Available on: http://gjetr.ep-journals.org/ E-ISSN: 3051-3782, P-ISSN: 3051-3774

Global Journal of Engineering and Technology Review

Volume: 02 Issue: 09 September 2026

A Hardware-In-The-Loop CI/CD Validation Framework for IoT and Vehicle Embedded Systems Using C# and Azure Devops

Kingsley Chinazaekpere Ndupu
Kennesaw State University, Georgia, USA

Emmanuel Eniola Ajala
Independent Researcher, Atalanta, Georgia

Serif Oyindamola Oyesiji
Harrisburg University of Science and Technology, Harrisburg, PA

Stanley Nwakamma
Independent Researcher, USA

Published Date: September 09, 2026 **Article DOI : 10.65150/EP-gjetr/V2E9/2026-10**

ABSTRACT: Continuous integration and delivery transformed mainstream software engineering, yet embedded development for Internet of Things devices and vehicle systems remains only partially served by it, because the property that matters most, correct behavior of software on real hardware interacting with real signals, is exactly the property that conventional build-and-unit-test pipelines never exercise. Hardware-in-the-loop validation closes that gap in principle, but in most organizations, it lives outside the delivery pipeline, operated manually, scheduled informally, and consulted late, so integration defects that only hardware reveals are discovered at the cost and latency that continuous practices exist to eliminate. This paper proposes a framework that makes hardware-in-the-loop validation a first-class, automated stage of a CI/CD pipeline, specified concretely for C#-based orchestration within Azure DevOps and targeted at embedded software for IoT and vehicle systems. The framework defines a four-ring test architecture, static and unit rings on build agents, software-in-the-loop rings against simulation, and hardware-in-the-loop rings against instrumented device farms, with ring promotion governed by explicit gates; an orchestration layer, implemented as C# services, that manages rig inventory, device provisioning and flashing, test scheduling with priority and quarantine policies, signal-level stimulus and measurement, and artifact capture; and pipeline integration expressed as pipelines-as-code with environment checks, approval gates, and traceable promotion from commit to release candidate. The framework treats the known pathologies of continuous embedded testing, scarce rigs, long test times, flaky hardware, and version skew between software, firmware, and rig configuration, as design inputs with named mechanisms. An evaluation plan measures defect-detection shift, pipeline latency, rig utilization, and flakiness containment. Applications, limitations, and adoption pathways are analyzed.

KEYWORDS: Hardware-in-the-loop (HIL), CI/CD pipelines, Embedded systems validation, IoT firmware testing, Automotive software, Azure DevOps, C# orchestration, Cyber-physical systems testing

1. INTRODUCTION

The case for continuous practices is settled in general software engineering. Continuous integration reduces integration risk by building and testing every change (Duvall et al., 2007; Hilton et al., 2016), continuous delivery extends the discipline through automated promotion to releasable states (Humble & Farley, 2010; Chen, 2015), the research literature has mapped the practices, their variants, and their measured effects (Ståhl & Bosch, 2014; Shahin et al., 2017; Fitzgerald & Stol, 2017), and organizational studies associate delivery performance with these capabilities at scale (Forsgren et al., 2018), within the broader DevOps movement that fuses development and operations responsibilities (Bass et al., 2015; Ebert et al., 2016). Practice-facing studies of pipeline strategy, tool selection, and version control governance in packaged-platform ecosystems show the same automation ladder being rebuilt wherever delivery is industrialized, with environment promotion, branch policy, and reviewed change history recurring as the decisive variables rather than the particular vendor (Badmus, Dosunmu, & Ozowara, 2018; Badmus, Dosunmu, Ozowara, & Anunagba, 2020; Badmus, Jooda, Ozowara, & Anunagba, 2022).

Embedded software strains this template at its foundation. Vehicle and IoT software is cyber-physical: its correctness is defined against sensors, actuators, buses, and timing, properties of the composed hardware-software system rather than of code in isolation (Lee, 2008; Broy, 2006; Pretschner et al., 2007), the same coupling that makes substation, grid, and industrial control estates behave as composed physical systems rather than as software that happens to run on equipment (Shittu, Olagunju, et al., 2024; Ojukwu, Oyesiji, & Nwakamma, 2023). The automotive field answered decades ago with hardware-in-the-loop simulation, executing the real controller against simulated plant and signal environments with

production interfaces, first for engine control (Isermann et al., 1999) and then as a standard development instrument whose history and breadth the survey literature records (Mihalič et al., 2022; Fathy et al., 2006), complemented by model-based testing that generates and assesses tests from behavioral models (Bringmann & Krämer, 2008; Utting et al., 2012). Safety standardization institutionalizes the expectation that such verification is systematic and traceable (International Organization for Standardization, 2018).

The two traditions meet awkwardly in practice. Studies of continuous integration in software-intensive embedded organizations report the recurring impediments directly: test environments that are scarce and contended, hardware-dependent tests too slow for every commit, flakiness rooted in physical variability, and complex version coupling among application software, firmware, and test harness (Mårtensson et al., 2016; Shahin et al., 2017). The result in many organizations is a bifurcated process, fast pipelines that never touch hardware and hardware validation that never joins the pipeline, reproducing exactly the late-integration failure mode that continuous practice was designed to remove (Duvall et al., 2007; Humble & Farley, 2010). Meanwhile the delivery stakes rise: connected devices and vehicles ship software continuously into the field, over-the-air update capability makes release cadence a product property, and update integrity is itself a studied engineering problem (Halder et al., 2020; Al-Fuqaha et al., 2015; Atzori et al., 2010), against fleets whose energy, connectivity, and reliability budgets make defective updates expensive to remediate once distributed (Asiedu & Quainoo, 2024; Jimoh et al., 2023).

This paper proposes a framework that integrates hardware-in-the-loop validation into the delivery pipeline as an engineered, automated stage rather than an adjacent manual practice, specified for a concrete and widely deployed toolchain: C# as the orchestration implementation language and Azure DevOps as the pipeline platform. The contribution is architectural: a ring model that spends scarce hardware only where cheaper rings cannot answer, an orchestration layer that turns rigs into schedulable, observable infrastructure, pipeline integration that makes hardware results first-class quality signals with gates and traceability, and explicit mechanisms for the flakiness, capacity, and version-skew pathologies the empirical literature documents. Section 2 reviews background; Section 3 states requirements and principles; Sections 4 through 8 specify the framework; Section 9 sets out the evaluation plan; Sections 10 through 11 treat applications and limitations; Section 12 concludes.

2. BACKGROUND AND RELATED WORK

The continuous practices literature supplies the process substrate and its evidence. Foundational treatments define the practices and their automation ladder (Duvall et al., 2007; Humble & Farley, 2010), empirical work characterizes adoption patterns, costs, and benefits in open source and industry (Hilton et al., 2016; Ståhl & Bosch, 2014), systematic reviews organize the approaches, tools, and reported challenges (Shahin et al., 2017), and the continuous software engineering agenda situates integration and delivery within a larger continuum of continuous activities (Fitzgerald & Stol, 2017). Performance research links these capabilities to organizational outcomes (Forsgren et al., 2018), and the DevOps literature frames the operational side, environments, deployment, and monitoring as shared engineering concerns (Bass et al., 2015; Ebert et al., 2016). Within this substrate, two testing-at-scale findings shape the framework directly: flaky tests are endemic and must be managed as a category with detection and quarantine machinery rather than wished away (Luo et al., 2014; Memon et al., 2017), and regression test selection and prioritization are the levers by which slow suites coexist with fast pipelines (Yoo & Harman, 2012). The same substrate has absorbed machine-assisted testing, where learning-based techniques are applied to case generation, defect prediction, and automated regression control as extensions of the selection apparatus rather than replacements for it (Borah et al., 2022; Mbonu, Aliliele, et al., 2021), and where secure software engineering research attaches automated threat detection to the same pipelines (Odejobi et al., 2025). A parallel line in security engineering has converged independently on continuous validation, asserting control effectiveness through scheduled automated exercise against the live estate rather than through periodic assessment, which is structurally the argument this framework makes for hardware validation (Dosunmu & Ogundele, 2024a, 2024b).

The embedded verification literature supplies the validation substrate. Automotive software engineering roadmaps establish the domain's scale, heterogeneity, and safety coupling (Broy, 2006; Pretschner et al., 2007), and hardware-in-the-loop simulation is its characteristic instrument: real controllers exercised against real-time plant models through production electrical interfaces, with fidelity and coverage trade-offs studied across decades of application (Isermann et al., 1999; Fathy et al., 2006; Mihalič et al., 2022). Model-based testing contributes systematic stimulus generation and assessment from behavioral models (Bringmann & Krämer, 2008; Utting et al., 2012), general testing theory contributes the oracle problem and adequacy measurement that bound what any suite certifies (Barr et al., 2015; Ammann & Offutt, 2016; Jia & Harman, 2011), and fault injection contributes the dependability-assessment techniques that hardware rings are uniquely positioned to host (Hsueh et al., 1997; Natella et al., 2016). Simulation environments extend the pre-hardware rings, with high-fidelity simulators demonstrating what software-in-the-loop can absorb before hardware is spent (Dosovitskiy et al., 2017), and digital twin research generalizes the plant-model asset into a maintained engineering artifact (Tao et al., 2019), a generalization now demonstrated across substations, renewable assets, and process equipment where the twin is operated continuously against live telemetry rather than built once for design (Shittu, Shittu, et al., 2023; Shittu & Shittu, 2024; Olodo et al., 2025; Komi & Adeniji, 2023). On the pipeline side of this literature, infrastructure-as-code research supplies the practice by which environments, including test environments, become versioned, reviewable artifacts (Rahman et al., 2019), with governance and audit-trail models showing what disciplined configuration change control requires in regulated settings (Mbonu, Iwuanyanwu, et al., 2020; Oshoba, Hammed, & Odejobi, 2020), and the embedded-CI empirical strand names the integration problems this framework treats as requirements (Mårtensson et al., 2016). Service management research contributes the complementary vocabulary for treating a shared, contended estate as a service with owners, catalogues, request queues, and measured delivery rather than as equipment (Ladapo, Jooda, et al., 2023; Ladapo, Dosunmu, et al., 2023; Oteri & Edivri, 2026). The gap the framework addresses is the specified join: the literatures are mature separately, while the engineering of hardware-in-the-loop as a pipeline stage, scheduled, gated, observable, and versioned like any other, remains under-specified.

3. REQUIREMENTS AND DESIGN PRINCIPLES

Six requirements, drawn from the documented impediments, govern the design. Hardware economy: rig time is the scarce resource, so the architecture must answer every question at the cheapest ring capable of answering it and promote to hardware only what hardware alone can

decide, the same allocation logic that capacity planning and resource placement research applies to any constrained shared estate (Mårtensson et al., 2016; Yoo & Harman, 2012; Edivri & Oteri, 2022; Ahmed & Odejobi, 2018). Bounded latency with unbounded assurance: every commit receives fast feedback from early rings, while deeper rings run on selective and scheduled triggers, so pipeline latency and validation depth are decoupled by design rather than traded ad hoc (Duvall et al., 2007; Memon et al., 2017). Flakiness containment: hardware variability is a fact, so the framework distinguishes product failures from environment failures structurally, with retry, quarantine, and rig-health machinery whose decisions are logged and auditable (Luo et al., 2014). Version coherence: a test verdict is meaningful only against a recorded tuple of application build, firmware, plant model, rig configuration, and test suite versions, so the framework versions all five and stamps every result with the tuple, extending to physical configuration the reviewed-change discipline that version control governance and configuration audit trails already impose on software (Rahman et al., 2019; Humble & Farley, 2010; Badmus, Jooda, et al., 2022; Oshoba, Hammed, & Odejobi, 2020). Traceability: results, artifacts, and promotion decisions must be traceable from requirement to release in the manner safety processes expect, with the classification and regulatory-traceability mechanisms enterprise data governance has formalized supplying a usable model for evidence linkage (International Organization for Standardization, 2018; Pretschner et al., 2007; Aliliele et al., 2024). And platform realism: the framework is specified against a concrete toolchain, C# orchestration services and Azure DevOps pipelines, environments, and gates, because integration engineering lives in specifics, as comparative studies of delivery tooling in enterprise contexts consistently show (Badmus, Dosunmu, Ozowara, & Anunagba, 2020), while the architecture remains portable in its concepts.

4. FRAMEWORK OVERVIEW

The framework composes three planes. The test plane is the four-ring architecture of Section 5, a severity-ordered sequence from static analysis to hardware-in-the-loop, each ring with declared entry criteria, verdict semantics, and cost class. The orchestration plane, Section 6, is a set of C# services that own the physical estate: rig and device inventory, provisioning and flashing, scheduling, signal-level execution, and artifact capture, exposing the estate to pipelines as a capacity-managed test service rather than a collection of bench setups, in the sense service assurance research gives to a service with declared capacity, reliability targets, and consumers (Oteri & Edivri, 2026; Ladapo, Jooda, et al., 2023). The delivery plane, Section 7, is the Azure DevOps integration: pipelines-as-code that invoke rings as stages, environment and approval gates that consume ring verdicts, and artifact and traceability flows that carry the version tuple from commit to release candidate. Cutting across all three, the quality-signal model of Section 8 defines what a verdict is, how flake adjudication modifies it, and how release governance consumes it. The framework's unit of delivery is the validated candidate: a build whose recorded tuple has passed the rings its change class requires, with evidence attached.

5. THE FOUR-RING TEST ARCHITECTURE

Ring zero is static: compilation for all targets, static analysis, and interface contract checks, executed on hosted agents for every commit, the conventional base of continuous integration (Duvall et al., 2007). Ring one is unit and component: hardware-independent logic exercised with the platform's test frameworks, with hardware access abstracted behind interfaces so that the ring runs anywhere, and with mutation-informed adequacy tracking so suite strength is measured rather than presumed (Ammann & Offutt, 2016; Jia & Harman, 2011), supported where useful by the automated regression control techniques that keep fast suites current as the codebase moves (Mbonu, Aliliele, et al., 2021). Ring two is software-in-the-loop: the application, and where toolchains permit the target binary under instruction-set simulation, executed against versioned plant models and simulated signal environments, absorbing the behavioral and integration questions that need dynamics but not electrons, the layer high-fidelity simulation has shown can carry substantial validation load (Dosovitskiy et al., 2017; Tao et al., 2019; Utting et al., 2012), and the layer at which network-scale reliability questions for large device populations are answerable by simulation rather than by deployment (Asiedu & Quainoo, 2023b), with the plant models themselves maintained as tiered digital twins whose fidelity is matched to the question asked (Komi & Adeniji, 2023; Shittu, Shittu, et al., 2023; Olodo et al., 2025). Ring three is hardware-in-the-loop: production or production-representative devices on instrumented rigs, real buses and electrical interfaces, real-time plant models, and physical or emulated peripherals, hosting the test classes that justify its cost, timing and latency verification, bus-level integration, power and brown-out behavior, sensor and actuator path validation, and fault injection at the physical boundary (Isermann et al., 1999; Mihalič et al., 2022; Hsueh et al., 1997; Natella et al., 2016). Power-path behavior deserves particular emphasis for energy-constrained IoT classes, where intermittent supply makes correctness across power interruption a first-order product property rather than an edge case (Asiedu & Quainoo, 2025; Asiedu, Quainoo, & Asiedu, 2026). For wireless IoT products, radio-path characterization joins these classes, since the integration of contemporary system-on-chip transceivers makes antenna-port metrics aggregate behaviors of coupled RF, analog, and digital blocks that require protocol-specific test methodologies of their own (Quainoo & Ogundapo, 2026b), with impedance matching, channel-dependent throughput, and multi-radio coexistence supplying concrete measurable criteria for the ring's pass conditions (Quainoo, Ogundapo, & Asiedu, 2024, 2025a; Quainoo & Ogundapo, 2026a).

Promotion between rings is governed, not habitual. Every change carries a computed change class, from touched components and declared risk, that maps to a required ring depth and a selected subset within deep rings, applying regression selection where whole-suite execution would break latency budgets (Yoo & Harman, 2012). Ring verdicts are ring-scoped: a ring-two pass asserts behavior against models at recorded versions, never hardware correctness, and the oracle limits of each ring travel with its verdict, in the discipline the testing literature's oracle analysis demands (Barr et al., 2015). Nightly and pre-release schedules run the deep rings broadly regardless of change class, so selective per-commit depth is backstopped by periodic breadth (Memon et al., 2017).

6. THE ORCHESTRATION LAYER

The orchestration layer is specified as C# services because the estate it manages is stateful, concurrent, and long-lived, properties served well by a typed, asynchronous service platform, and because the implementation language claim is part of the framework's platform realism. The inventory service models rigs, devices, fixtures, and their capabilities as a typed catalogue with health state, calibration status, and configuration version, the estate's source of truth, in the tradition of reliability and condition-monitoring practice that treats equipment state as recorded data rather than

technician knowledge (Gideon et al., 2019; Adaramola et al., 2018). The provisioning service owns device preparation: firmware flashing, configuration injection, and post-flash verification, recording the resulting device state into the version tuple of every subsequent result. The scheduling service allocates test requests to rigs by capability matching, priority, and fairness across teams, with preemption classes for release-blocking runs and reservations for scheduled breadth passes, turning contention, the first documented impediment, into managed queueing with observable wait metrics (Mårtensson et al., 2016), a problem whose formal shape is shared with constraint-based allocation, placement, and predictive scaling of shared computing capacity, and whose demand side is addressed by capacity forecasting rather than by reactive expansion (Ahmed, Odejobi, & Oshoba, 2019, 2020; Ahmed & Odejobi, 2018; Odejobi & Ahmed, 2018; Edivri & Oteri, 2022). The execution service drives tests at signal level: stimulus generation against the plant model, measurement capture, pass criteria evaluation, and time-synchronized logging of buses, signals, and device output, with model-based stimulus supported where behavioral models exist (Bringmann & Krämer, 2008). The service's instrument-facing design draws on the test-automation practice of wireless hardware engineering, where programmable instrument control, managed instrument state, calibration traceability, and engineered error recovery are the documented disciplines separating production automation from ad hoc scripting (Quainoo, Ogundapo, & Asiedu, 2025b), and on instrumented condition-monitoring practice, where sensor acquisition and analysis are built as sustained measurement systems rather than one-off benches (Omoegun et al., 2023). The artifact service captures everything a verdict rests on, waveforms, logs, configuration snapshots, and environmental readings, content-addressed and linked to the result record.

Two cross-cutting mechanisms complete the layer. Rig health is actively managed: known-answer self-tests run between allocations, drift and failure statistics accrue per rig, and rigs exceeding health thresholds are automatically drained from scheduling, so environment degradation surfaces as capacity events rather than as spurious product failures (Luo et al., 2014). This is the predictive maintenance transition applied to test infrastructure, replacing repair-on-failure with sensed condition, statistical degradation models, and scheduled intervention, an approach whose returns in equipment-intensive settings are well documented and whose economics improve as the estate grows (Sunday et al., 2020; Sunday & Omoegun, 2022; Mayo et al., 2021; Oyeleke et al., 2026; Olodo et al., 2026a, 2026b). And the whole layer is configuration-as-code: rig definitions, plant model bindings, and suite mappings live in versioned repositories, reviewed and promoted like software, extending infrastructure-as-code practice to the physical estate with the governance controls that reviewed configuration change requires (Rahman et al., 2019; Mbonu, Iwuanyanwu, et al., 2020; Badmus, Jooda, et al., 2022; Oshoba, Hammed, & Odejobi, 2020).

7. AZURE DEVOPS INTEGRATION

The delivery plane expresses the framework in the platform's native constructs. Pipelines are defined as code in the repository, with ring zero and ring one as standard build-and-test stages on hosted or self-hosted agents, and rings two and three as deployment-style stages targeting registered environments that represent simulation clusters and rig pools. Environment checks and approvals implement the gates: automated checks consume ring verdicts and health signals from the orchestration layer's API, and human approvals are reserved for the promotions where judgment is required, release-candidate designation foremost, keeping the automation ladder consistent with continuous delivery practice (Humble & Farley, 2010; Chen, 2015) and with the human-in-the-loop design principle that reserves human attention for decisions automation cannot adjudicate rather than for those it can (Ladapo, Dosunmu, et al., 2022; Ladapo, Jooda, et al., 2024). Comparative evaluations of delivery tooling in enterprise platform contexts inform the division of labor assumed here, where the platform supplies orchestration and evidence while domain tooling supplies the specialized execution (Badmus, Dosunmu, & Ozowara, 2018; Badmus, Dosunmu, Ozowara, & Anunagba, 2020). The orchestration layer is invoked from pipeline stages through service connections, submitting test requests with the change class and receiving structured verdicts; agent-side steps remain thin, so pipeline latency reflects queueing and execution rather than harness logic. Those service connections are a security boundary as much as an integration one, since pipeline identities hold standing access to physical devices, which is precisely the case that federated identity governance and identity-centric zero trust models are designed for (Oshoba, Hammed, & Odejobi, 2019; Ogbole, Okoruwa, Fadayomi, et al., 2021; Ahmed, Adegbite, & Adebayo, 2021).

Traceability rides the platform's artifact and work item graph. Builds carry the version tuple as immutable metadata; test results publish into the platform's result stores with links to orchestration artifacts; and work items link requirements to the tests that verify them, so the requirement-to-evidence chain that safety processes expect is navigable within the toolchain rather than reconstructed for audits (International Organization for Standardization, 2018; Pretschner et al., 2007; Aliliele et al., 2024; Oshoba, Hammed, & Odejobi, 2020). Release pipelines consume the same signals for promotion toward field deployment, where over-the-air delivery makes the pipeline's end state a fleet state and update integrity requirements attach (Halder et al., 2020). The integration's design position is deliberate: the platform provides orchestration of stages, gates, and evidence, the framework provides orchestration of hardware, and the boundary between them is a typed API, which is what makes the architecture portable beneath its concrete specification, following the general lesson that portability across execution hosts is bought by a stable interface contract rather than by avoiding platform commitment (Ojukwu, Oyesiji, & Nwakamma, 2025). Where the estate is already governed as a service, the same boundary lets rig requests, incidents, and change records flow into existing service management systems instead of a parallel toolchain (Ladapo, Jooda, et al., 2023).

8. QUALITY SIGNALS AND RELEASE GOVERNANCE

A verdict in this framework is a structured claim: outcome, ring, version tuple, artifacts, and adjudication state. Flake adjudication is the machinery that keeps hardware variability from corrupting the signal: failures trigger bounded automatic retry under identical configuration; discordant outcomes route to quarantine, where the test executes on alternate rigs to separate product from environment; and quarantine outcomes update both test and rig reliability statistics, with persistently unstable tests demoted from gating status until repaired, the management pattern large-scale continuous testing has converged on (Luo et al., 2014; Memon et al., 2017). Gating consumes adjudicated verdicts only, and gate policies are declared per branch and change class: fast rings gate merges, deep rings gate release candidacy, and breadth passes gate release, with every gate decision recorded against the evidence it consumed, in the release-gating pattern that automated evaluation research has formalized for judgments whose reliability must itself be characterized before it is trusted to block (Nwakamma, Ojukwu, & Oyesiji, 2024). The same as-code discipline

extends to regulatory and organizational compliance obligations, which compliance-as-code practice formalizes as machine-readable, version-controlled policy validated within the pipelines themselves (Oshoba, Ahmed, & Odejobi, 2023), and which control automation and governance research extends toward auditable, defensible decision records at the organizational level (Mbonu, Aliliele, et al., 2022; Ominyi et al., 2026).

Governance closes the loop with measurement. The framework's own health is tracked as a small set of registered indicators, defect-detection ring distribution, where in the ring sequence defects are first caught, pipeline latency percentiles per change class, rig utilization and queue delay, flake and quarantine rates, and escaped-defect counts from field and late-stage discovery, chosen to make the framework's claims falsifiable in operation and aligned with the measurement orientation of delivery-performance research (Forsgren et al., 2018; Ståhl & Bosch, 2014). Indicator design follows the practice established for service delivery and operational monitoring, where a small registered set tied to decisions outperforms broad dashboards that report everything and settle nothing (Edivri & Oteri, 2024; Ogbole, Okoruwa, Babatope, & Oyewole, 2021; Tonoyan et al., 2024; Oteri & Edivri, 2026). Release governance is thereby evidence-based twice over: candidates are promoted on adjudicated verdicts, and the process that produces verdicts is itself monitored against the pathologies it exists to control, the same reflexive posture continuous security validation adopts when it measures the assurance apparatus rather than assuming it (Dosunmu & Ogundele, 2024b).

9. EVALUATION PLAN

The framework's evaluation is designed as a staged engineering study. Phase one is reference implementation and bench validation: the orchestration services and pipeline definitions are implemented against a reference estate of representative IoT and vehicle-class devices, and the mechanisms are verified in isolation, provisioning correctness, scheduler behavior under contention, health-drain triggering, retry and quarantine flows, and version-tuple integrity across the chain, with fault injection into the estate itself, induced rig faults, corrupted configurations, exercising the containment machinery (Hsueh et al., 1997; Natella et al., 2016), and with failure analysis conducted in the systematic manner reliability engineering prescribes so that estate faults are classified rather than merely observed (Gideon et al., 2019). Structured acceptance validation with the engineers who will operate the estate is included in this phase, since a validation framework that passes its own tests but fails its users has not been validated (Eyetsemitan, Ambali, et al., 2023b).

Phase two is comparative pipeline measurement on seeded workloads. A defect corpus, seeded product faults spanning logic, integration, timing, and hardware-interface classes, is injected into a reference codebase's history, and the framework's ring architecture is measured against two baselines, a software-only pipeline and a manual-HIL process model, on detection ring distribution, time-to-detection per defect class, hardware hours consumed per detected defect, and pipeline latency per change class, with mutation-based adequacy tracking validating that suite strength differences do not confound the comparison (Jia & Harman, 2011; Yoo & Harman, 2012). Flakiness containment is measured by injecting environment instability and scoring signal corruption, gating decisions altered by environment rather than product, with and without adjudication.

Phase three is field study: deployment in one or more embedded product organizations, with the Section 8 indicators collected over successive releases and practitioner experience gathered against the documented impediment list, contention, latency, flakiness, and version skew, that motivated the design (Mårtensson et al., 2016; Hilton et al., 2016; Edivri & Oteri, 2024). Because adoption is itself an intervention, the study records the organizational change alongside the technical one, following change management research that treats staged adoption, role clarity, and demonstrated early value as measurable variables rather than background conditions (Eyetsemitan, Ambali, et al., 2023a). The plan's endpoints are stated in the framework's own terms: leftward shift of detection ring distribution at bounded pipeline latency, hardware economy relative to baselines, and containment of environment-caused signal corruption, with organizational outcomes interpreted through, not substituted for, these mechanism-level measures (Forsgren et al., 2018).

10. ANTICIPATED APPLICATIONS

The framework's primary application is embedded product delivery where hardware validation currently sits outside the pipeline: vehicle electronic control units and their subsystem software, where the HIL tradition is strongest and the safety traceability requirements are explicit (Broy, 2006; International Organization for Standardization, 2018), and IoT device firmware across industrial and consumer classes, where fleet scale and over-the-air cadence raise the cost of late discovery (Al-Fuqaha et al., 2015; Halder et al., 2020). Energy-constrained and intermittently powered device classes are a natural early target, since their failure modes concentrate precisely in the power, timing, and radio behaviors that only ring three observes (Asiedu & Quainoo, 2023a; Asiedu, Quainoo, & Asiedu, 2025), as are on-device intelligence workloads whose resource envelopes must be validated on the silicon that will run them rather than on development hosts (Nwakamma, Ojukwu, & Oyesiji, 2025). Secondary applications follow from the architecture's parts: the orchestration layer alone modernizes existing rig estates into schedulable services even before full pipeline integration; the ring model with change-class promotion transfers to any cyber-physical domain with layered fidelity environments, robotics, energy systems, and semiconductor process equipment included (Lee, 2008; Tao et al., 2019; Akanbi & Sunday, 2024, 2025; Akanbi, Ganiu, & Sunday, 2025; Shittu, Adeniji, et al., 2026; Shittu & Shittu, 2024; Olodo et al., 2026b), and it extends to converged IT and operational technology estates where software changes reach physical process equipment through the same delivery path (Adegbite et al., 2022; Ojukwu, Oyesiji, & Nwakamma, 2023); and the seeded-defect evaluation methodology of Section 9 is itself reusable as a benchmarking instrument for embedded pipeline designs. For engineering organizations mid-transition, the framework also supplies an adoption pathway, rings adopted left to right, estate onboarded rig by rig, gates tightened as signal quality is demonstrated, consistent with the incremental adoption patterns the continuous-practices literature records (Shahin et al., 2017; Fitzgerald & Stol, 2017) and with the staged, procedure-anchored transformation approaches documented for resource-constrained and multi-program organizations (Ambali et al., 2021; Eyetsemitan, Oyeleye, et al., 2022; Eyetsemitan, Ambali, et al., 2023a; Ladapo, Dosunmu, et al., 2025; Ladapo, Jooda, et al., 2022), where governance of the transition program is as decisive as the engineering (Amayo et al., 2023, 2024).

11. LIMITATIONS

The framework's claims are bounded honestly. Ring verdicts inherit oracle limits: hardware-in-the-loop exercises the controller against modeled plants, and fidelity gaps between model and world remain a validity risk that no pipeline mechanism removes, only documents and manages through model versioning and periodic vehicle-level or field validation outside the framework's scope (Fathy et al., 2006; Mihalič et al., 2022; Barr et al., 2015), a gap tiered digital twin practice makes explicit by declaring what each fidelity level does and does not license (Komi & Adeniji, 2023). Capacity economics constrain applicability: the architecture reduces hardware demand per assurance unit but presupposes an instrumented estate, and organizations without one face the rig investment before the framework's returns, an investment whose maintenance and utilization economics are themselves nontrivial in equipment-intensive settings (Oyeleke et al., 2026; Olodo et al., 2026a). The concrete platform specification is a double-edged commitment: it makes the integration engineering real, and it ties the reference design to one vendor's constructs, with portability resting on the orchestration API boundary rather than demonstrated multi-platform implementations (Ojukwu, Oyesiji, & Nwakamma, 2025). Flake adjudication reduces but cannot eliminate misclassification, and its retry economics consume the same scarce hardware it protects; nor does recording gate decisions make them defensible on its own, since the quality of the evidence and of the policy that consumes it remains a governance question (Ominyi et al., 2026). Finally, the evaluation plan's comparative phases rest on seeded defect corpora whose representativeness bounds external validity, a standard limitation of testing research the field-study phase mitigates but does not remove (Ammann & Offutt, 2016; Mårtensson et al., 2016).

12. CONCLUSION

This paper has proposed a framework that makes hardware-in-the-loop validation a governed, automated stage of continuous delivery for IoT and vehicle embedded software, specified for C# orchestration within Azure DevOps. Its architecture answers the documented impediments of continuous embedded testing with named mechanisms: a four-ring test model that spends hardware only where hardware decides, an orchestration layer that turns rig estates into versioned, health-managed, schedulable infrastructure, pipeline integration that carries verdicts, artifacts, and the full version tuple through native gates and traceability, and a quality-signal model whose flake adjudication keeps physical variability from corrupting release decisions. The evaluation plan renders the framework's promises falsifiable, detection shifted earlier at bounded latency, hardware economized per defect found, environment noise contained, before organizational claims are made.

The broader position is that the divide between fast pipelines and hardware truth is an engineering artifact, not a necessity: with the estate treated as code-configured infrastructure, tests treated as capacity-scheduled workloads, and verdicts treated as adjudicated evidence, the delivery discipline that transformed general software extends to the systems where software meets the physical world, which is precisely where its assurances are worth most.

REFERENCES

1) Adaramola, T. S., Fadero, S., & Gideon, E. N. (2018). Predictive maintenance and condition monitoring in critical power and energy infrastructure. Iconic Research and Engineering Journals, 2(5), 454-477.
2) Adegbite, M. P., Adebayo, A., & Ahmed, M. O. (2022). A security architecture model for IT and OT convergence in regulated energy networks: Design principles and governance alignment. International Journal of Computer Science and Mathematical Theory, 8(2), 81-132.
3) Ahmed, K. S., & Odejobi, O. D. (2018). Resource allocation model for energy-efficient virtual machine placement in data centers. IRE Journals, 2(3), 1-10.
4) Ahmed, K. S., Odejobi, O. D., & Oshoba, T. O. (2019). Algorithmic model for constraint satisfaction in cloud network resource allocation. IRE Journals, 2(12), 516-532.
5) Ahmed, K. S., Odejobi, O. D., & Oshoba, T. O. (2020). Predictive model for cloud resource scaling using machine learning techniques. Journal of Frontiers in Multidisciplinary Research, 1(1), 173-183.
6) Ahmed, M. O., Adegbite, M. P., & Adebayo, A. (2021). Zero trust architecture for operational technology in North American critical infrastructure: A framework for implementation and resilience optimization. International Journal of Engineering and Modern Technology, 7(1), 66-113.
7) Akanbi, O., & Sunday, E. A. (2024). An integrated path-planning and slotting optimization model for AMR-enabled high-density warehousing. International Journal of Advanced Multidisciplinary Research and Studies, 4(6), 3226-3243.
8) Akanbi, O., & Sunday, E. A. (2025). A systematic review of AI-driven autonomous mobile robots (AMR) in scalable micro-fulfillment centers. International Journal of Advanced Multidisciplinary Research and Studies, 5(6), 2363-2379.
9) Akanbi, O., Ganiu, O. S., & Sunday, E. A. (2025). A multi-agent AI framework for swarm intelligence in autonomous mobile robot (AMR) fleet coordination. International Journal of Scientific Research in Humanities and Social Sciences, 2(1), 81-109.
10) Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of Things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys & Tutorials, 17(4), 2347-2376.
11) Aliliele, C., Mbonu, I. S., & Iwuanyanwu, U. (2024). A conceptual framework for enterprise data sensitivity classification and regulatory traceability mechanisms. International Journal of Advanced Multidisciplinary Research and Studies, 4(6), 3103-3124.
12) Amayo, E. B., Owulade, O. A., & Isi, L. R. (2023). Optimizing project governance in multinational infrastructure projects: Insights from General Electric's global operations. International Journal of Multidisciplinary Research and Growth Evaluation, 4(1), 975-983.
13) Amayo, E. B., Owulade, O. A., & Isi, L. R. (2024). Best practices for managing data center lifecycle projects: Ensuring security, efficiency, and compliance in U.S. enterprises. Iconic Research and Engineering Journals, 7(7), 598-617.
14) Ambali, K. B., Eyetsemitan, R. A., Oyeleye, A. O., & Fadayomi, O. (2021). Lean Six Sigma for small enterprises: A systematic review and Lite-DMAIC adaptation framework for resource-constrained organizations. Iconic Research and Engineering Journals, 5(5), 562-583.

15) Ammann, P., & Offutt, J. (2016). Introduction to software testing (2nd ed.). Cambridge University Press.
16) Asiedu, W., & Quainoo, R. (2023a). How far can energy harvesting take us? A systematic review of radio frequency strategies for energy autonomous sensing. Shodhshauryam, International Scientific Refereed Research Journal, 6(1), 448-466.
17) Asiedu, W., & Quainoo, R. (2023b). Toward maintenance free wireless infrastructure: Simulating performance and reliability in large scale intermittently powered IoT networks. Gyanshauryam, International Scientific Refereed Research Journal, 6(1), 489-510.
18) Asiedu, W., & Quainoo, R. (2024). Rethinking energy, reliability, and latency trade-offs in green communication for next generation IoT. International Journal of Multidisciplinary Research and Growth Evaluation, 5(6), 1987-1994.
19) Asiedu, W., & Quainoo, R. (2025). GleanCast: Epoch-aligned reliable broadcast in batteryless intermittent sensor networks. International Journal of Engineering and Modern Technology, 11(12), 205-218.
20) Asiedu, W., Quainoo, R., & Asiedu, A. (2025). A conceptual framework for characterizing fundamental energy, reliability, and latency trade-offs in green communication paradigms for next-generation IoT. International Journal of Advanced Multidisciplinary Research and Studies, 5(6), 2447-2453.
21) Asiedu, W., Quainoo, R., & Asiedu, A. (2026). Predictive power management for intermittent IoT devices using lightweight machine learning under uncertain energy harvest. Gulf Journal of Engineering and Technology, 2(4), 108-118.
22) Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. Computer Networks, 54(15), 2787-2805.
23) Badmus, O., Dosunmu, A. A., & Ozowara, D. E. (2018). A systematic review of CI/CD pipeline strategies in Salesforce DevOps: Tools, practices, and deployment outcomes. Iconic Research and Engineering Journals, 2(6).
24) Badmus, O., Dosunmu, A. A., Ozowara, D. E., & Anunagba, C. O. (2020). A comparative framework for Salesforce DevOps tooling: Evaluating Copado, Flosum, and Salesforce DX across enterprise deployment contexts. International Journal of Multidisciplinary Research and Growth Evaluation, 1(5), 1021-1031.
25) Badmus, O., Jooda, D., Ozowara, D. E., & Anunagba, C. O. (2022). A framework for Git-based version control and code review governance in Salesforce development teams. Shodhshauryam, International Scientific Refereed Research Journal, 5(2), 473-493.
26) Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). The oracle problem in software testing: A survey. IEEE Transactions on Software Engineering, 41(5), 507-525.
27) Bass, L., Weber, I., & Zhu, L. (2015). DevOps: A software architect's perspective. Addison-Wesley.
28) Borah, S., Aliliele, K. C., Rakshit, S., & Vajjhala, N. R. (2022). Applications of artificial intelligence in software testing. In P. K. Mallick et al. (Eds.), Cognitive informatics and soft computing (Lecture Notes in Networks and Systems, Vol. 375, pp. 727-736). Springer.
29) Bringmann, E., & Krämer, A. (2008). Model-based testing of automotive systems. Proceedings of the International Conference on Software Testing, Verification, and Validation, 485-493.
30) Broy, M. (2006). Challenges in automotive software engineering. Proceedings of the International Conference on Software Engineering, 33-42.
31) Chen, L. (2015). Continuous delivery: Huge benefits, but challenges too. IEEE Software, 32(2), 50-54.
32) Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). CARLA: An open urban driving simulator. Proceedings of the Conference on Robot Learning, 1-16.
33) Dosunmu, A. A., & Ogundele, P. O. (2024a). Breach and attack simulation frameworks for continuous validation of enterprise security controls. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 10(3), 1100-1119.
34) Dosunmu, A. A., & Ogundele, P. O. (2024b). Enterprise scale continuous security validation models for regulated digital infrastructures. International Journal of Scientific Research in Humanities and Social Sciences, 1(2), 929-945.
35) Duvall, P. M., Matyas, S., & Glover, A. (2007). Continuous integration: Improving software quality and reducing risk. Addison-Wesley.
36) Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. IEEE Software, 33(3), 94-100.
37) Edivri, J., & Oteri, O. (2022). Predictive capacity planning and resource utilization forecasting models for multi-program and multi-stakeholder IT portfolios. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 8(4), 826-845.
38) Edivri, J., & Oteri, O. (2024). A conceptual KPI-driven decision and optimization framework for IT service delivery, portfolio performance, and adoption. Gyanshauryam, International Scientific Refereed Research Journal, 7(1), 167-187.
39) Eyetsemitan, R. A., Ambali, K. B., Oyeleye, A. O., & Fadayomi, O. (2023a). Change management in small business digital transformation: A systematic review and lean change adoption framework. Gyanshauryam, International Scientific Refereed Research Journal, 6(6), 521-550.
40) Eyetsemitan, R. A., Ambali, K. B., Oyeleye, A. O., & Fadayomi, O. (2023b). User acceptance testing in small business technology deployment: A structured validation framework for lean operational environments. International Journal of Multidisciplinary Research and Growth Evaluation, 4(6), 1512-1531.
41) Eyetsemitan, R. A., Oyeleye, A. O., Ambali, K. B., & Fadayomi, O. (2022). Standard operating procedures as strategic assets in small business operations: A systematic review and implementation framework. Gyanshauryam, International Scientific Refereed Research Journal, 5(2), 438-465.
42) Fathy, H. K., Filipi, Z. S., Hagena, J., & Stein, J. L. (2006). Review of hardware-in-the-loop simulation and its prospects in the automotive area. Proceedings of SPIE, 6228, 62280E.
43) Fitzgerald, B., & Stol, K.-J. (2017). Continuous software engineering: A roadmap and agenda. Journal of Systems and Software, 123, 176-189.
44) Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The science of lean software and DevOps. IT Revolution Press.

45) Gideon, E. N., Adaramola, T. S., & Fadero, S. (2019). Reliability engineering and failure analysis in complex engineering systems. Iconic Research and Engineering Journals, 2(11), 703-727.
46) Halder, S., Ghosal, A., & Conti, M. (2020). Secure over-the-air software updates in connected vehicles: A survey. Computer Networks, 178, 107343.
47) Hilton, M., Tunnell, T., Huang, K., Marinov, D., & Dig, D. (2016). Usage, costs, and benefits of continuous integration in open-source projects. Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, 426-437.
48) Hsueh, M.-C., Tsai, T. K., & Iyer, R. K. (1997). Fault injection techniques and tools. Computer, 30(4), 75-82.
49) Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley.
50) International Organization for Standardization. (2018). ISO 26262: Road vehicles - functional safety. ISO.
51) Isermann, R., Schaffnit, J., & Sinsel, S. (1999). Hardware-in-the-loop simulation for the design and testing of engine-control systems. Control Engineering Practice, 7(5), 643-653.
52) Jia, Y., & Harman, M. (2011). An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering, 37(5), 649-678.
53) Jimoh, H. O., Abolle-Okoyeagu, C. J., Ahmed, M. O., & Lawal, N. O. (2023). Advancing security in IoT-driven critical infrastructure: A focus on smart transportation system. American Journal of Engineering Research, 12(12), 33-46.
54) Komi, N. M., & Adeniji, I. O. (2023). A tiered digital twin that brings predictive diagnostics to resource-constrained renewable energy sites. International Journal of Engineering and Modern Technology, 9(3), 287-347.
55) Ladapo, O. O., Dosunmu, A. A., Jooda, D., & Abolaji, T. O. (2022). Human-in-the-loop machine learning: A state of the art. Journal of Frontiers in Multidisciplinary Research, 3(1), 656-669.
56) Ladapo, O. O., Dosunmu, A. A., Jooda, D., & Abolaji, T. O. (2023). An IT service management literature review: Challenges, benefits, opportunities, and implementation practices. International Journal of Advanced Multidisciplinary Research and Studies, 3(6), 2875-2889.
57) Ladapo, O. O., Dosunmu, A. A., Jooda, D., & Abolaji, T. O. (2025). Migration of applications and information systems to cloud computing infrastructure: Lessons from a South African retail bank. International Journal of Multidisciplinary Research and Growth Evaluation, 6(6), 1361-1375.
58) Ladapo, O. O., Jooda, D., Dosunmu, A. A., & Abolaji, T. O. (2022). Navigating digital transformation: Best practices for cloud migration strategies in the enterprise. Journal of Frontiers in Multidisciplinary Research, 3(1), 643-655.
59) Ladapo, O. O., Jooda, D., Dosunmu, A. A., & Abolaji, T. O. (2023). A comprehensive survey on ServiceNow for IT service management. International Journal of Multidisciplinary Research and Growth Evaluation, 4(6), 1532-1545.
60) Ladapo, O. O., Jooda, D., Dosunmu, A. A., & Abolaji, T. O. (2024). Keeping humans in the loop: Human-centered automated annotation with generative AI. International Journal of Multidisciplinary Futuristic Development, 5(1), 81-95.
61) Lee, E. A. (2008). Cyber physical systems: Design challenges. Proceedings of the IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 363-369.
62) Luo, Q., Hariri, F., Eloussi, L., & Marinov, D. (2014). An empirical analysis of flaky tests. Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 643-653.
63) Mårtensson, T., Ståhl, D., & Bosch, J. (2016). Continuous integration applied to software-intensive embedded systems - problems and experiences. Proceedings of the International Conference on Product-Focused Software Process Improvement, 448-457.
64) Mayo, W., Ogbole, J. I., Okoruwa, P. O., & Babatope, O. M. (2021). Designing an AI-predictive maintenance model for e-commerce systems using machine learning and cloud analytics. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(5), 416-440.
65) Mbonu, I. S., Aliliele, C., Iwuanyanwu, U., & Uzoka, E. (2021). Advances in artificial intelligence techniques for secure software testing and automated regression control mechanisms. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(5), 468-496.
66) Mbonu, I. S., Aliliele, C., Iwuanyanwu, U., & Uzoka, E. (2022). A conceptual framework for AI enabled IT general controls and SOX audit automation processes. Gyanshauryam, International Scientific Refereed Research Journal, 5(5), 384-414.
67) Mbonu, I. S., Iwuanyanwu, U., Aliliele, C., & Uzoka, E. (2020). Advances in infrastructure as code governance for secure terraform based enterprise cloud deployments. International Journal of Multidisciplinary Research and Growth Evaluation, 1(5), 811-828.
68) Memon, A., Gao, Z., Nguyen, B., et al. (2017). Taming Google-scale continuous testing. Proceedings of the International Conference on Software Engineering: Software Engineering in Practice Track, 233-242.
69) Mihalič, F., Truntič, M., & Hren, A. (2022). Hardware-in-the-loop simulations: A historical overview of engineering challenges. Electronics, 11(15), 2462.
70) Natella, R., Cotroneo, D., & Madeira, H. (2016). Assessing dependability with software fault injection: A survey. ACM Computing Surveys, 48(3), 44.
71) Nwakamma, S., Ojukwu, J., & Oyesiji, S. O. (2024). LLM-as-a-judge for automated model governance: A review of methods, biases, and release-gating practices. World Journal of Innovation and Modern Technology, 8(6), 185-216.
72) Nwakamma, S., Ojukwu, J., & Oyesiji, S. O. (2025). On-device multimodal small language models for privacy-preserving edge intelligence through quantization, pruning, distillation, and energy-efficient inference. World Journal of Innovation and Modern Technology, 9(12), 271-302.
73) Odejobi, O. D., & Ahmed, K. S. (2018). Performance evaluation model for multi-tenant Microsoft 365 deployments under high concurrency. IRE Journals, 1(11), 92-107.

74) Odejobi, O. D., Okonkwo, C. S., Ahiaeke Patrick, M. C., Okeke, O. T., & Mayo, W. (2025). AI-augmented secure software engineering: Leveraging deep learning for autonomous threat detection and mitigation. International Journal of Engineering and Modern Technology, 11(12), 101-121.
75) Ogbole, J. I., Okoruwa, P. O., Babatope, O. M., & Oyewole, T. (2021). Developing an integrated data visualization model for continuous business performance monitoring and optimization. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(5), 441-467.
76) Ogbole, J. I., Okoruwa, P. O., Fadayomi, O., Abolaji, T. O., Edivri, J., & Akeju, B. (2021). Conceptual model for identity-centric zero trust architecture in enterprise security governance. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 7(5), 393-415.
77) Ojukwu, J., Oyesiji, S. O., & Nwakamma, S. (2023). Cyber-physical ransomware defense in operational technology using physics-informed detection, digital twins, network telemetry, and resilient recovery. International Journal of Computer Science and Mathematical Theory, 9(5), 193-228.
78) Ojukwu, J., Oyesiji, S. O., & Nwakamma, S. (2025). WebAssembly component model and WASI for portable cloud-to-edge software: Language interoperability, security, performance, and container comparisons. International Journal of Computer Science and Mathematical Theory, 11(12), 232-263.
79) Olodo, A., Akanbi, O., & Adenuga, O. (2025). Conceptualizing a closed-loop digital twin for real-time plasma control in large-area magnetron sputtering systems. International Journal of Engineering and Modern Technology, 11(12), 205-227.
80) Olodo, A., Akanbi, O., & Adenuga, O. (2026a). Conceptualizing a lean-standardized predictive maintenance framework for high-vacuum deposition equipment. International Journal of Engineering and Modern Technology, 12(3), 295-322.
81) Olodo, A., Akanbi, O., & Adenuga, O. (2026b). Data-driven maintenance and reliability in high-volume semiconductor fabs: A review of Industry 4.0 methodologies. International Journal of Engineering and Modern Technology, 12(3), 323-348.
82) Ominyi, M., Anichukwueze, C. C., & Uzougbo, N. S. (2026). Conceptualizing enterprise AI governance functions as institutional standards for auditable and defensible risk management. Journal of Accounting and Financial Management, 12(7), 326-356.
83) Omoegun, G. O., Sunday, E. A., Essien, M. A., & Oluokun, O. A. (2023). Vibration-based condition monitoring of rotating machinery using LabVIEW. International Journal of Scientific Research in Civil Engineering, 7(6), 82-108.
84) Oshoba, T. O., Ahmed, K. S., & Odejobi, O. D. (2023). Compliance-as-code model for automated governance pipelines in hybrid cloud. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 10(1), 617-631.
85) Oshoba, T. O., Hammed, N. I., & Odejobi, O. D. (2019). Secure identity and access management model for distributed and federated systems. IRE Journals, 3(4), 550-567.
86) Oshoba, T. O., Hammed, N. I., & Odejobi, O. D. (2020). Blockchain-enabled compliance and audit trail model for cloud configuration management. Journal of Frontiers in Multidisciplinary Research, 1(1), 193-201.
87) Oteri, O., & Edivri, J. (2026). An operational reliability and service assurance framework for enterprise IT systems supporting large user populations. Gulf Journal of Engineering and Technology, 2(1), 1-19.
88) Oyeleke, A. V., Eze, F. C., & Asiedu, W. (2026). Reliability-centered maintenance strategies for minimizing downtime and maximizing performance in high-density GPU cluster environments. International Journal of Engineering Technology Research & Management, 10(7), 18-38.
89) Pretschner, A., Broy, M., Krüger, I. H., & Stauner, T. (2007). Software engineering for automotive systems: A roadmap. Proceedings of Future of Software Engineering, 55-71.
90) Quainoo, R., & Ogundapo, O. (2026a). A system-level power behavior model for Bluetooth and Wi-Fi coexistence in dual-mode wireless devices. International Journal of Computer Science and Mathematical Theory, 12(2), 298-358.
91) Quainoo, R., & Ogundapo, O. (2026b). Wireless system-on-chip performance in next-generation Internet of Things devices: A systematic review of integrated transceiver testing methodologies. World Journal of Innovation and Modern Technology, 10(5), 76-126.
92) Quainoo, R., Ogundapo, O., & Asiedu, W. A. (2024). Modeling the impact of impedance mismatch on wireless link performance: A framework for prototype development and system optimization. International Journal of Computer Science and Mathematical Theory, 10(2), 62-116.
93) Quainoo, R., Ogundapo, O., & Asiedu, W. A. (2025a). Predicting throughput degradation in Wi-Fi 6 networks under varying channel conditions: A physical layer performance model. International Journal of Engineering and Modern Technology, 11(12), 205-264.
94) Quainoo, R., Ogundapo, O., & Asiedu, W. A. (2025b). Test automation in wireless hardware engineering: A comprehensive review of scripting frameworks and instrument control strategies. International Journal of Engineering and Modern Technology, 11(10), 405-464.
95) Rahman, A., Mahdavi-Hezaveh, R., & Williams, L. (2019). A systematic mapping study of infrastructure as code research. Information and Software Technology, 108, 65-77.
96) Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. IEEE Access, 5, 3909-3943.
97) Shittu, H., & Shittu, M. A. (2024). AI powered digital twins for predictive maintenance and operational optimization of renewable energy systems. International Journal of Science, Architecture, Technology and Environment, 1, 151-163.
98) Shittu, H., Adeniji, I. O., Oteri, O., & Shittu, M. A. (2026). Autonomous energy management systems for port and maritime electrical infrastructure using digital-twin-driven architectures. International Journal of Advanced Multidisciplinary Research and Studies, 6(1), 1792-1805.
99) Shittu, H., Olagunju, F., & Shittu, M. A. (2024). Cyber physical resilience in digital substations: IoT enabled adaptive protection for secure DER integration. International Journal of Science, Architecture, Technology and Environment, 1(3), 81-99.

Corresponding Author: Kingsley Chinazaekpere Ndupu

100) Shittu, M. A., Shittu, H. A., Adeleke, O. J., & Adedokun, O. J. (2023). Digital twin modeling for real time monitoring and fault detection in smart substations. International Journal of Industrial Engineering Research and Development, 14(2), 25-44.
101) Ståhl, D., & Bosch, J. (2014). Modeling continuous integration practice differences in industry software development. Journal of Systems and Software, 87, 48-59.
102) Sunday, E. A., & Omoegun, G. O. (2022). Smart fault detection in HVAC systems using sensor-based monitoring. International Journal of Scientific Research in Science, Engineering and Technology, 7(4), 380-403.
103) Sunday, E. A., Omoegun, G. O., Essien, M. A., & Oluokun, O. A. (2020). Transitioning from reactive to predictive maintenance in mechanical systems. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 6(6), 425-447.
104) Tao, F., Zhang, H., Liu, A., & Nee, A. Y. C. (2019). Digital twin in industry: State-of-the-art. IEEE Transactions on Industrial Informatics, 15(4), 2405-2415.
105) Tonoyan, A., Dada, O., & Ayivi-Donkor, S. S. (2024). Real-time KPI tracking systems: A review of automated performance monitoring and data-driven decision making. World Journal of Innovation and Modern Technology, 8(6), 247-281.
106) Utting, M., Pretschner, A., & Legeard, B. (2012). A taxonomy of model-based testing approaches. Software Testing, Verification and Reliability, 22(5), 297-312.
107) Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. Software Testing, Verification and Reliability, 22(2), 67-120.

Corresponding Author: Kingsley Chinazaekpere Ndupu