–381, 2014, doi: 10.5465/amr.2011.0431.

[32] A. Gorod, B. Sauser, and J. Boardman, "System-of-systems engineering management: a review of modern history and a path forward," *IEEE Syst. J.,* vol. 2, no. 4, pp. 484–499, 2008.

[33] A. Haldar, "Past, present, and future of engineering under uncertainty: safety assessment and management," in *Proc. Int. Symp. Eng. under Uncertainty (ISEUSAM-2012*), S. Chakraborty and G. Bhattacharya, Eds. New Delhi, India: Springer, 2013, pp. 1–25.

[34] C. Perrow, *Normal Accidents: Living with High-Risk Technologies*. New York, NY, USA: Basic Books, 1984.

[35] J. Bakhshi, V. Ireland, and A. Gorod, "Clarifying the project complexity construct: past, present and future," *Int. J. Proj. Manag*., vol. 34, no. 7, pp. 1199–1213, 2016.

[36] E. Bayraktar, M. C. Jothishankar, and E. Tatoglu, "Evolution of operations management: past, present and future," *Manag. Res. News*, vol. 30, no. 11, pp. 843–871, 2007.

[37] Y. Liao, F. Deschamps, and E. D. F. R. Loures, "Past, present and future of Industry 4.0: a systematic literature review and research agenda proposal," *Int. J. Prod. Res*., vol. 55, no. 12, pp. 3609–3629, 2017, doi: 10.1080/00207543.2017.1308576.

[38] A. S. Pothukuchi, L. V. Kota, and V. Mallikarjunaradhya, "Impact of generative AI on the software development lifecycle," *Int. J. Creat. Res. Thoughts*, vol. 11, no. 8, 2023.

[39] V. Vimpari, A. Kultima, and P. Hämäläinen, ""An adapt-or-die type of situation": perception, adoption, and use of text-to-image-generation AI by game industry professionals," *Proc. ACM Hum.-Comput. Interact.*, vol. 7, no. CHI PLAY, pp. 131–164, 2023, doi: 10.1145/3611025.

[40] C. V. Barrett, "*Engineering management competencies: a framework for present and future engineering environments*," M.S. thesis, Old Dominion Univ., Norfolk, VA, USA, 2020.

[41] M. Cristofaro and P. L. Giardino, "Surfing the AI waves: the historical evolution of artificial intelligence in management and organizational studies and practices," *J. Manag. Hist*., 2025, doi: 10.1108/JMH-01-2025-0002.

DOI: 10.21791/IJEMS.2026.13

[42] R. Kohli and N. P. Melville, "Digital innovation: a review and synthesis," *Inf. Syst. J.*, vol. 29, no. 1, pp. 200–223, 2019, doi: 10.1111/isj.12193.

[43] B. Flyvbjerg, "What you should know about megaprojects and why: an overview," *Proj. Manag. J.*, vol. 45, no. 2, pp. 6–19, 2014, doi: 10.1002/pmj.21409.

[44] G. M. Winch, "Escalation in major projects: lessons from the Channel Fixed Link," *Int. J. Proj. Manag.,* vol. 31, no. 5, pp. 724–734, 2013.

[45] J. Söderlund, S. Sankaran, and C. Biesenthal, "The past and present of megaprojects," *Proj. Manag. J.*, vol. 48, no. 6, pp. 5–16, 2017, doi: 10.1177/875697281704800602.

[46] K. E. Weick and K. M. Sutcliffe, *Managing the Unexpected: Assuring High Performance in an Age of Complexity*. San Francisco, CA, USA: Jossey-Bass, 2001.

[47] J. Pariès, L. Macchi, C. Valot, et al., "Comparing HROs and RE in the light of safety management systems," *Saf. Sci*., vol. 117, pp. 501–511, 2019.

[48] M. Sujan, O. Lounsbury, L. Pickup, et al., "Resilience engineering concepts, Safety-II language, FRAM practice, and co-creating communities: how Hollnagel reshaped patient safety," *Saf. Sci*., vol. 200, art. 107210, 2026.

[49] B. Luther, I. Gunawan, and N. Nguyen, "Identifying effective risk management frameworks for complex socio-technical systems," *Saf. Sci.,* vol. 158, art. 105989, 2023.

[50] P. M. Salmon, N. A. Stanton, G. H. Walker, et al., *Handbook of Systems Thinking Methods*. Boca Raton, FL, USA: CRC Press, 2023.

[51] Columbia Accident Investigation Board, *Report of the Columbia Accident Investigation Board*, vol. 1. Washington, DC, USA: U.S. Government Printing Office, 2003.

[52] J. Teich, "Hardware/software codesign: the past, the present, and predicting the future," *Proc. IEEE*, vol. 100, Special Centennial Issue, pp. 1411–1430, 2012.

[53] A. J. G. Silvius and R. P. J. Schipper, "Sustainability in project management competencies: analyzing the competence gap of project managers," *J. Hum. Resour. Sustain. Stud*., vol. 2, no. 2, pp. 40–58, 2014.

[54] Y. Omurtag, "Engineering management: past, present and a brief look into the future," *Eng. Manag. J.,* vol. 21, no. 1, pp. 3–4, 2009, doi: 10.1080/10429247.2009.11431819.

[55] O. Žižlavský, "Past, present and future of the innovation process," *Int. J. Eng. Bus. Manag*., vol. 5, art. 47, 2013, doi: 10.5772/56920.

[56] H. Liang, N. Wang, Y. Xue, et al., "Unraveling the alignment paradox: how does business-IT alignment shape organizational agility?," *Inf. Syst. Res*., vol. 28, no. 4, pp. 863–879, 2017, doi: 10.1287/isre.2017.0711.

[57] D. J. Teece, *Dynamic Capabilities: Foundational Concepts*. Cambridge, U.K.: Cambridge Univ. Press, 2025.

DOI: 10.21791/IJEMS.2026.13

[58] P. Töytäri, T. Turunen, M. Klein, et al., "Aligning the mindset and capabilities within a business network for successful adoption of smart services," *J. Prod. Innov. Manag*., vol. 35, no. 5, pp. 763–779, 2018, doi: 10.1111/jpim.12462.

[59] R. G. Cooper, "Stage-gate systems: a new tool for managing new products," *Bus. Horiz*., vol. 33, no. 3, pp. 44–54, 1990. https://doi.org/10.1016/0007-6813(90)90040-I

[60] INCOSE*, Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*, 5th ed. Hoboken, NJ, USA: Wiley, 2023.

[61] Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, IEC 61508, 2010.

[62] Road Vehicles: Functional Safety, ISO 26262, 2018.

[63] E. M. Goldratt, *Critical Chain. Great Barrington*, MA, USA: North River Press, 1997.

[64] K. M. Eisenhardt, "Building theories from case study research," *Acad. Manag. Rev.*, vol. 14, no. 4, pp. 532–550, 1989, doi: 10.2307/258557.

[65] A. A. Gümüsay, M. Smets, and T. Morris, ""God at work": engaging central and incompatible institutional logics through elastic hybridity," *Acad. Manag. J*., vol. 63, no. 1, pp. 124–154, 2020, doi: 10.5465/amj.2016.0481.

[66] M. A. Hitt, P. W. Beamish, S. E. Jackson, et al., "Building theoretical and empirical bridges across levels: multilevel research in management," *Acad. Manag. J*., vol. 50, no. 6, pp. 1385–1399, 2007, doi: 10.5465/amj.2007.28166219.

[67] L. Yang, T. L. Henthorne, and B. George, "Artificial intelligence and robotics technology in the hospitality industry," in *Digital Transformation in Business and Society*, B. George and J. Paul, Eds. New York, NY, USA: Springer, 2019, pp. 211–228.

[68] N. Agarwal and A. Brem, "Frugal innovation: past, present, and future," *IEEE Eng. Manag. Rev*., vol. 45, no. 3, pp. 37–41, 2017.

[69] J. Pollack, C. Biesenthal, S. Sankaran, et al., "Classics in megaproject management: a structured analysis of three major works," *Int. J. Proj. Manag.,* vol. 36, no. 2, pp. 372–384, 2018.