iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: https://doi.org/10.22214/ijraset.2026.85342

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

An Agentic AI Framework for Master Data Synchronization, Validation and Self-Healing Across Integrated Healthcare Information Systems

Samiyullaha Sayyed
Independent Researcher, India

Abstract: ***Background. Large healthcare organizations run many specialty applications, such as prescribing and dispensing, warehouse and inventory, laboratory, blood bank and billing, that depend on shared master data, chiefly the patient identifier (MRN) and the product identifier with its description. Standards such as HL7 v2 and FHIR can propagate a new or changed record from an origin system, but propagation failures (rejections, unmapped codes, missing mandatory fields, duplicates and timing dependencies) still rely on manual diagnosis and repair. The resulting gaps surface as held or failed transactions at the point of care and supply. Failures also arise without any interface error, when a record exists but its local build or configuration (for example location, charge or dispensing parameters) is incomplete. Objective. To specify a human-governed, closed-loop framework in which AI agents act as integration architect, builder, verifier and support engineer: designing field mappings, configuring receiving systems against an approved build standard, verifying cross-system data validity and local build completeness, and diagnosing and remediating propagation errors. Methods. The study followed a design-science approach. The synchronization gap is formalized, a seven-class failure taxonomy is derived, and a four-layer architecture (sense, reason, act, govern) is specified with tiered autonomy grounded in the Parasuraman-Sheridan-Wickens model of types and levels of automation. An explicit tier-decision rule and tool contracts are defined, verification checks are mapped to established electronic health record data quality frameworks, the closed loop is illustrated with a synthetic worked example, and a staged evaluation protocol with quantified safety gates is specified. Results. The framework specifies agent roles and tool contracts; a proactive build-completeness check for local configuration; a default and maximum autonomy tier for each of seven failure classes; a decision rule in which the permitted tier is the minimum of a policy tier, a confidence tier and a criticality cap; safeguards (patient identity actions reserved for humans, grounding in versioned build standards, deterministic pre-execution validation, and treatment of message content as untrusted data) mapped to a threat model; and evaluation metrics (time to consistency, held-transaction rate, false-fix rate, harmful-action rate) with the sample sizes needed to demonstrate safety bounds, for example 149 consecutive correct fixes to show with 95% confidence that the false-fix rate is below 2%. This is a conceptual and design paper; no empirical results are reported. The tier-decision rule was checked exhaustively against ten stated properties over 11,440 input combinations, with no violations, and six deliberately seeded faults were each detected (Section IV-I). Conclusions. The framework proposes that an AI agent layer can be added over existing integrated systems without replacing them, turning error queues from passive backlogs into a supervised remediation loop. Whether it achieves this is untested. The paper supplies testable hypotheses, pre-specified safety gates and a staged path from shadow mode to limited autonomy for future empirical validation.***

Keywords: agentic AI, large language models, HL7 FHIR, interoperability, master data management, data quality, healthcare information systems, human-automation interaction, hospital supply chain.

I. INTRODUCTION

A modern hospital or health system is rarely one application. It is a federation of specialty systems, each optimized for its own workflow: one for prescribing and dispensing, another for warehouse and stores, others for the laboratory and the blood bank, and another for billing. These systems are integrated, and they depend on a small set of shared master data. The most important are the patient identifier (the medical record number, MRN) and the product identifier with its description. Both must be present, unique and consistent in every system that transacts against them. Interoperability standards have matured to carry this data. HL7 version 2 messages remain widely deployed for intra-organizational exchange, and HL7 FHIR introduced a resource-based, web-oriented approach designed to be simpler to implement [1,2,3]. In principle, when a patient or product is created in the origin system, the record can be propagated automatically to every dependent system.

©IJRASET: All Rights are Reserved

In practice, propagation is only half of the problem. Acceptance of a message depends on the configuration of the receiving system: its code tables, units of measure, mandatory fields, location-level build and validation rules, which standards deliberately leave local. A syntactically valid message can therefore be rejected, queued, or accepted incompletely. In many organizations the remaining work is manual.

Stakeholders of each system build the missing record by hand, or a support analyst reads the interface error, finds the cause, corrects the build and resends. The gap between record creation and availability is a latent condition in the sense of Reason's model of organizational accidents [9]. It stays invisible until a dependent transaction, such as a dispense, a receipt, a specimen label or a charge, fails or is held. The consequences are delay, rework, workarounds such as duplicate creation, and reconciliation effort. A recent narrative review of hospital pharmacy stock discrepancies reaches a related conclusion for physical stock: systems can be integrated at the interface level without sharing a transaction-level record, so they can legitimately disagree until a reconciliation step resolves the difference [34].

The problem is most visible in the supply chain that feeds clinical departments. A hospital runs on a continuous flow of materials: syringes, gloves, cleaning solutions, linen such as bed sheets, operational tools, tubing, laboratory consumables and medications. Procurement starts in the materials-management department, which classifies each item (pharmaceutical, laboratory, surgical, non-surgical and so on) and creates the item master with a unique item code that is primary in the materials system. The same item must then exist in the other systems that handle it, often from different vendors: the pharmacy information system, the laboratory information system and billing each keep their own item master. In many organizations each downstream record is created by hand by IT or application staff, system by system.

When an item has been purchased and received but its record does not yet exist in the pharmacy or laboratory system, it cannot be dispensed or used for a test, even though it is on the shelf. The pharmacist or technologist must then ask the IT team to create the item and complete its parameters. The same happens when the record exists but the local build or configuration is incomplete, for example when the item has not been added to the dispensing location or has no charge mapping in billing: the transaction is held at the point of care or billing, and no interface error is raised. In the author’s practical experience, a routine build request can take one to two days, and an urgent request raised as a high-priority ticket can take two to three hours; meanwhile a patient may wait more than two hours for a medication or a laboratory test. These figures are practitioner observations, not measured data. They motivate the baseline measurement in phase P0 (Section VI) and are not offered as evidence of effect. Two consequences follow. Item masters are duplicated across vendors and must be kept in agreement, and creating the minimum record needed for an item to be available is repetitive human work that delays care. The framework proposes that, once the origin item is created, the minimum downstream records are created or proposed automatically, that missing local build or configuration is detected and, within defined limits, completed from approved templates, and that the agents alert the responsible owners to the further parameters that remain to be completed (for example high-alert status, formulary type, drug classification, and facility and location mapping).

Recent large language model (LLM) agents can read logs and documentation, reason about causes, and call tools to act [10,11]. This suggests a different operating model for integration support. Instead of an error queue that waits for a person, an agent could classify each failure, retrieve the relevant build standard, propose or apply a fix within defined limits, and verify that the data now agree across systems. The same capabilities carry risks: unsupported (hallucinated) output [12], non-deterministic behavior, security exposure, and, in healthcare, patient-safety implications. A practical design must therefore specify not only what the agents do but also where they must stop.

This paper makes five contributions: (1) a formal definition of the synchronization gap and a seven-class taxonomy of failures; (2) a layered architecture and four-role agent design, with explicit tool contracts, for building, configuring, verifying and supporting master data synchronization; (3) a tiered autonomy policy that assigns each failure class a default and a maximum level of automation and combines it with diagnostic confidence and attribute criticality in an explicit decision rule; (4) a staged evaluation protocol with metrics, hypotheses and quantified safety gates; and (5) an end-to-end illustrative walk-through and a threat model that show how the pieces fit together. The scope is deliberately limited: the paper is conceptual, uses no patient data, and reports no empirical results.

The remainder of the paper is organized as follows. Section II reviews related work. Section III formulates the problem. Section IV presents the framework, including a worked example (Section IV-H) and a specification-level verification of the tier-decision rule (Section IV-I). Section V addresses safety, security and governance, including a threat model. Section VI proposes the evaluation methodology. Sections VII and VIII discuss implications, limitations and future work.

II. BACKGROUND AND RELATED WORK

A. *Interoperability Standards and the Persistence of Integration Failure*

HL7 v2 messaging has been deployed internationally for decades and remains a common choice for exchanging patient and operational information within and between hospitals [23]. HL7 v3 attempted a more rigorous model but was criticized as complex and costly to implement [1]. FHIR adopts a resource-based design with RESTful APIs, JSON and XML representations, and a modular specification intended to lower the barrier to implementation [1,2,3]. The SMART on FHIR experience shows that standards-based APIs can enable substitutable applications, while also exposing the need for pragmatic constraints on the base specification before production use [2]. These advances address how data are expressed and transported. They do not remove the local configuration dependencies of each receiving application, which is where the failures considered in this paper arise. Systematic reviews of FHIR implementations similarly report that practical challenges accompany the standard's benefits [24].

B. *Master Data and Data Quality in Healthcare*

Data quality is multidimensional. Wang and Strong framed quality in terms of fitness for use by data consumers [5]. In the health setting, Weiskopf and Weng reviewed methods and dimensions of EHR data quality assessment, identifying completeness, correctness, concordance, plausibility and currency as recurring dimensions [6]. Kahn and colleagues harmonized existing terminologies into a framework of conformance, completeness and plausibility, each assessed by verification and validation [7]. These frameworks were developed mainly for secondary use of data (research and analytics). We reuse their categories here for an operational purpose: continuously verifying that shared master data are valid in every transacting system. For identity-related problems, probabilistic record linkage theory [8,25] provides the classical basis for detecting possible duplicates, a task that remains a matter for human adjudication in our framework.

C. *Integration Patterns and Error Handling*

Enterprise integration patterns describe the building blocks of messaging-based integration, including message translation, guaranteed delivery, idempotent receivers and the dead letter channel, in which undeliverable messages are routed to a queue for inspection [4]. These patterns make failures visible and recoverable. The remediation step, however, is conventionally performed by an analyst who must interpret the error, locate the relevant build knowledge and apply a correction. Much of that knowledge is held in specifications, runbooks and the experience of individuals.

D. *LLM-based Agents*

LLM-based autonomous agents combine a language model with profiling, memory, planning and action modules to complete tasks by interleaving reasoning and tool use [10,11]. Reasoning-and-acting patterns allow a model to consult external sources and take actions, which reduces reliance on parametric knowledge alone [10]. Reliability remains a central concern: language models can produce fluent but unsupported content, which has been surveyed extensively [12]. Work on LLMs and machine learning in medicine concentrates largely on clinical knowledge and decision support [13,14], and shows that capability must be evaluated carefully before deployment.

Three adjacent strands are relevant. First, foundation models can perform data cleaning and integration tasks such as entity matching, error detection and imputation with few or no task-specific examples [26]; this supports the Architect and Verifier roles but does not address who may act on the output. Second, research on AIOps surveys methods for failure prediction, detection, diagnosis and remediation in IT operations [27]; it addresses mainly infrastructure telemetry rather than the business semantics of master data, and it does not treat patient-safety constraints on remediation. Third, a systematic review of 519 evaluations of LLMs in health care found that only about 5% used real patient data and that most concentrated on accuracy rather than other dimensions of performance [28], which motivates the safety-gated, operational-metadata-based evaluation design in Section VI. On the security side, indirect prompt injection, in which adversarial instructions are embedded in data that an LLM-integrated application retrieves, has been demonstrated against real-world applications [29].

Within the sources reviewed for this paper, we did not identify a framework specifying governed, tiered-autonomy agents for operational master data synchronization across integrated healthcare applications. This statement is limited to the sources reviewed: related work here is selective rather than systematic, and a systematic literature review is planned (Section VIII).

Search note. On 22 October 2026 the author searched the web for closely related work using combinations of the terms agentic AI, LLM agents, HL7, FHIR, master data, interface errors, self-healing, tiered autonomy and hospital information systems. This was a limited web search, not a systematic database search, and it may have missed relevant work.

The closest item found is a pattern catalogue and orchestration framework for governed agent ecosystems in hospital information management systems, which defines agent roles for reconciliation, auditing and decision support with a risk-tier model and human-oversight checkpoints [36]. That work is broader in scope and addresses governance of hospital agents in general; it does not define a taxonomy of master data propagation failures, a per-class autonomy ceiling combined with confidence and criticality in an explicit decision rule, or a build-completeness check, which are the contributions made here. Other items found, on LLM-based data transformation and on autonomy levels for industrial agents, were preprints or reports whose bibliographic details could not be verified and are therefore not cited.

E. Human-automation Interaction

Parasuraman, Sheridan and Wickens proposed that automation can be applied to four classes of function (information acquisition, information analysis, decision and action selection, and action implementation) and at levels from fully manual to fully automatic [15]. A meta-analysis of the human-performance consequences of automation found benefits for routine performance as the degree of automation rises, but worse performance and situation awareness when automation fails, with the largest negative effects when automation moves from supporting analysis to supporting action selection [16]. This evidence motivates tiered autonomy: agents may be given wide latitude to acquire and analyze information while action authority is granted selectively and incrementally. A sociotechnical view of health IT further reminds us that technology, workflow, people, governance and measurement interact [17].

F. Research Gap

Standards move data; data quality frameworks define what valid data look like; agent research shows that models can reason and act with tools. What is missing is an integrated, safety-conscious specification of how agents can close the loop from propagation failure to verified correction in multi-system healthcare environments. This paper addresses that gap at the design level. Section VII positions the proposal against the alternatives an organization would otherwise use (Table IX).

III. PROBLEM FORMULATION

Let $S = \{s_0, s_1, \ldots, s_n\}$ be the set of systems, where s_0 is the origin system for a master entity type (patient or product) and s_1 to s_n are receiving systems, such as pharmacy, warehouse, laboratory, blood bank and billing. An entity e is created at origin at time $t_c(e)$. Each receiving system s_i should hold a valid representation $r_i(e)$ before the first transaction that depends on it. We define the *synchronization gap* at time t as:

$$G(e, t) = \{\ s_i \in S \setminus \{s_0\} : r_i(e) \text{ is absent, invalid, incomplete or in conflict with the origin record at } t\ \}$$

Here a representation is invalid if it fails a check in Table IV (for example, a code that is not defined in the receiving system's tables), incomplete if it lacks local build or configuration that a dependent transaction requires (for example, an item that is not built at the dispensing location or has no charge mapping), and in conflict if it is valid in isolation but disagrees with the origin record (for example, a different description or a possible duplicate identifier). The definition therefore covers incomplete build (F7), identity conflicts (F5) and semantic drift (F6) as well as absent or rejected records (F1 to F4). A representation counts as valid for this purpose when it contains at least the minimum build needed for the first dependent transaction (for example item code, description, unit of measure and location); the remaining parameters are completed afterwards.

The *time to consistency* is the elapsed time until the gap closes:

$$\mathrm{TTC}(e) = \min\ \{\ t - t_c(e) : G(e, t) = \emptyset\ \}$$

A dependent transaction x that references e at system s_i at time t_x is *held or failed* if $s_i \in G(e, t_x)$. Fig. 1 illustrates the idea schematically. The operational objective is to minimize the expected TTC and the rate of held transactions, subject to a safety constraint: no autonomous action may alter patient identity or clinically significant product attributes without human authorization, and every autonomous action must be attributable, verifiable and reversible.

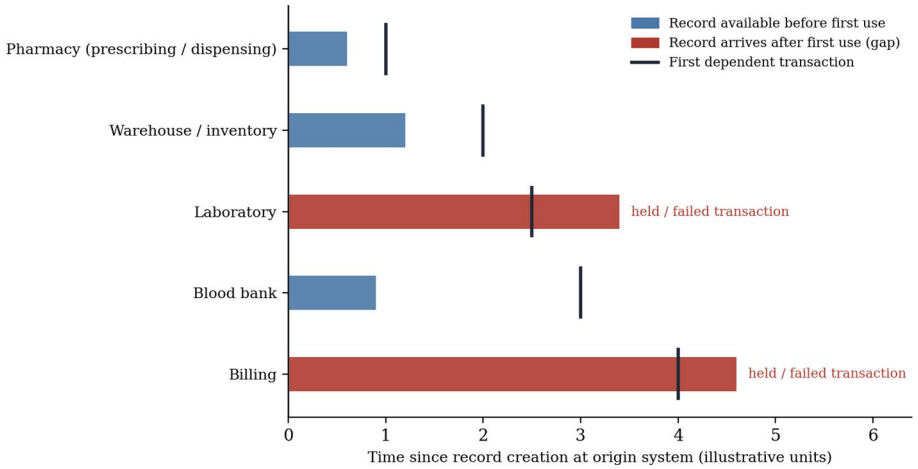


Fig. 1 Schematic synchronization gap. Bars show time until the record is available in each receiving system; vertical marks show the first dependent transaction. Where the bar extends beyond the mark, the transaction is held or failed. Values are illustrative, not empirical data.

A. Failure Taxonomy

Drawing on common integration practice, we group failures of master data propagation and local build into seven classes (Table I). Each incident is assigned a primary class by its most proximate root cause, and any secondary classes are recorded as tags; an incident with several classes is handled at the most restrictive tier among them. The classes are therefore not strictly mutually exclusive (for example, a code mismatch (F1) can coexist with semantic drift (F6)), and the reliability of the scheme is to be assessed in phase P1 by inter-rater agreement (Cohen's kappa) between analysts and the agent. F1 to F6 arise from messages or records that fail on propagation; F7 arises inside a receiving system and may produce no message at all.

TABLE I
FAILURE TAXONOMY FOR MASTER DATA PROPAGATION AND LOCAL BUILD

Class	Failure type	Typical example	Typical root cause
F1	Reference or code mismatch	Unit of measure or category rejected as not defined	Value exists at origin but is absent from the receiving system's code table or crosswalk
F2	Structural or mandatory field	Required field empty; description exceeds length limit	Field mapping incomplete or origin data below receiving-system requirements
F3	Dependency or ordering	Product references a group or location not yet built; queued message after interface pause	Parent record missing or message processed out of order
F4	Transport or availability	Timeout; interface down; queue backlog	Network, endpoint or engine fault
F5	Identity conflict	Possible duplicate MRN; demographic mismatch	Duplicate registration or conflicting source data
F6	Semantic drift	Same product with different descriptions in two systems	Attribute edited locally after creation; no governance of the origin as source of truth
F7	Incomplete local build or configuration	Item exists but is not built at the dispensing location; no charge mapping in billing; missing storage or reorder parameters	Required local parameters absent or incomplete; no interface message is generated, so the gap surfaces only at the dependent transaction

IV. PROPOSED FRAMEWORK

A. Design Principles

- *Origin as source of truth.* Master data are defined at the origin; receiving systems are brought into agreement with it, never the reverse, unless a human decides otherwise.
- *Grounding in the build standard.* Agents reason over a versioned, approved build standard (interface specifications, field maps, code tables, runbooks, past incidents) rather than free-form model knowledge.
- *Tiered, earned autonomy.* Authority to act is assigned by failure class and expanded only after evidence.
- *Verify after acting.* No fix is complete until cross-system verification passes.
- *Attribution and learning.* Every step is logged, and outcomes update an incident memory that improves future diagnosis.

B. Architecture

The framework is an overlay on existing applications and the integration engine (Fig. 2). It does not replace them. It adds four layers. The sense layer subscribes to integration-engine acknowledgments and queues, origin change events (for example ADT feeds or FHIR subscriptions), read-only application logs and configuration snapshots. The reason layer hosts the agent team and retrieves evidence from the build standard and incident memory. The act layer exposes narrowly scoped tools: propose a configuration change as a structured difference, apply it in a test environment, run test messages, promote through the organization's change-control process, and replay a message. The govern layer enforces the autonomy policy, approval workflow, audit log, monitoring of the agents themselves and an emergency stop.



Fig. 2 Layered architecture of the agent overlay. Existing applications and the integration engine are unchanged.

C. Agent Roles

- *Architect agent.* Reads specifications and sample data for each system, proposes field mappings and crosswalks (for example origin description and unit to the receiving system's equivalents), and drafts interface specifications for human review.
- *Builder agent.* Prepares and, where authorized, applies configuration changes such as code-table entries and required-field settings, first in test, with test messages and a recorded difference. It can also apply an approved location template to complete a missing local build.
- *Verifier agent.* Continuously compares each entity across systems against the checks in Table IV and raises discrepancies, including those that never generate an interface error. It also runs a build-completeness check, comparing each receiving system's configuration with the profile that the build standard requires for the item class and location.
- *Support agent.* Monitors queues, classifies failures (Table I), diagnoses root cause, requests or performs the permitted remediation, replays messages and confirms resolution.

Roles may be realized as one agent with different tools or as cooperating agents. The separation matters mainly for permissions: for example, the verifier needs read access only, which keeps the checking function independent of the component that makes changes. The agents act on the environment only through a small set of tools with explicit permissions, inputs and outputs (Table II). Tool contracts matter for two reasons: they make the permission model auditable, and they allow a deterministic policy engine, rather than the language model, to decide whether a given call is permitted.

TABLE II
TOOL CONTRACTS FOR THE AGENT LAYER

Tool	Permission	Used by	Input and output	Guard
read_queue()	Read-only	Support, Verifier	No input; returns ACK/NACK events with codes, record keys and timestamps	Free text quarantined and passed only as quoted data
get_record(system, key)	Read-only	Verifier, Support	Entity key; returns a normalized record with masked identifiers	Masking or tokenization; every access logged
query_standard(q)	Read-only	All agents	Query; returns versioned passages with section identifiers	Output must cite the passage identifier relied on
propose_change(diff)	Proposal only	Architect, Builder	Structured difference; returns a change-request identifier	Schema and policy validation; no execution
apply_in_test(id)	Write: test only	Builder	Change-request identifier; returns test outcome and regression results	Test environment only
promote_change(id)	Write: controlled	Builder, Support	Change-request identifier; returns a change-control ticket or an execution record	Allowed only if the tier permits and any required approval is recorded
replay_message(id)	Write: idempotent	Support	Message identifier; returns the delivery outcome	Idempotency key; rate limit
escalate(case)	Notify	All agents	Case packet (diagnosis, evidence, proposed fix); creates a work-queue item	Always available; the stop command is human-only

D. Closed-loop Remediation

Fig. 3 and Algorithm 1 describe the loop. The key properties are that the agent proposes changes as structured objects that can be validated deterministically (schema, policy and a dry run) before any execution, that the permitted action depends jointly on failure class and diagnostic confidence, and that verification and rollback are part of the loop rather than afterthoughts.

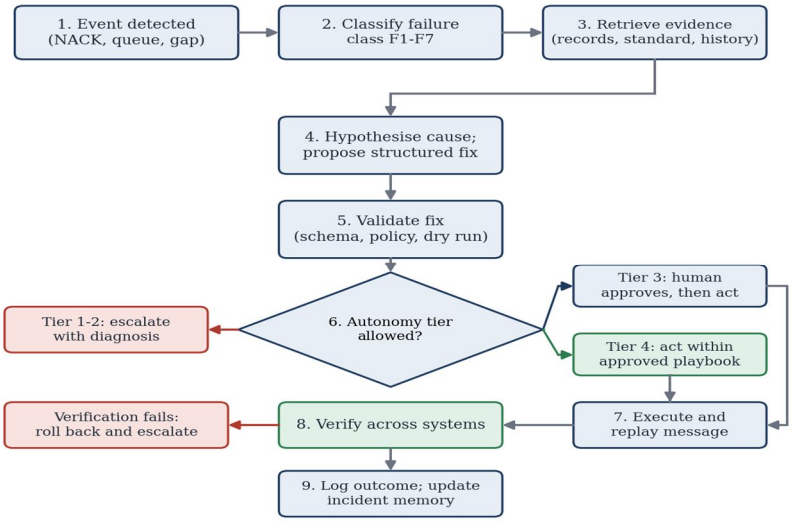


Fig. 3 Closed-loop remediation. Tiers are defined in Section IV-E.

Algorithm 1. Remediate(event)

1. (class, c) ← Classify(event) // primary class F1 to F7, with confidence c
2. evidence ← Retrieve(origin record, receiving record, build-standard sections, similar past incidents)
3. cause, fix ← Diagnose(evidence) // fix is a structured change, not free text
4. if not Validate(fix) {schema, policy, test dry run} then Escalate(event, cause)
5. tier ← min(PolicyTier(class), ConfidenceTier(c), CriticalityCap(fix))
6. if tier ≤ 2 then Escalate(event, cause, fix)
7. if tier = 3 and not RequestApproval(fix) then Escalate(event, cause, fix)
8. Execute(fix); Replay(event)
9. if Verify(entity) fails then Rollback(fix); Escalate(event, cause)
10. Log(event, cause, fix, outcome); UpdateIncidentMemory()

PolicyTier is the current tier of the failure class, between its default and its ceiling (Table III). ConfidenceTier and CriticalityCap are defined in Section IV-E.

E. Tiered Autonomy Policy

Following the stages-and-levels view of automation [15,16], we define four tiers of action authority. Tier 1: alert only. Tier 2: diagnose and recommend; a human acts. Tier 3: prepare the change and act after explicit human approval. Tier 4: act within a pre-approved playbook, with after-the-fact review. Each failure class has a default tier at which the agent starts and a ceiling it may reach only through the evaluation gates in Section VI (Table III). Changes that touch patient identity, or clinically significant product attributes (for example strength, dose form or controlled-substance schedule), are capped at Tier 2 regardless of class. The same cap applies to prices and charge amounts, which carry financial consequences. Safety-relevant configuration flags (for example high-alert status, formulary type and drug classification) are capped at Tier 3.

The tier at which an action is permitted is the minimum of three quantities: the policy tier of the failure class (starting at its default and raised toward its ceiling only after the evaluation gates), a confidence tier, and a criticality cap. The confidence tier maps the agent's diagnostic confidence c to a maximum tier: $c \geq 0.90$ allows up to Tier 4; $0.75 \leq c < 0.90$ up to Tier 3; $0.50 \leq c < 0.75$ up to Tier 2; and $c < 0.50$ Tier 1. Because a language model's stated confidence may be poorly calibrated, c is not taken from the model's own probability alone. It is computed from verifiable signals: agreement between the failure class and the interface error code, whether the cited build-standard passage matches the field and value at issue exactly, whether the dry run in the test environment succeeds, and agreement among repeated independent diagnoses. The thresholds are provisional and are to be calibrated in phase P1 against analysts' recorded resolutions, using reliability diagrams and expected calibration error. The criticality cap is Tier 2 for any change that touches patient identity, product-defining clinical attributes (for example strength, dose form or controlled-substance schedule), or financial parameters such as prices and charge amounts; Tier 3 for safety-relevant configuration flags such as high-alert status, formulary type and drug classification; and Tier 4 otherwise.

TABLE III
AUTONOMY TIERS BY FAILURE CLASS

Class	Failure type	Default tier	Ceiling tier	Rationale
F1	Reference or code mismatch	3	4	Ceiling applies only to mappings already in the approved crosswalk
F2	Structural or mandatory field	3	3	Missing data may need a clinical or business owner's input
F3	Dependency or ordering	3	4	Replay after prerequisite verified is low risk and idempotent
F4	Transport or availability	4	4	Retry with back-off; escalate if persistent
F5	Identity conflict	2	2	Merge and unmerge decisions remain human; agent may only flag and package evidence
F6	Semantic drift	2	3	Origin proposed as source of truth; owner decides
F7	Incomplete local build or configuration	3	4	Ceiling of 4 applies only to mechanical mappings from an approved template or policy table; safety-relevant clinical flags are capped at Tier 3; prices, charges and product-defining attributes stay at Tier 2

F. Verification Layer

A message being accepted does not establish that the data are valid. The verifier agent therefore applies checks organized by the data quality categories of Kahn et al. [7] and the dimensions of Weiskopf and Weng [6] (Table IV). The checks run on a schedule and on each creation event, so that discrepancies are found before the first dependent transaction wherever possible. Such a layer matters because discrepancies are often invisible to users: in a Markov-model analysis of about two million automated dispensing cabinet transactions, only about 6% of discrepancies generated a user report [35]. Because an incomplete local build raises no interface error, the completeness check runs proactively on each creation event and on a schedule, so that a missing configuration is flagged before the first dependent transaction and not after a patient is already waiting.

TABLE IV
CROSS-SYSTEM VERIFICATION CHECKS MAPPED TO DATA QUALITY CATEGORIES

Check	What is verified	Framework category
Presence	Every new MRN and Product ID exists in every system that needs it	Completeness
Value validity	Codes, units and formats are valid in the receiving system's tables	Value conformance
Referential integrity	Parent records (groups, locations, categories) exist	Relational conformance
Uniqueness	No duplicate identifiers; no two identifiers for one patient or product	Uniqueness plausibility
Concordance	Descriptions, units and key attributes agree across systems, or differences are explained	Concordance [6]
Timeliness	Record is available before its first dependent transaction	Temporal plausibility; currency [6]
Build completeness	Every parameter that the build standard requires for the item class and location (for example dispensing location, charge mapping, storage bin, reorder settings) is present in each receiving system	Completeness (configuration)

Required build profiles. The completeness check needs a definition of what ‘fully built’ means for each receiving system. The framework represents this as a build profile: a versioned list, kept in the build standard, of the parameters that an item class requires in that system, with the source of each value and the highest tier at which an agent may supply it. For a medication in the pharmacy information system, a profile could include high-alert status, formulary type, drug type, drug classification, the mapping of the item to the hospital facility, and its mapping to pharmacy and store locations. The laboratory and billing systems have their own profiles, for example test and specimen parameters and charge mapping. Each parameter has one of three sources: copied from the origin record (for example description and unit of measure); resolved by deterministic lookup in an authoritative drug knowledge base (for example drug classification from a licensed database such as Multum); or taken from an institutional policy table or approved location template (for example facility and location mapping, or a formulary type already decided by the responsible committee). A parameter with no authoritative source, which needs clinical or business judgement, is never inferred by the agent; it is flagged to the named owner.

Replacing manual queue review. In the usual workflow the origin system sends the new item to each receiving system as an HL7 or FHIR message, which waits in an inbound queue until that system’s administrator reviews and approves it and then completes the build by hand. In the framework the same message triggers the completeness check. The agent compares the message with the receiving system’s build profile, prepares the missing configuration as a structured change from the sources above, and presents it as one reviewable item with its evidence (Tier 3), so the administrator approves a prepared build instead of constructing it. Once a parameter class has passed the evaluation gates in Section VI, mechanical mappings such as facility and location mapping from an approved template can be applied within a pre-approved playbook (Tier 4). Safety-relevant flags such as high-alert status, formulary type and drug classification stay at Tier 3 at most, because an error changes how the drug is ordered, verified and dispensed; strength, dose form and controlled-substance schedule define the product itself and stay at Tier 2, where the agent can only flag a disagreement with the knowledge base. Raising the ceiling for a flag requires a false-fix rate within the pre-specified bound and pharmacist review of a sample of applied changes.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

G. Reliability of the Agent Itself

To limit unsupported output, the framework (a) restricts reasoning to retrieved, versioned sources and requires the agent to cite the section of the standard it relied on; (b) requires structured outputs that a deterministic validator can check; (c) uses a dry run in a test environment before any production change; (d) applies confidence thresholds below which the agent must escalate rather than act; and (e) ensures idempotent replay, so that resending a message cannot create duplicates [4].

H. Illustrative Walk-through

To make the loop concrete, we trace one synthetic incident end to end (Fig. 4 and Table V). All identifiers, codes and timings are invented for illustration. No real system or patient data are involved, and the example is not evidence of performance. A new item is created in the origin materials-management system, which holds the item master and assigns the unique item code, and is propagated to the pharmacy information system as an HL7 v2 master-file notification (MFN^M15, inventory item master). The interface translates the origin unit of measure ‘STRIP’ to the pharmacy code ‘STR’ listed in the approved crosswalk, but the pharmacy unit-of-measure table has not yet been given that code. The pharmacy system therefore rejects the message with the following simplified acknowledgment (segment and field positions are illustrative). Until the record is built, the item is on the shelf and in the materials system but cannot be dispensed:

```
MSH|^~\&|PIS|PH01|MMS|MM01|20261020090002||MFK^M15^MFK|MSG000457|P|2.5.1
MSA|AE|MSG000456
ERR||IIM^1^4|103^Table value not found^HL70357|E|||Unit of measure STR not defined
```

The support agent classifies the failure as F1 with confidence 0.93, based on the HL7 error code 103 (table value not found) and the field in error. It retrieves the crosswalk row and the state of the pharmacy unit-of-measure table in test and production, and it proposes a structured change object rather than free text:

```
{
 "incident": "INC-0001 (synthetic)",
 "failure_class": "F1",  "confidence": 0.93,
 "target_system": "pharmacy",
 "action": "add_code_table_entry",
 "table": "UNIT_OF_MEASURE",
 "entry": {"code": "STR", "description": "Strip"},
 "evidence": ["BuildStd v3.2 sec 4.1.2, crosswalk row UoM-017",
              "ACK ERR-3 = 103"],
 "preconditions": ["code absent in test and production"],
 "rollback": "delete_code_table_entry(UNIT_OF_MEASURE, STR)"
}
```

A deterministic validator checks the object against the tool schema and the policy, and a dry run in the test environment confirms that replaying the original message is accepted. The tier decision is then min(PolicyTier = 3, ConfidenceTier = 4, CriticalityCap = 4) = 3, so the change is prepared and a human approves it through change control (phase P3). Once class F1 has earned its ceiling through the gates in Section VI, the same change could be applied within a pre-approved playbook (Tier 4) with after-the-fact review, because it uses a mapping that is already in the approved crosswalk. After approval the agent applies the change and replays the message, and the verifier confirms presence, value validity and concordance in the pharmacy system, so that the gap G(e, t) is closed. Without such a loop, the same rejection would wait for a manual build request, which in the author’s experience can take hours to days (Section I).

©IJRASET: All Rights are Reserved

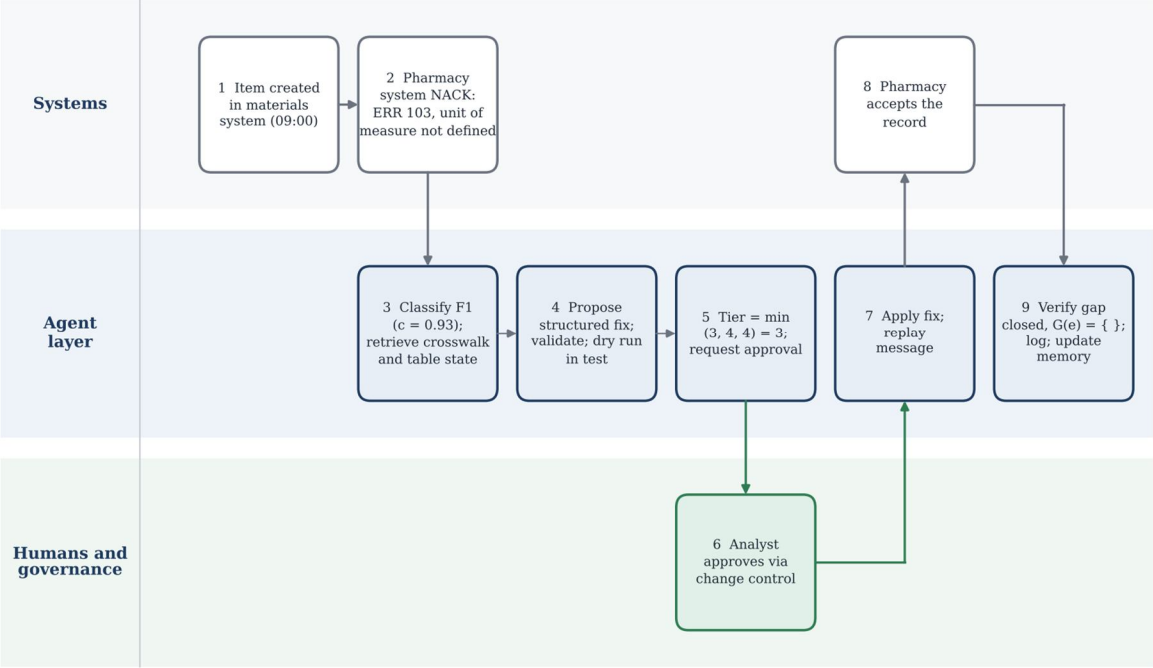


Fig. 4 Illustrative end-to-end remediation of a synthetic F1 incident. Lanes show systems, the agent layer and human governance; step numbers match Table V. All data are synthetic.

TABLE V
WORKED EXAMPLE: TRACE OF A SYNTHETIC INCIDENT (ILLUSTRATIVE; NO EMPIRICAL DATA)

Step	Layer	What happens	Control applied
1	System	Item created in the materials-management system (origin of the item master and item code) and propagated to the pharmacy system as MFN^M15.	Origin is the source of truth.
2	System	Pharmacy system rejects the message with ERR 103: unit of measure not defined; NACK queued.	Sense layer captures the ACK/NACK; free text is treated as data.
3	Agent	Classify as F1 (c = 0.93); retrieve crosswalk row UoM-017 and the unit-table state in test and production.	Reasoning limited to retrieved, versioned sources; citation required.
4	Agent	Propose a structured change (add code STR); schema, policy and dry-run validation pass.	Deterministic validator before any execution.
5	Agent	Tier = min(3, 4, 4) = 3; approval requested.	The policy engine, not the model, decides the tier.
6	Human	Analyst reviews the evidence and the difference and approves through change control.	Named accountable owner; audit log.
7	Agent	Apply the change; replay the message.	Idempotency key prevents duplicate records.
8	System	Pharmacy system accepts the record; the item can be dispensed.	Positive acknowledgment (MSA|AA).
9	Agent	Verify presence, value validity and concordance across systems; log the outcome; update incident memory.	Verify after acting; roll back if any check fails.

Variants show how the same loop behaves under tighter constraints. (a) If the origin unit had no row in the approved crosswalk, the agent could not cite an approved mapping; it would escalate at Tier 2 with a proposed mapping and its evidence, and a data owner would decide. (b) If the incident involved a change to strength, dose form or controlled-substance schedule, the criticality cap would hold it at Tier 2 whatever the failure class. (c) If diagnostic confidence were 0.70, the confidence tier would cap the action at Tier 2

even for an F1 incident with a crosswalk match. (d) If the dry run failed, the change would not be proposed for execution and the incident would be escalated with the failing test output. (e) If the item arrived correctly but was never built at the dispensing location, no interface error occurs; the build-completeness check would flag the missing local build when the item is created, and the agent would propose applying the approved location template (Tier 3 by default). (f) If the item is a medication, the same creation event also triggers the pharmacy build profile (Section IV-F): the agent resolves the drug classification from the licensed drug knowledge base, proposes the facility and location mappings from the approved template, and flags high-alert and formulary status for pharmacist approval (Tier 3).

I. Specification-level Verification of the Tier-decision Rule

Because the decision rule determines what an agent may do without a human, it is checked separately from the language model. The rule is a deterministic function of the failure class, the current policy tier, the diagnostic confidence, the attribute criticality and the result of pre-execution validation. Its input space is small enough to enumerate. For each of the seven failure classes, each policy tier between its default and its ceiling, 104 confidence values (every 0.01 from 0 to 1, plus the values just below each threshold), five criticality categories and both validation outcomes, the function was evaluated, giving 11,440 combinations. Eight safety properties and two liveness properties were checked on every combination. The safety properties include that identity, product-defining and financial changes are never executed, that safety-relevant flags are never executed autonomously, that identity conflicts (F5) are never executed, that no action exceeds the class ceiling, that nothing is executed below a confidence of 0.75 or after a failed validation, and that the tier does not fall as confidence rises. The liveness properties ensure that the rule is not safe merely because it never acts: an eligible case must be executed autonomously and an approval-eligible case must reach at least approval.

The reference rule violated none of the ten properties. To test whether the checks can detect faults, six mutants of the rule were created, each with one deliberate defect: removal of the criticality cap, removal of the confidence cap, replacement of the minimum by the maximum, ignoring the validation result, relaxing the F5 restriction, and an off-by-one error at the 0.90 boundary. Every mutant violated at least one property, and the boundary error was detected only by a liveness property, which shows why both kinds are needed.

This is verification of the rule as specified, not validation of the framework. It shows that the rule is internally consistent and that these classes of implementation error would be caught. It does not show that the confidence values are calibrated, that the failure classes are assigned correctly, that the criticality categories suit a given organization, or that the thresholds are clinically adequate; those remain empirical questions for the evaluation in Section VI. The properties were written by the same author as the rule, so independent review of them is also needed.

V. SAFETY, SECURITY AND GOVERNANCE

- *Patient identity*. Merging, unmerging and correcting patient identity are never autonomous. Agents may flag a possible duplicate and assemble evidence for medical records staff.
- *Change control*. Production changes go through the organization's existing change-control process; agents act within it and cannot bypass it.
- *Least privilege*. Each role receives the minimum permissions: the verifier is read-only; the builder can write only to designated configuration areas; no agent holds clinical-ordering rights.
- *Privacy by minimization.* Diagnosis usually needs error codes, field names and record keys, not clinical content. Identifiers are masked or tokenized where the task allows, and agent processing is subject to the organization's privacy and data-protection rules.
- *Untrusted message content.* Free-text fields in messages and logs may contain text that resembles instructions. Content from messages, logs and records must be treated strictly as data, never as instructions to the agent, to reduce the risk of prompt-injection style attacks. Indirect prompt injection, in which adversarial instructions are embedded in data retrieved by an LLM-integrated application, has been demonstrated against real-world systems [29] and is among the principal risks listed for LLM applications [30].
- *Audit, monitoring and stop.* All observations, reasoning summaries, tool calls and outcomes are logged. The agents are monitored for drift, error rates and unusual action patterns, and can be paused immediately.
- *Accountability and risk management.* A named human owner is accountable for each autonomy tier. The program can be organized around the Govern, Map, Measure and Manage functions of the NIST AI Risk Management Framework [20] and its Generative AI Profile [31] and a sociotechnical review of workflow and people [17].

Table VI consolidates the main threats and the corresponding controls in the framework, using the risk categories commonly applied to LLM applications [30].

TABLE VI
THREAT MODEL AND CONTROLS

Threat	Example in this setting	Controls
Indirect prompt injection	A free-text field in a product description or interface log contains text that looks like an instruction to the agent.	Message and log content is passed only as quoted data; tool arguments must come from validated structured fields; per-role tool allow-list; policy engine independent of the model.
Excessive agency	An agent applies a change outside its permitted class or scope.	Least privilege per role; tier = min(policy, confidence, criticality); deterministic policy engine; change control cannot be bypassed.
Unsupported or stale diagnosis	The agent cites a superseded build standard or invents a mapping.	Retrieval from the versioned standard only; mandatory citation of the passage; dry run; post-fix verification; escalation below the confidence threshold.
Sensitive data exposure	Patient identifiers appear in prompts or logs.	Minimization and tokenization of identifiers; processing inside the organization's approved environment; access-controlled audit logs.
Model drift or version change	A model update changes diagnoses.	Pinned model versions; regression suite of replayed historical cases before any upgrade; monitoring of false-fix and rollback rates.
Cascading or runaway changes	A faulty fix is applied repeatedly across many records.	Rate and blast-radius limits per class; idempotent replay; emergency stop; automatic rollback on failed verification.

VI. PROPOSED EVALUATION METHODOLOGY

Because the framework is a design artifact, we adopt the design-science research approach [18,19] and select evaluation episodes following the FEDS framework [32]: build, evaluate and refine the artifact against defined problem metrics. Evaluation proceeds in five phases with explicit gates (Fig. 5). Progression to the next phase requires pre-registered criteria for agreement and safety, and any harmful event pauses progression.

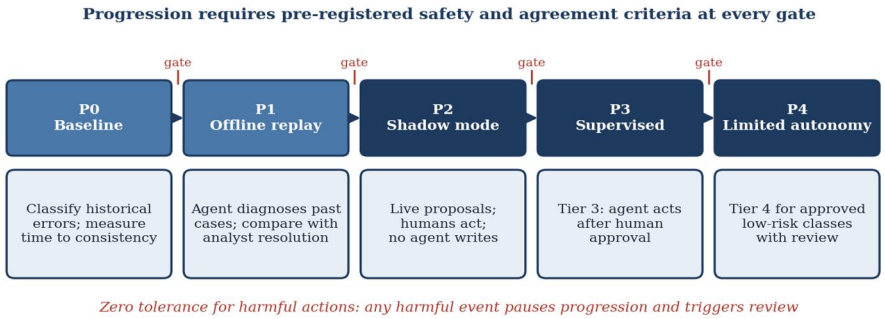


Fig. 5 Staged evaluation pathway with safety and agreement gates.

A. *Phases*

- *P0, baseline.* Retrospective analysis of historical interface errors and held transactions in one organization, classified using Table I, with measurement of TTC by class. Because few discrepancies reach formal incident reports [34,35], the baseline should be reconstructed from interface logs and held-transaction records rather than from reported incidents. Baseline measures should also include build lead time for routine and urgent requests, the share of requests escalated as urgent, and patient waiting time when a dispense or laboratory order is held.

- *P1, offline replay.* The agent diagnoses historical cases with the outcome hidden; its classification, root cause and proposed fix are compared with the analysts' recorded resolution.
- *P2, shadow mode.* The agent receives live events and proposes actions, while humans act as usual. No agent writes occur.
- *P3, supervised operation.* Tier 3 actions after explicit human approval, on a limited set of interfaces.
- *P4, limited autonomy.* Tier 4 for approved low-risk classes only (for example F3 and F4), with review of all actions.

Promotion of a failure class to a higher tier requires, in the preceding phase, (i) zero harmful actions, (ii) an upper 95% confidence bound for the false-fix rate below the bound pre-specified for that class, and (iii) diagnostic agreement at or above the pre-specified margin, each on a pre-specified minimum number of cases (Table VIII).

B. Metrics

TABLE VII
PROPOSED EVALUATION METRICS

Metric	Definition	Direction / role
Time to consistency (TTC)	Median and 95th percentile of time from creation to G(e,t) empty	Lower is better; primary
Held-transaction rate	Held or failed dependent transactions per 1,000 dependent transactions	Lower is better; primary
Diagnostic agreement	Agreement between agent and senior analysts on class and root cause	Non-inferiority margin pre-specified
Verifier performance	Precision and recall of discrepancies against a manually audited sample	Higher is better
False-fix rate	Proportion of executed fixes later judged incorrect or requiring correction	Within pre-specified bound; safety
Harmful-action count	Actions causing patient-safety, privacy or financial harm	Zero tolerance; gate
Rollback rate	Fixes rolled back after verification failure	Monitored
Analyst effort	Analyst minutes per incident and manual interventions per 1,000 entities	Lower is better
Patient waiting time	Minutes between a dependent transaction (for example a dispense or laboratory order) being held and its completion, where recorded	Lower is better; secondary
Build completeness at first use	Proportion of new items whose required local build is complete in every receiving system before the first dependent transaction	Higher is better; primary

C. Hypotheses and Analysis

- H1: Median TTC is lower in P3-P4 than at baseline.
- H2: The held-transaction rate is lower in P3-P4 than at baseline.
- H3: Agent diagnostic agreement with senior analysts is non-inferior to a pre-specified margin.
- H4: The false-fix rate does not exceed a pre-specified bound, and no harmful actions occur.
- H5: Analyst effort per incident is lower in P3-P4 than at baseline.

Analysis would use interrupted time series or a stepped, interface-by-interface rollout to separate the effect of the agents from secular trends, with results stratified by failure class and interface. Non-inferiority margins and safety bounds are fixed before data collection. Threats to validity include changes in case mix, concurrent vendor upgrades, learning effects among analysts, observer effects, and dependence on one organization's build standard, which limits generalizability. The study would use operational metadata rather than clinical content, and should be reviewed under the institution's quality-improvement or ethics process. Reporting of future empirical studies should draw on the relevant items of the DECIDE-AI guideline for early-stage evaluation of AI-driven systems [33], adapted because this system supports operational rather than clinical decisions.

Sample sizes for the safety gates. If no false fix is observed in n executed fixes, the exact one-sided 95% upper confidence bound for the false-fix rate is $1 - 0.05^{(1/n)}$; showing a bound p requires $n \geq \ln(0.05) / \ln(1 - p)$ fixes (the 'rule of three' approximates the bound as 3/n). Table VIII gives the minimum number of consecutive qualifying fixes for common bounds, allowing zero, one or two false fixes, computed from the exact binomial distribution. For diagnostic agreement, estimating an agreement of about 0.90 with a 95% confidence half-width of 0.05 requires about 139 independently adjudicated cases (normal approximation); a non-inferiority test against analysts would be sized from the pre-specified margin. Failure classes with low incident volumes may not accumulate these numbers within one organization, so their tier would stay at the default until a multi-site evidence base exists.

TABLE VIII
MINIMUM NUMBER OF EXECUTED FIXES NEEDED TO SHOW, WITH 95% CONFIDENCE (EXACT ONE-SIDED BINOMIAL), THAT THE FALSE-FIX RATE IS BELOW A GIVEN BOUND

Bound on false-fix rate	0 false fixes	At most 1	At most 2
Below 5%	59	93	124
Below 2%	149	236	313
Below 1%	299	473	628
Below 0.5%	598	947	1,258

VII. DISCUSSION

The framework reframes the integration error queue as an active, supervised remediation loop. Several points deserve emphasis.

Agents versus rules. Deterministic rules and robotic process automation already handle known, repetitive repairs, such as retry after a timeout (F4) or replay after a prerequisite exists (F3). The additional value of language-model agents lies in novel or ambiguous cases: reading an unfamiliar error, relating it to build documentation, explaining the cause in plain language, and drafting a mapping from specifications. We therefore recommend a hybrid in which a deterministic policy engine governs what the agent may do, and the agent supplies diagnosis and drafting.

Table IX positions the proposal against the approaches an organization would otherwise use. It is a qualitative characterization by the authors, not a systematic comparison of products.

TABLE IX
POSITIONING AGAINST EXISTING APPROACHES TO INTEGRATION ERROR REMEDIATION (QUALITATIVE)

Approach	Novel or ambiguous errors	Cross-system verification	Governed autonomy	Main limitation
Manual support analyst	Yes	Ad hoc	Not applicable	Slow; knowledge held by individuals; backlog grows.
Deterministic rules and RPA	No: known patterns only	Limited	Implicit in the script	Brittle; maintenance grows with every new pattern.
Interface-engine monitoring and alerting	No: detects and alerts	Rarely	Not applicable	Queue stays passive; no diagnosis or repair.
AIOps for IT operations [27]	Partly, from telemetry	No: not master-data aware	Varies	Targets infrastructure signals, not the business semantics of master data.
Proposed framework	Yes: grounded LLM diagnosis	Yes: independent verifier	Yes: tiers, gates, audit	Untested; depends on build-standard quality and vendor access.

Prevention and correction are complementary. Mistake-proofing at build time, in the spirit of poka-yoke [21], reduces failures by design: for example, a configurable function that prevents a medication from being dispensed from a location where it was never built can avoid hold transactions upstream (Samiyullaha [22]). Agentic remediation addresses the failures that remain. Hold transactions of that kind are instances of class F7 (incomplete local build) in the taxonomy. The framework does not depend on that work.

Evidence from stock reconciliation. Work on hospital pharmacy stock discrepancies supports two design choices in this framework. First, an AI layer placed over an unresolved data gap can mask rather than resolve upstream defects, for example by treating a chronic discrepancy as normal behavior [34]. This is why the framework grounds agents in the versioned build standard and keeps an independent verifier, instead of letting an agent learn what is normal from the error queue itself. Second, the same review calls for continuous rather than periodic reconciliation, named ownership of exception queues, and standing metrics of discrepancy rate and time to reconciliation [34]. These correspond to the continuous verification, the accountable human owner for each tier, and the time-to-consistency and held-transaction metrics proposed here. That review concerns physical stock rather than master data, is a narrative review screened by a single reviewer, and is by the same author as this paper, so it is treated as adjacent evidence and not as validation.

An overlay path. Organizations that cannot replace their application portfolio can add the agent layer over existing systems and integration engines. Systems that lack AI-ready interfaces risk being treated as legacy not because they fail, but because they cannot participate in this kind of supervised automation. How far agents can act depends on what each vendor exposes: where configuration is only accessible through proprietary screens, the practical ceiling is Tier 2 or 3, with the agent preparing the change and a person applying it.

A. Limitations

- *No empirical validation.* This is a conceptual design. The benefits and risks described are hypotheses until tested.
- *Dependence on the build standard.* The agent is only as good as the documentation it retrieves; incomplete or outdated standards will degrade diagnosis.
- *Model behavior.* LLM output is probabilistic and can change across model versions. The validation, tiering and verification layers are intended to contain, not eliminate, that risk.
- *Access constraints.* Vendor APIs, licensing and security policies may limit what agents can read or change.
- *Scope.* The framework addresses patient and product master data. Other entities (providers, locations, price lists) and clinical content are outside scope.
- *Illustrative example only.* The walk-through uses synthetic data and a single failure class. It demonstrates the mechanics of the loop, not its performance.
- *Literature coverage.* Related work is selective rather than systematic; a systematic review is planned.
- *Specification-level verification only.* The tier-decision rule was checked exhaustively as specified, but the agents, the language model, the confidence estimates and the integration with real systems were not tested.
- *Practitioner observations.* The build delays and patient waiting times described in Section I come from the author's experience and were not measured; they motivate the baseline phase and should not be read as findings.

VIII. CONCLUSION AND FUTURE WORK

Standards such as HL7 and FHIR move master data between systems, but the last step of making that data valid and usable in each receiving system often depends on manual diagnosis and repair. We have specified a governed, closed-loop framework in which AI agents act as integration architect, builder, verifier and support engineer, with autonomy tiered by failure class, verification built into every fix, and patient identity left to humans. The framework is designed to be overlaid on existing systems and to be evaluated in stages. The framework also checks for incomplete local build or configuration, which raises no interface error, so that a gap can be flagged before a patient is waiting. A worked example and a threat model show how the pieces fit together, and quantified safety gates define what evidence would justify moving from shadow mode to supervised and then limited autonomy. The principal limitation is the absence of empirical validation: the claims of this paper are design claims and hypotheses, and the staged protocol is intended to test them.

Future work includes: a pilot following the protocol in Section VI, beginning with calibration of the confidence thresholds and a comparison against a deterministic rule-based baseline; a shared, de-identified benchmark of interface-error cases for comparing agents; implementation-level and formal verification of the policy engine, beyond the specification-level check reported in Section IV-I; study of multi-agent coordination and failure modes; extension to additional master entities and to FHIR-native validation and subscription mechanisms; and a systematic literature review of agentic AI in healthcare operations and interoperability.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

IX. DECLARATIONS

1) *Ethics.* This conceptual study used no patient data and involved no human participants.
2) *Conflict of interest and funding.* [Author to complete. Consider declaring any relationship with vendors of the systems or knowledge bases named in the paper.]
3) *Use of AI assistance.* A generative AI assistant (Claude, Anthropic) was used to help with editing, with web searches for related work, and with writing the verification script described in Section IV-I. The author reviewed and edited the content and is responsible for it. [Author to confirm and adapt to the target journal's policy.]
4) *Data availability.* No datasets were generated or analyzed.

REFERENCES

[1] Bender D, Sartipi K. HL7 FHIR: an agile and RESTful approach to healthcare information exchange. In: Proceedings of the 26th IEEE International Symposium on Computer-Based Medical Systems (CBMS). IEEE; 2013. p. 326-31. doi:10.1109/CBMS.2013.6627810
[2] Mandel JC, Kreda DA, Mandl KD, Kohane IS, Ramoni RB. SMART on FHIR: a standards-based, interoperable apps platform for electronic health records. J Am Med Inform Assoc. 2016;23(5):899-908. doi:10.1093/jamia/ocv189
[3] HL7 International. FHIR Release 4 (v4.0.1). Available from: https://hl7.org/fhir/R4/ [accessed 22 October 2026].
[4] Hohpe G, Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Boston: Addison-Wesley; 2003.
[5] Wang RY, Strong DM. Beyond accuracy: what data quality means to data consumers. J Manag Inf Syst. 1996;12(4):5-33.
[6] Weiskopf NG, Weng C. Methods and dimensions of electronic health record data quality assessment: enabling reuse for clinical research. J Am Med Inform Assoc. 2013;20(1):144-51.
[7] Kahn MG, Callahan TJ, Barnard J, Bauck AE, Brown J, Davidson BN, et al. A harmonized data quality assessment terminology and framework for the secondary use of electronic health record data. eGEMs. 2016;4(1):1244. doi:10.13063/2327-9214.1244
[8] Fellegi IP, Sunter AB. A theory for record linkage. J Am Stat Assoc. 1969;64(328):1183-210.
[9] Reason J. Human error: models and management. BMJ. 2000;320(7237):768-70.
[10] Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, Cao Y. ReAct: synergizing reasoning and acting in language models. In: International Conference on Learning Representations (ICLR); 2023. arXiv:2210.03629.
[11] Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, et al. A survey on large language model based autonomous agents. Front Comput Sci. 2024;18(6):186345. doi:10.1007/s11704-024-40231-1
[12] Ji Z, Lee N, Frieske R, Yu T, Su D, Xu Y, et al. Survey of hallucination in natural language generation. ACM Comput Surv. 2023;55(12):Article 248.
[13] Singhal K, Azizi S, Tu T, Mahdavi SS, Wei J, Chung HW, et al. Large language models encode clinical knowledge. Nature. 2023;620(7972):172-80.
[14] Rajkomar A, Dean J, Kohane I. Machine learning in medicine. N Engl J Med. 2019;380(14):1347-58.
[15] Parasuraman R, Sheridan TB, Wickens CD. A model for types and levels of human interaction with automation. IEEE Trans Syst Man Cybern A Syst Hum. 2000;30(3):286-97. doi:10.1109/3468.844354
[16] Onnasch L, Wickens CD, Li H, Manzey D. Human performance consequences of stages and levels of automation: an integrated meta-analysis. Hum Factors. 2014;56(3):476-88.
[17] Sittig DF, Singh H. A new sociotechnical model for studying health information technology in complex adaptive healthcare systems. Qual Saf Health Care. 2010;19(Suppl 3):i68-74.
[18] Hevner AR, March ST, Park J, Ram S. Design science in information systems research. MIS Q. 2004;28(1):75-105.
[19] Peffers K, Tuunanen T, Rothenberger MA, Chatterjee S. A design science research methodology for information systems research. J Manag Inf Syst. 2007;24(3):45-77.
[20] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1. Gaithersburg (MD): NIST; 2023.
[21] Shingo S. Zero Quality Control: Source Inspection and the Poka-Yoke System. Cambridge (MA): Productivity Press; 1986.
[22] Samiyullaha S. A configurable, mistake-proofed function for adding and removing medications across hospital pharmacy inventory locations. Int J Novel Res Dev. 2026;11(9):b392-b400.
[23] HL7 International. HL7 Version 2.5.1 Messaging Standard (ANSI/HL7 V2.5.1-2007). Ann Arbor (MI): Health Level Seven International; 2007.
[24] Ayaz M, Pasha MF, Alzahrani MY, Budiarto R, Stiawan D. The Fast Health Interoperability Resources (FHIR) standard: systematic literature review of implementations, applications, challenges and opportunities. JMIR Med Inform. 2021;9(7):e21929. doi:10.2196/21929
[25] Christen P. Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Berlin: Springer; 2012.
[26] Narayan A, Chami I, Orr L, Arora S, Ré C. Can foundation models wrangle your data? Proc VLDB Endow. 2022;16(4):738-46. arXiv:2205.09911
[27] Notaro P, Cardoso J, Gerndt M. A survey of AIOps methods for failure management. ACM Trans Intell Syst Technol. 2021;12(6):Article 81. doi:10.1145/3483424
[28] Bedi S, Liu Y, Orr-Ewing L, Dash D, Koyejo S, Callahan A, et al. Testing and evaluation of health care applications of large language models: a systematic review. JAMA. 2025;333(4):319-28. doi:10.1001/jama.2024.21700
[29] Greshake K, Abdelnabi S, Mishra S, Endres C, Holz T, Fritz M. Not what you've signed up for: compromising real-world LLM-integrated applications with indirect prompt injection. In: Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23). ACM; 2023. p. 79-90. arXiv:2302.12173
[30] OWASP Foundation. OWASP Top 10 for Large Language Model Applications (2025). Available from: https://genai.owasp.org/llm-top-10/

[31] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. NIST AI 600-1. Gaithersburg (MD): NIST; 2024. doi:10.6028/NIST.AI.600-1

[32] Venable J, Pries-Heje J, Baskerville R. FEDS: a framework for evaluation in design science research. Eur J Inf Syst. 2016;25(1):77-89. doi:10.1057/ejis.2014.36

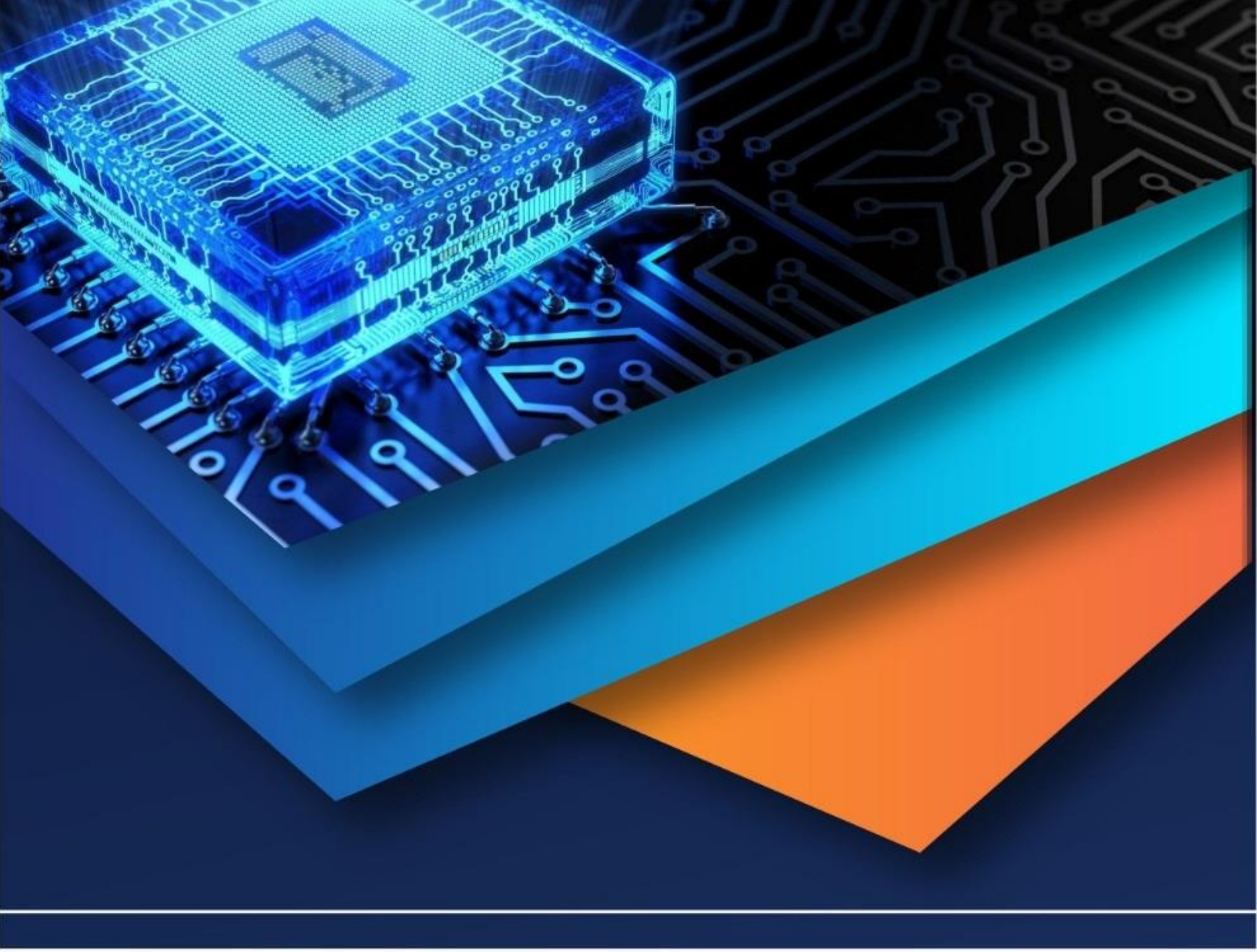
[33] Vasey B, Nagendran M, Campbell B, Clifton DA, Collins GS, Denaxas S, et al. Reporting guideline for the early-stage clinical evaluation of decision support systems driven by artificial intelligence: DECIDE-AI. Nat Med. 2022;28(5):924-33. doi:10.1038/s41591-022-01772-9

[34] Samiyullaha S. Persistent stock discrepancies in hospital pharmacy supply chains: why robotics, system integration, and artificial intelligence have not closed the reconciliation gap. Int J Sci Res Technol. 2026;3(10):80-7. doi:10.5281/zenodo.23116496

[35] Fong A, Kazi S, Kazanas M, Ratwani RM, Hettinger AZ. Estimating the frequency of automated dispensing cabinet discrepancy safety events using Markov models. Drugs Ther Perspect. 2022;38(3):146-55. doi:10.1007/s40267-022-00900-2

[36] Dhar M, Singh R, Manikonda SCK. From single chatbots to governed agent ecosystems: an agentic AI pattern catalogue and orchestration framework for mission-critical hospital information management systems. Int J Innov Sci Res Technol. 2026;11(7):1851-84. doi:10.38124/ijisrt/26jul830

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)