ISSN: 2096-3246
Volume 58, Issue 03, 2026

AUTOGENIUS: AN AI-DRIVEN, EVENT-DRIVEN FRAMEWORK FOR END-TO-END ERP TRANSACTION VALIDATION AND OBSERVABILITY AT ENTERPRISE SCALE

Sridhar Dachepelly
Independent Researcher, USA
ORCID ID: 0009-0005-9478-4831

Abstract
Enterprise resource planning (ERP) transformation programmes generate exponential scenario complexity that conventional testing approaches are structurally unable to handle. As organisations migrate to cloud-native ERP platforms and connect diverse upstream source systems, the number of valid transaction combinations routinely exceeds the capacity of manual, rule-based, and module-focused automation tools. This article presents AutoGenius, an AI-driven, event-driven framework for end-to-end ERP transaction validation and observability at enterprise scale. The framework adopts a plug-and-play, ERP-agnostic architecture built on asynchronous event streaming, loose coupling, and horizontal scalability, enabling transaction throughput to grow without degradation. AutoGenius provides complete lifecycle tracking from order creation through fulfilment and billing, with context-rich failure diagnostics anchored by system-level correlation identifiers. An RAG layer for defect correlation against existing issue trackers and a hybrid conversational interface for failure diagnosis and order tracking implements 'automated business validation testing (BVT)', making production-ready status available within any 24 hour runtime period between maintenance windows or scheduled outages. The framework design is evaluated against established software quality criteria and compared with existing testing paradigms, demonstrating superior coverage breadth, failure traceability, and scalability. AutoGenius offers a replicable architectural pattern for organisations undertaking large-scale ERP system transformation
Keywords: Enterprise Resource Planning Validation, Test Automation Architecture, Event-Driven Framework, AI-Augmented Testing, Software Observability, End-to-End Quality Assurance

1. Introduction
Large-scale enterprise resource planning (ERP) transformations rank among the most complex software migration programmes undertaken by modern organisations. When an enterprise migrates from a legacy ERP to a cloud-native platform—spanning quoting, ordering, fulfilment, billing, taxation, logistics, and downstream analytics—the scope of required validation expands far beyond what testing teams have historically managed [1]. Each product line, pricing tier, regional configuration, and regulatory variant contributes multiplicatively to the total scenario space, producing test matrices that can reach hundreds of thousands of distinct transaction combinations within a single enterprise programme.

Conventional testing approaches are structurally unsuited to this challenge. Manual testing teams cannot execute, verify, and re-execute at the required volume within programme timelines. Rule-based automation frameworks, while effective at the module level, do not provide visibility into cross-system transaction lifecycles, cannot adapt to new ERP configurations without substantial re-engineering effort, and do not produce actionable diagnostics when failures occur in downstream systems [2]. End-to-end observability is not available, and testers cannot differentiate between problems related to configuration, integration, and data quality without conducting a time-consuming manual review process.

The gap this article addresses is the absence of a scalable, ERP-agnostic, production-grade validation framework that delivers end-to-end transaction coverage, failure traceability, and continuous health monitoring within a single architectural boundary. Existing solutions address these concerns only in part: some tools provide test execution at scale but not observability; others offer dashboards but not automated root-cause correlation. No widely documented framework integrates asynchronous event-driven execution, AI-augmented diagnostics, and continuous business validation into a single plug-and-play platform reusable across transformation programmes.

This article presents AutoGenius, a framework built to fill this gap. AutoGenius is an ERP-agnostic, event-driven platform capable of executing and validating complete business transaction lifecycles across heterogeneous ERP environments. Its architecture accommodates unlimited transaction volumes through horizontal scalability, provides real-time lifecycle tracking with context-rich failure diagnostics, and incorporates RAG-based AI to correlate failures with known defect records. A hybrid conversational interface further reduces the diagnostic burden on testing and operations teams.

The remainder of this article is organised as follows. Section 2 surveys related work in test automation, continuous testing, and AI-augmented quality assurance. Section 3 characterises the problem domain and its design requirements. Section 4 presents the AutoGenius architectural design. Section 5 describes core validation and observability capabilities. Section 6 details AI-driven enhancements. Section 7 covers the technology stack and deployment model. Section 8 evaluates the framework. Section 9 discusses implications and limitations. Section 10 concludes.

2. Background and Related Work

Software test automation for enterprise systems has matured considerably over the past decade. Early frameworks focused on scripted test execution at the user interface or application programming interface level, providing repeatable coverage for isolated modules [2]. This approach addressed repeatability and speed for individual components but did not extend to multi-system transaction lifecycles. The challenge of end-to-end testing in distributed architectures has received growing scholarly attention as organisations have shifted from monolithic platforms to microservice-based systems. A systematic mapping study analysing 93 primary studies drawn from a pool of 19,595 research publications identified nine distinct categories of microservices testing methods, confirming that cross-boundary end-to-end testing presents fundamentally different challenges from component-level approaches [3]. Regression test selection in microservice-based systems compounds this difficulty because per-test execution traces do not map cleanly across distributed service boundaries [4].

The integration of continuous testing into DevOps delivery pipelines has shifted quality assurance from a pre-release gate to a sustained practice embedded in the software lifecycle [5]. The DevOpRET approach demonstrates that operational profile-based reliability testing can be incorporated into acceptance stages before each release, producing accurate and efficient reliability estimates across successive DevOps cycles [16]. It follows from this premise that recent work on maintainable hybrid automation frameworks gives evidence that the combination of structured test design patterns and continuous integration tooling leads to meaningful improvement of test reliability and code quality [8] (this is an architectural problem, not a procedural one).

AI is affecting test automation and other stages of the software testing lifecycle. A survey of 76 industrial tools and peer-reviewed studies published between 2020 and 2025 found that LLMs and RL agents are gradually replacing heuristic-based methods for test case generation, execution repair, and test maintenance [7]. At the industrial scale, simulation-driven end-to-end test generation at Meta demonstrated that automated oracle inference—combining production observation with integrity constraint modelling—can produce test suites without manual test authorship [9]. These developments confirm that AI-augmented validation is operationally feasible in high-volume enterprise contexts.
Despite this body of work, no published framework integrates ERP-agnostic plug-and-play deployment, asynchronous event-driven execution, AI-augmented failure diagnostics, and continuous business validation within a single architectural design. The literature on ERP-specific test automation remains largely practitioner-oriented, focusing on individual tools or migration patterns rather than replicable architectures [1]. AutoGenius addresses this gap by synthesising event-driven design principles, AI-augmented diagnostics, and continuous observability into a single deployable platform with a design sufficiently generic to transfer across ERP programmes.

3. Problem Domain: Challenges in Large-Scale ERP Validation

The challenge of validating large-scale ERP transformations arises from three converging structural factors: exponential scenario growth, multi-system integration depth, and zero-tolerance production requirements. When an organisation operates across multiple product lines, markets, pricing regions, and order channels, the combinatorial space of valid transaction scenarios grows multiplicatively. A

single configuration change—such as a pricing rule update or a regulatory tax reclassification—can propagate across hundreds of dependent scenario branches, each requiring independent validation before production readiness can be confirmed [1]. Under these conditions, manual execution schedules are incompatible with programme timelines regardless of team size.

Multi-system integration substantially compounds this complexity. Modern ERP landscapes are ecosystems of interconnected platforms spanning commercial market channels, partner systems, logistics providers, billing engines, and regulatory compliance layers. A transaction that passes validation within the ERP core may fail silently in a downstream fulfilment or taxation system, producing incorrect output without surfacing an error at the point of origin. Frameworks that do not track transaction state across all system boundaries cannot detect these silent failures, leaving programmes exposed to production incidents that validation was intended to prevent. Research on microservices testing confirms that cross-boundary observability gaps remain one of the most persistently unsolved challenges in distributed system validation practice [3].

Table 1. Comparison of ERP Validation Approaches: Manual, Rule-Based Automation, and AutoGenius

Scenario Scale	Limited (hundreds)	Moderate (thousands)	Unlimited (event-driven)
ERP Agnosticism	System-specific	Partial	Full (plug-and-play)
End-to-End Coverage	Incomplete	Module-level only	Complete (quote to billing)
Failure Traceability	Manual investigation	Partial (log-based)	Context-rich with correlation IDs
AI-Augmented Diagnostics	None	None	RAG-based defect correlation + chatbot
Continuous Monitoring	None	Scheduled runs only	24x7 Business Validation Testing

Operational constraints introduce a third layer of challenge. Programme timelines are set by business commitments, and production failures carry financial and reputational consequences that compress risk tolerance. When failure diagnostics are incomplete, testing teams must coordinate manually across functional, technical, and support domains to identify root causes—a process that adds latency structurally incompatible with the pace of large-scale ERP programmes [2]. Together, these three factors define the design requirements that AutoGenius was built to satisfy: unlimited scenario execution capacity, complete lifecycle visibility, context-rich failure attribution, and continuous health monitoring before and after production cutover.

4. AutoGenius Framework Architecture

AutoGenius is structured around three interrelated architectural principles: ERP agnosticism, event-driven execution, and built-in observability. The ERP-agnostic principle is realised through an abstraction layer that separates scenario definition from system-specific communication protocols. This allows new ERP environments—whether commercial or custom—to be onboarded through configuration rather than code modification, enabling the framework to be reused across successive transformation programmes without rebuilding core validation logic. The practical consequence is that organisations can amortise the investment in validation infrastructure across multiple programme cycles.

The event-driven execution model replaces the synchronous, polling-based approaches common in conventional test automation with asynchronous event propagation. For example, an order is created, processed in the ERP, fulfilled, billed and taxed as a sequence of events passing through a stream-based pipeline. This design eliminates the throughput ceilings and waiting periods that arise when test execution is coupled to synchronous API call chains, allowing the framework to scale horizontally by adding execution capacity without restructuring the pipeline [11]. When compared against API-driven architectures, event-driven designs have been shown to reduce end-to-end response time by 19.18%

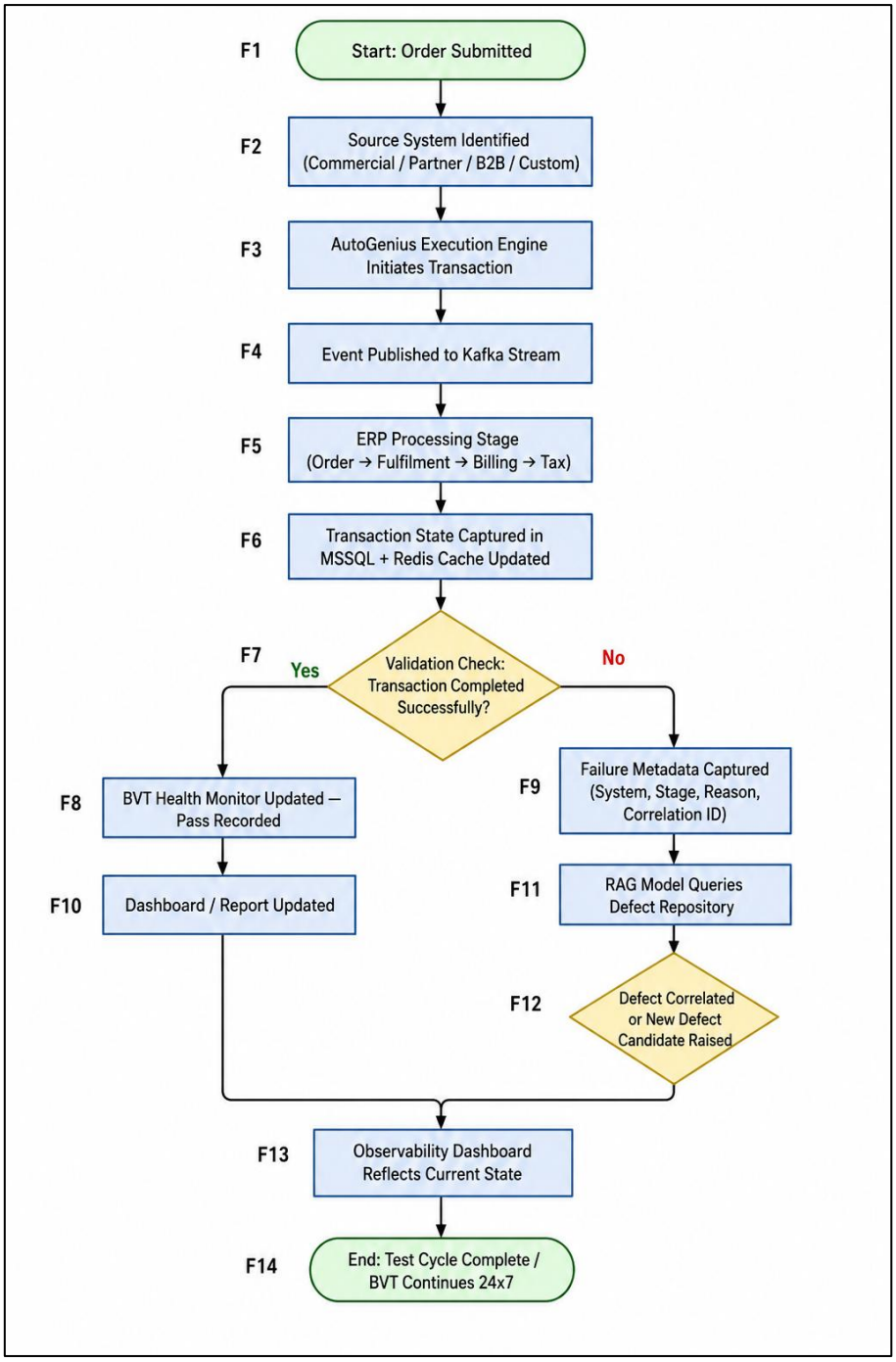
and reduce error rates by 34.40% under equivalent workloads [11], confirming that horizontal scaling becomes a function of infrastructure provisioning rather than architectural change.

Table 2. AutoGenius Architecture Components and Their Roles

Execution Engine	Python / Java / C#	Language-independent scenario execution and orchestration
Event Streaming Layer	Apache Kafka	Asynchronous event propagation across ERP pipeline stages
In-Memory Cache	Redis	High-speed caching for transaction state retrieval performance
Persistent Data Store	MSSQL	Centralised transaction state storage and audit history
Deployment Platform	PCF / KOB	Cloud-native container deployment and elastic orchestration
AI Layer	RAG model + Hybrid Chatbot	Defect correlation, failure diagnosis, order status queries

Loose coupling between components is enforced throughout the architecture. Each ERP subsystem—fulfilment, logistics, billing, monitoring—communicates with the framework through well-defined event contracts rather than direct service dependencies. This pattern, consistent with established recommendations for scalable event-driven architectures [12], ensures that changes in any individual ERP subsystem do not propagate breaking changes to the validation framework. State consistency across the distributed pipeline is maintained through a centralised shared database, with Redis providing high-throughput in-memory caching at stages where repeated state lookups would otherwise introduce latency at scale [10].

ISSN: 2096-3246
Volume 58, Issue 03, 2026



Flowchart 1. AutoGenius End-to-End Transaction Lifecycle

The container-native execution model also supports Pivotal Cloud Foundry (PCF) and Kubernetes-based container orchestration platforms (KOB). It allows elastic scaling, the ability of a system to automatically increase its capacity to execute based on the demand without changing the underlying infrastructure. To ease language-agnostic scenario logic that can be implemented or tailored by an engineering team in a language they already know, the execution engine is implemented in Python, Java and C#. Large enterprise programmes typically involve multiple development teams with heterogeneous capabilities and a validation platform that imposes a single language will both obstruct adoption and constrain coverage.

5. Core Validation and Observability Capabilities

The validation layer of AutoGenius provides coverage across the complete ERP transaction lifecycle, from order creation through upstream source systems to fulfilment, billing, and downstream reporting.

Order generation is supported across commercial market platforms, partner and marketplace channels, custom enterprise applications, and digital business-to-business interfaces. This breadth of source system coverage matters because failures that arise in specific channel configurations—particularly in regional or partner-specific pricing paths—are frequently invisible to frameworks that generate orders from a single, idealised source. Faithfully replicating the diversity of real transaction flows is a prerequisite for meaningful end-to-end validation.

When a failure occurs, the framework captures system-level metadata, lifecycle stage, error reason, and a correlation identifier that traces the failure across all systems the transaction has traversed. This reduces investigation time by providing testing teams with contextual guidance on which support or functional team should be engaged, removing the need for manual triage across system logs. The approach is consistent with observability-oriented microservice architectures that use logs, traces, and metrics as the foundational data triad for system health monitoring [14]. Distributed tracing has been validated as a reliable mechanism for attributing failures to their originating system and lifecycle stage in complex multi-service deployments [13].

Table 3. AutoGenius Observability Features and Their Engineering Significance

Real-Time Transaction Tracking	Visibility across all lifecycle stages	Eliminates blind spots in multi-system ERP flows
Failure Diagnostics	System, stage, reason metadata with correlation IDs	Enables targeted remediation without manual root-cause analysis
Business Validation Testing (BVT)	24x7 continuous transaction execution	Maintains production readiness as a continuous state, not a one-time gate
Dashboard Reporting	Batch, order, region, date-range filters	Empowers business teams to self-serve diagnostics without technical support
Trend Analysis	Recurring failure pattern detection and bottleneck identification	Supports proactive maintenance planning across ERP programmes

In both cases, the reporting layer and dashboard provide for queries on the dimensions of batch number, sales order number, date range, offer identifier, region, and country. Predefined time ranges of last 24 hours and last seven days are built-in for operational dashboards to avoid creating a query for each report. Advanced reporting provides script-level failure diagnosis, reporting on blocked transactions falling outside their expected processing time windows, and identifying trends in repeated failures across test batches. These can turn the observability layer into an anticipatory quality management mechanism as opposed to being a mere observation and diagnosis tool.

Business validation testing (BVT) extends the framework's utility beyond pre-production validation into continuous production readiness monitoring. BVT executes a representative transaction set continuously across the 24-hour operational cycle, maintaining real-time visibility into the preceding four days of application health and activity. This execution model aligns with the DevOpRET principle that operational profile-based testing should operate as a sustained practice integrated into the delivery lifecycle rather than as a one-time production gate [16]. The practical outcome is that ERP transformation programmes can maintain a continuously confirmed state of production readiness across post-go-live maintenance cycles, reducing the risk that incremental system updates introduce regression failures.

6. AI-Driven Enhancements: RAG Integration and Intelligent Diagnostics

The AI layer of AutoGenius comprises two components that address distinct aspects of the failure management lifecycle. The first is a retrieval-augmented generation (RAG)-based integration with issue tracking systems that automates the correlation of failed transactions with existing defect records. When a failure is detected, the RAG model queries the defect repository using failure metadata—system, stage, error reason, and affected transaction attributes—as retrieval keys. Matching failures are automatically associated with existing records, providing testing and support teams with immediate context on known issues without manual search. Where no match exists, the failure is surfaced as a candidate for new defect creation, preventing the duplicate ticket proliferation that commonly characterises large-scale testing programmes [7].

The second component is a hybrid conversational interface that combines language model reasoning, structured database query, and live ERP application programming interface calls. Users can retrieve order status, interpret failure messages, and receive platform guidance through natural language queries. The hybrid architecture grounds responses in real-time system state rather than static knowledge, which is particularly valuable in ERP environments where transaction states evolve continuously. This design is consistent with transformer-based conversational AI architectures that have demonstrated effectiveness in enterprise support contexts, where few-shot contextual adaptation reduces the training data required for domain-specific deployment [15].

Together, these two AI components reduce the manual overhead of failure triage and defect management in large-scale testing programmes. By automating defect correlation and providing a natural-language diagnostic interface, AutoGenius reduces mean time to resolution without requiring specialised technical skills from all team members. Business owners, product managers, and functional testers can interact with the system directly, decoupling diagnostic capability from the availability of dedicated testing engineers [1][15]. This democratisation of observability data reflects a broader architectural trend toward conversational interfaces that lower the barrier to enterprise system interaction across diverse organisational roles.

7. Scalability, Deployment, and Technology Stack

The scalability of the AutoGenius framework is an endogenous property of the architecture. This horizontal scalability is achieved by routing transaction lifecycle events through the Apache Kafka message broker, without requiring changes to upstream or downstream system integrations. Kafka's use of data partitioning allows throughput to be increased by adding new nodes (brokers), allowing for higher amounts of events to be processed at low latency under production workloads [6]. For ERP programmes requiring sustained high-volume execution—particularly during regression cycles immediately preceding production cutover—this capacity can be expanded through infrastructure provisioning alone, without architectural changes to the framework.

The Redis caching layer addresses performance at transaction state retrieval points where repeated database access would introduce latency under scale. Research on multi-tenant in-memory key-value cache partitioning has demonstrated throughput improvements of up to 262.8% over standard in-memory storage architectures under multi-tenant workloads, confirming the material performance contribution of optimised caching strategies in distributed validation systems [10]. For a framework executing large numbers of concurrent transaction scenarios across multiple ERP workstreams simultaneously, the caching layer is a critical contributor to consistent response time. The centralised MSSQL data store provides durable, indexed persistence for transaction history, correlation identifiers, and audit records, supporting both real-time querying and retrospective programme analysis.

Table 4. AutoGenius Technology Stack Components, Roles, and Scalability Properties

Event Streaming	Apache Kafka	Horizontal partition-based throughput scaling; high-volume event processing at low latency [6]
Caching	Redis (kRedis variant)	Up to 262.8% throughput improvement over standard in-memory stores under multi-tenant workloads [10]
Execution Engine	Python / Java / C#	Language-agnostic; enables team-independent horizontal scale-out
Deployment	PCF / KOB	Container-native autoscaling; elastic resource provisioning under load
Persistent Store	MSSQL	Indexed correlation IDs; audit-safe transaction log with retrospective query support

Deployment flexibility is preserved through support for both Pivotal Cloud Foundry and Kubernetes-based container orchestration environments. Container-native deployment models provide elastic resource provisioning, enabling execution capacity to scale in response to programme demand without manual infrastructure intervention. The language-independent execution engine and its Python, Java,

and C# implementations enable engineering teams to avoid being locked into a single language ecosystem when extending scenario coverage or integrating with new ERP subsystems. This reflects the practical reality that large enterprise transformation programmes typically involve multiple development teams, and frameworks that impose language constraints create adoption barriers that limit coverage breadth over the programme lifecycle.

8. Evaluation: Design Validity and Comparative Analysis

The AutoGenius framework is evaluated against five quality criteria drawn from software architecture practice: coverage breadth, failure traceability, integration depth, scalability, and maintainability. Coverage breadth measures the proportion of valid transaction scenarios the framework can represent and execute. AutoGenius addresses this through its ERP-agnostic abstraction layer and event-driven execution model, which impose no structural ceiling on scenario count. Rule-based automation frameworks require explicit scenario encoding, creating a maintenance burden that grows proportionally with scenario complexity—a limitation well documented in enterprise test automation research [2]. End-to-end benchmark studies in microservice-based systems confirm that scenario coverage depth remains a persistent challenge for distributed architectures where transaction diversity across channels and regions is high [17].

Failure traceability is assessed against the criterion that every failed transaction should be attributable to a specific system, lifecycle stage, and error condition without manual cross-system investigation. AutoGenius satisfies this through its correlation identifier model, which propagates a unique trace token through all lifecycle stages of each transaction. This approach is consistent with observability-oriented tracing practices documented in microservice observability research that uses distributed trace data as the primary failure attribution mechanism [13][14]. Rule-based and manual approaches produce failure records at the scenario level only, requiring subsequent investigation to identify root systems and conditions—an overhead that accumulates significantly across large-scale ERP validation programmes.

The scalability of a framework is determined by the extent to which the framework's execution-throughput and failure-detection-accuracy are maintained under increasing transaction load. AutoGenius employs an event-driven architecture that follows the Kafka horizontal-partition replication model [12] for networks that depend on data streaming. Unlike polling-based and synchronous architectures, for which throughput depends on the latency of the slowest constituent system in the integration pipeline, AutoGenius uses loose coupling to decouple execution throughput from any single sub-system's latency. Consequently, it is able to continue to maintain pipeline velocity, even whilst the downstream ERP system may be temporarily slowed down [11]. Likewise, reinforcement learning end-to-end testing frameworks have demonstrated this architectural decoupling at the application layer, extending the generalization of architectural independence as a scalability enabler across testing models [18].

9. Discussion

AutoGenius makes two principal contributions to the enterprise software architecture literature. First, it demonstrates that end-to-end ERP validation at scale can be achieved through event-driven architectural design rather than through infrastructure over-provisioning, reframing scalability as an architectural property rather than a capacity management problem. Second, it establishes that AI-augmented diagnostics can function as a first-class architectural capability—not an optional post-processing add-on—when retrieval-augmented generation is embedded at the point of failure detection. These contributions extend existing research on continuous testing architectures [5][16] and AI-driven test automation surveys [7] by situating their findings within the specific operational constraints of large-scale ERP transformation programmes.

Several design limitations warrant acknowledgement. The RAG-based defect correlation component depends on the quality and consistency of the defect repository it queries; organisations with poorly maintained issue trackers may observe lower correlation accuracy than those with well-structured defect records. The ERP-agnostic abstraction layer requires initial configuration when onboarding a new ERP system, which, while substantially lower-cost than building a bespoke validation framework, represents a non-trivial setup investment for first deployments. Additionally, the BVT

continuous monitoring capability generates a persistent stream of execution data whose storage and querying costs at scale have not been empirically modelled in the current design; longitudinal cost studies post-go-live would provide a more complete economic picture of the framework's lifecycle value.

Future research directions include the integration of predictive failure analytics that use trend data to forecast failure-prone scenarios before execution, and the extension of the RAG defect correlation model to support multi-repository aggregation across distributed programme teams. The applicability of the AutoGenius architectural pattern to domains beyond ERP—such as large-scale financial systems integration and regulated healthcare platform migration—represents a further empirical validation opportunity. Comparative benchmarking studies against commercial ERP testing tools using standardised scenario complexity metrics would provide a stronger quantitative foundation for the framework's scalability and coverage claims.

10. Conclusion

This article has presented AutoGenius, an AI-driven, event-driven framework for end-to-end ERP transaction validation and observability at enterprise scale. The framework addresses structural limitations inherent in conventional testing approaches—including bounded scenario execution capacity, the absence of cross-system failure traceability, and the lack of continuous health monitoring—through four architectural contributions: an ERP-agnostic plug-and-play design; asynchronous event-driven execution with horizontal scalability; context-rich failure diagnostics anchored by correlation identifiers; and AI-augmented defect correlation supported by a conversational diagnostic interface. These capabilities collectively provide organisations with a validation platform that retains operational value not only during ERP transformation programmes but across the ongoing post-go-live maintenance lifecycle.

The design principles underlying AutoGenius—loose coupling, event-driven processing, AI integration at the point of failure detection, and continuous observability—are consistent with established guidance in the software architecture and quality assurance literature. The framework extends these principles into the specific context of large-scale ERP validation, demonstrating that enterprise-grade validation capability can be packaged as a replicable architectural pattern rather than a bespoke implementation. This replicability is a meaningful practical contribution for organisations undertaking ERP transformation, where validation infrastructure costs represent a recognised and recurring programme risk.

The broader significance of AutoGenius lies in its reframing of ERP validation as a continuous, platform-level capability rather than a periodic pre-release activity. By sustaining 24-hour business validation testing, automated failure attribution, and AI-assisted diagnostics as persistent operational services, the framework positions quality assurance as an enduring enterprise asset—a component of the intelligent enterprise infrastructure that continuous integration practices and AI-augmented operations are collectively making possible.

Reference

1. M. A. Mannan, A. K. Shaikh, A. A. Khan, K. Raahemifar, and M. A. Mohamed, "Transforming ERP systems with collaborative AI: Paving the path to strategic growth and sustainability," Array, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590005625001444
2. E. Celik et al., "Software test automation and a sample practice for an enterprise business software," in Proc. IEEE Int. Conf. Software Quality, Reliability and Security Companion (QRS-C), 2017, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/8093583
3. M. Hui et al., "Unveiling the microservices testing methods, challenges, solutions, and solutions gaps: A systematic mapping study," Journal of Systems and Software, vol. 220, p. 112232, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0164121224002760
4. D. Elsner, D. Bertagnolli, A. Pretschner, and R. Klaus, "Challenges in regression test selection for end-to-end testing of microservice-based software systems," in Proc. 3rd ACM/IEEE Int. Conf. Automation of Software Test (AST), 2022, pp. 55–65. [Online]. Available: https://ieeexplore.ieee.org/document/9796458

5. R. Pietrantuono, A. Bertolino, G. De Angelis, B. Miranda, and S. Russo, "Towards continuous software reliability testing in DevOps," in Proc. 14th Int. Workshop Automation of Software Test (AST), 2019, pp. 21–27. [Online]. Available: https://ieeexplore.ieee.org/document/8822025/
6. K. Padmanaban et al., "Apache Kafka on big data event streaming for enhanced data flows," in Proc. 8th Int. Conf. I-SMAC (IoT in Social, Mobile, Analytics and Cloud), 2024, pp. 977–983. [Online]. Available: https://ieeexplore.ieee.org/document/10714884/
7. A. Faraji and N. Pombo, "AI-driven software test automation: An AI4SE-oriented survey of techniques, tools, and challenges," IEEE Access, vol. 13, pp. 183296–183313, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/11208623/
8. A. R. Patel and S. Tyagi, "Enhancing software quality of CI/CD pipeline through continuous testing: a DevOps-driven maintainable hybrid automation testing framework," International Journal of Information Technology, vol. 17, pp. 4119–4133, 2025. [Online]. Available: https://link.springer.com/article/10.1007/s41870-025-02578-x
9. S. Tuli, K. Bojarczuk, N. Gucevska, M. Harman, X. Wang, and G. Wright, "Simulation-driven automated end-to-end test and oracle inference," in Proc. IEEE/ACM 45th Int. Conf. Software Engineering in Practice (ICSE-SEIP), 2023, pp. 122–133. [Online]. Available: https://ieeexplore.ieee.org/document/10172701/
10. Y. Wang, J. Yang, and Z. Wang, "Multi-tenant in-memory key-value cache partitioning using efficient random sampling-based LRU model," IEEE Transactions on Cloud Computing, vol. 11, no. 4, pp. 3801–3814, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10205957/
11. A. Rahmatulloh, F. Nugraha, R. Gunawan, and I. Darmawan, "Event-driven architecture to improve performance and scalability in microservices-based systems," in Proc. Int. Conf. Advancement in Data Science, E-learning and Information Systems (ICADEIS), 2022, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10037390/
12. T. P. Raptis and A. Passarella, "A survey on networked data streaming with Apache Kafka," IEEE Access, vol. 11, pp. 85333–85350, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10213406/
13. Z. Purfallah Mazraemolla and A. Rasoolzadegan, "An effective failure detection method for microservice-based systems using distributed tracing data," Engineering Applications of Artificial Intelligence, vol. 133, p. 108558, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0952197624007164
14. U. Faseeha, H. J. Syed, F. Samad, S. Zehra, and H. Ahmed, "Observability in microservices: An in-depth exploration of frameworks, challenges, and deployment paradigms," IEEE Access, vol. 13, pp. 72011–72039, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/10967524/
15. M. Ahmed, H. U. Khan, and E. U. Munir, "Conversational AI: An explication of few-shot learning problem in transformers-based chatbot systems," IEEE Transactions on Computational Social Systems, vol. 11, no. 2, pp. 1–19, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10153715/
16. A. Bertolino, G. De Angelis, A. Guerriero, B. Miranda, R. Pietrantuono, and S. Russo, "DevOpRET: Continuous reliability testing in DevOps," Journal of Software: Evolution and Process, vol. 35, no. 5, p. e2298, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2298
17. S. Smith et al., "Benchmarks for end-to-end microservices testing," in Proc. IEEE Int. Conf. Service-Oriented System Engineering (SOSE), 2023, pp. 60–66. [Online]. Available: https://ieeexplore.ieee.org/document/10254752/
18. C. Paduraru, M. Cristea, and A. Stefanescu, "End-to-end RPA-like testing using reinforcement learning," in Proc. IEEE Int. Conf. Software Testing, Verification and Validation (ICST), 2024, pp. 419–429. [Online]. Available: https://ieeexplore.ieee.org/document/10638615/